Dependable and Scalable Public Ledger for Policy Compliance, a Blockchain Based Approach

Zhou Wu
Marquette University

DEPENDABLE AND SCALABLE PUBLIC LEDGER FOR POLICY COMPLIANCE A BLOCKCHAIN BASED APPROACH

by

Zhou Wu, B.A., M.A.

A Dissertation submitted to the Faculty of the Graduate School,
Marquette University,
in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy

Milwaukee, Wisconsin

May 2020

ABSTRACT
DEPENDABLE AND SCALABLE PUBLIC LEDGER FOR POLICY COMPLIANCE A BLOCKCHAIN BASED APPROACH

Zhou Wu

Marquette University, 2020
MSCS

Policies and regulations, such as the European Union General Data Protection Regulation (EU GDPR), have been enforced to protect personal data from abuse during storage and processing. We design and implement a prototype scheme that could 1) provide a public ledger of policy compliance to help the public make informative decisions when choosing data services; 2) provide support to the organizations for identifying violations and improve their ability of compliance. Honest organizations could then benefit from their positive records on the public ledger. To address the scalability problem inherent in the Blockchain-based systems, we develop algorithms and leverage state channels to implement an on-chain-hash-off-chain data structure. We identify the verification of the information from the external world as a critical problem when using Blockchains as public ledgers, and address this problem by the incentive-based trust model implied by state channels. We propose the Verifiable Off-Chain Message Channel as the integrated solution for leveraging blockchain technology as a general-purpose recording mechanism and support our thesis with performance experiments. Finally, we suggest a sticky policy mechanism as the evidence source for the public ledger to monitor cross-boundary policy compliance.

ACKNOWLEDGEMENTS

Zhou Wu

I would like to express my heartfelt thanks to my advisor, Debbie Perouli, for her invaluable guidance and mentorship. I would like to thank my faculty mentors, my thesis committee, and all of my academic collaborators (in alphabetical order): Iqbal Ahamed, Dennis Brylow, Xizhou Feng, Rong Ge, Satish Puri, and Andrew Williams. I would also like to express my deep appreciation for the friends and colleagues: Dayeon An, Joe Coelho, Haochen Sun, and Xuyong Yu.

I would like to thank my family and especially my wife, Mingqing Xiao, who supported me with endless patience and love, without whom I could not endure throughout my Ph. D..

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

In current networked services, personal data is collected by the service providers for analysis, acquiring insights of the market, customized service provision, etc. The users may upload data to online service for storage or sharing information within community (e.g., Facebook, Dropbox). Because of the growing volume of personal data exposed on the Internet (either passively or actively), the protection against data abuse has become highly critical. Regulations such as European Union General Data Protection Regulation (GDPR) were enforced to protect personal data from abuse during storage and processing. However, the public is hardly able to know whether the organizations who process their data are properly compliant with the policies. Disclosure of violations are usually in the form of breaking scandals. Data abuse or breaches take place not just in untrustworthy organizations. Unfortunately, well-known cases of data leakage or misuse are related to famous names of companies that are considered responsible organizations by the society (e.g. Facebook-Cambridge Analytica data scandal). Even worse, organizations usually react to such events passively, concealing the truth from the sight of the public. In 2017, Yahoo admitted 3 billion user accounts had been compromised in their data breach scandal which was disclosed in 2013, but at the moment of the exposure, they only announced 500 million users impacted.

While the news of data leakage build up the mental tension of the public, there is no easy solution to achieve absolute security as the public may desire. Therefore, a more feasible way is to make the data usage transparent and auditable so that the users could make informative decisions when choosing a service. From the experience of other industry sectors, transparency would only benefit an area rather than undermine its credibility, though organizations may hesitate to do so at the beginning. Transparency allows the users more reaction time to limit the

damage; and if auditable records exist, the users could measure and compare the risks of choosing different service providers. On the other hand, the organizations could benefit from a relatively positive record; and the records could serve as evidence to which the organizations are able to price their effort on data protection accordingly.

Therefore, a transparent and auditable public ledger recording the organization's performance of data protection seems desirable not only to the research society but also to the industry sectors in which information technology is a significant building block. To record policy compliance is a direct approach. It may seem intuitive at first glance, however, to build an applicable scheme is not a trivial task, due to the complexity of the current computing environment. Over the last few years, we have observed a rapid increase on the complexity of data usage and transmission patterns, which are characterized by broaden categories of devices, cross-platform data sharing, extensive adoption of outsourced processing, distributed storage, etc.

To illustrate this complexity, consider the application scenario of social robot (e.g., Jibo, Kuri, Olly, BIG-i, Zenbo). A social robot is a Cyber Physical System (CPS) designed to reside in homes and act as personal assistant with a "personality" that makes it feel like a human companion [1]. In a typical case, the social robot is not meant to be used in isolation, but in close communication with robot manufacturer's cloud service, in order to separate expensive computations from the robot's hardware. This design pattern is common in current light weight devices to reduce the market price and support powerful functionality. In a smart home setting, a social robot is expected act as a central controller so that it will be communicating with Internet of Things (IoT) devices directly or indirectly. Several of the social robot companies plan to or have already released a software development kit (SDK) for interested parties to create add-on robot skills, which

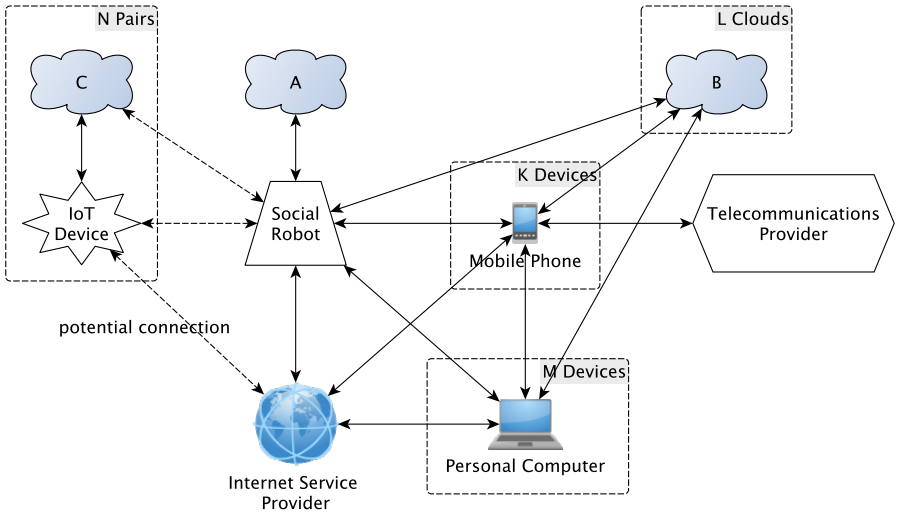

Figure 1.1: The CPS with the social robot in a home setting.

will lead the robot to communicate with additional could services other than that of the robot manufacturer.

In Fig.1.1, the robot manufacturer's cloud is depicted as cloud A. Apart from the robot's cloud that was just described the user could also have the robot exchange data with other clouds. Examples include Dropbox, Google Drive, and Facebook. The robot's company might even allow users to download add-on skills from platforms such as Google Play. These types of cloud services are depicted as a cluster of L clouds, in Fig.1.1.

The social robot will be just one of the devices sending or receiving data from the group of those L clouds. Smartphones, tablet and laptop devices found in a household today are often already connected to some of the L clouds. In addition, the user might desire these personal devices to exchange data with the social robot. For instance, the robot could send pictures taken with its camera through the Multimedia Messaging Service (MMS) or electronic mail. Some robots that are marketed not only as a friendly companion but also as a security patrol guard could also push alerts and videos of what they mark as unusual conditions. Similarly, the

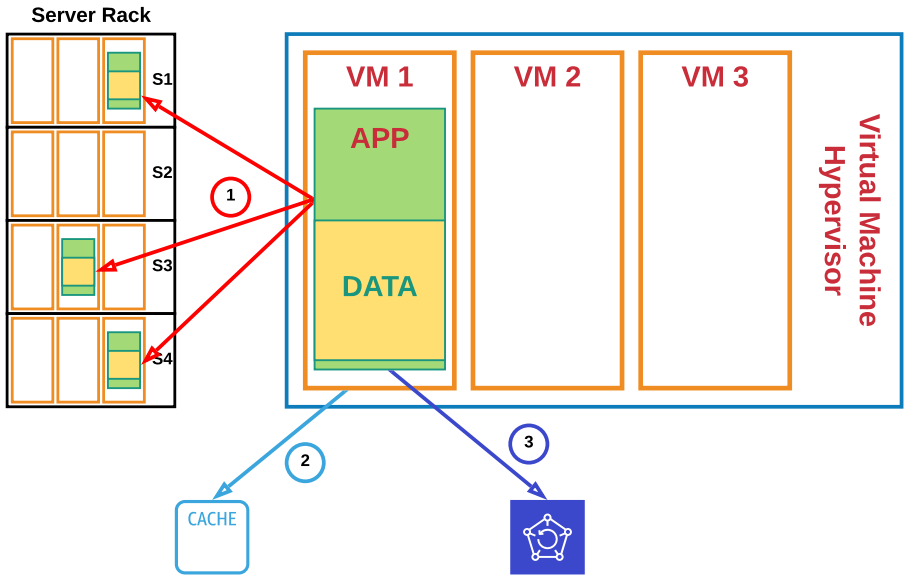

Figure 1.2: The risk of data management in IaaS cloud: 1) APP instance multi-server duplication for load balancing, 2) VM cached for fast recovery or migration, 3) APP and DATA backup on storage server, 4) Hypervisor could check the memory of VMs

user might want to share information like the contacts address book from the phone or personal computer to the social robot. The communication among the devices illustrated in Fig.1.1 could involve the cellular network, the Internet Service Provider's (ISP) network or a wireless local network (e.g. WiFi, Bluetooth Low Energy).

The introduction of a social robot in a home environment adds a significant degree of complication to the business relationships and network connections enabled in a smart home. Each of the mentioned clouds may be run by one entity as a Software-as-a-Service (SaaS), while a different administrative authority may be supplying the Platform or Infrastructure-as-a-Service (PaaS or IaaS). It could also be that the same IaaS (e.g., Amazon Web Services) is the layer below two different SaaS clouds (e.g., the social robot's cloud and an IoT device's cloud).

Given this complexity of connections and data movement, it is difficult to

monitor whether the data is properly protected on every connection and site because data will cross the boundary of domains, networks, platforms, and even boundary of nations with different regulations. Even if online service providers are generally trustworthy, their system could be misconfigured by the lack of experiences to dealing with the ever growing complexity of current computation environment. Fig.1.2 illustrates the execution model of an IaaS cloud and the difficulty in managing data in such environment. The system level intervene by the cloud infrastructure is out of the control of the service provider at the application level. A useful public ledger of policy compliance should contain the information of the entire view of data protection through the involved network.

In this dissertation, we propose a scheme that could provide a public ledger of policy compliance. To achieve this objective, we leverage public Blockchains as the public ledger. To address the scalability problem inherent in the Blockchain-based systems, we utilize state channels to implement the on-chain-hash-off-chain-data structure. We identify transaction verification as a critical problem when using the Blockchains as the public ledgers, since Blockchain is not as dependable as it is supposed to be in this scenario. We suggest a verification mechanism for the information that out of the view of the blockchain network, which is based on the incentive-based trust model implied by the state channel model. Then we propose the Verifiable Off-Chain Message Channel (VOCMC) as the integrated solution for leveraging blockchain technology as a general purpose recording mechanism. As mentioned previously, the verification of external information is critical for blockchain based public ledger, especially in the scenario considered in this work, where the information published on the ledger should be reliable. The VOCMC enable the verification of external information and the integration of blockchain with powerful off-chain computation to overcome the difficulty of scaling the blockchain. The VOCMC is derived from the state channel,

and its name emphasises the objective of the dissertation, which is to provide an approach to verify external information. On top of the VOCMC, we suggest a mechanism combined with sticky policy [2] to provide cross-boundary policy enforcement and monitoring, and thus a dependable and scalable public ledger for policy compliance. The result can be a public auditable record of the policy compliance, which can serve as the permanent immutable credit record of the service provider for the customer making informative decisions.

The rest of this dissertation is organized as follows.

- We first formalize the model of public ledger object and its verification mechanism. We observe that the verification of a blockchain transaction can be decoupled from the ledger object and in essence is part of the functionality of the transaction system. Therefore, we formalize this concept by *self-verfiable transaction system*, and thus prove its inability of verifying external information (Chapter 2).

- Secondly, we discuss the definition of VOCMC by starting with the concept of *incentive-based trust*. Briefly, if a piece of information is attached with an incentive that leads to interest conflicts among different parties, their agreement on what the information is could be a reliable description of the information. Based on this concept, the VOCMC actively adds incentives to the concerned external information and makes the participants of the channel to reach agreements on the information as a verification process. We examine the security properties under the universally composable security framework (Chapter 3).

- We thirdly use VOCMC as the building block to construct the prototype of public ledger of policy compliance, which combines off-chain database and on-chain hash. How to determine whether a policy complies with regulations

is out of the discussion scope of this dissertation, though it is a core mechanism. We assume that there exist a set of policies and a mechanism that could effectively determine whether a system has successfully complied with the polices in the set. We selected a representative policy set for concept demonstration. We leverage the *sticky policy* to track the user data and collect evidence required by the inference engine (Chapter 4).

- We evaluated the prototype to demonstrate its effectiveness on scaling blockchain applications. We performed prototype experiments to determine spacial and temporal overhead introduced by the implemented system as a whole, and the monetary and performance cost of on-chain execution. Based on these pilot tests, we examined how the off-chain component helps scale the system. We also tested the effectiveness of the sticky policy mechanism on a particular policy use case, the guaranteed deletion (Chapter 5).

The scalability of a blockchain system is the weakness for a lot of blockchain-based applications. This issue is amplified when dealing with micro-transactions in huge magnitude. Off-chain payment channels or state channels are designed to address the difficulty in scaling the blockchain system. Though the main objective of our proposed scheme is to provide verification of external information for the public ledger, it naturally reliefs the tension the system may suffer. At first glance, off-chain payment channels merge several micro-transactions into a single on-chain transaction. From another perspective, this process can be considered as reaching agreement on a sequence of computation results. In other words, any on-chain computation can be executed by an off-chain process as long as the result can be verified by the stakeholders, and reach an agreement on it. In consequence, we can limit the usage of on-chain computations, such as the smart contracts of Ethereum. The capacity of on-chain computation depends on the available balance for computation, which is not appropriate for persistent programs.

Another side effect of executing code on-chain is the halting problem that would significantly degrade the performance of the blockchain network.

Chapter 2

Blockchain-based Public Ledger

2.1 Introduction

Blockchain technology is attracting growing attention from both researchers and the general public after its success in cryptocurrency and the participation of governments and major financial industry players. Although blockchain and cryptocurrency technologies are often mentioned together, they are not tautologies; blockchain is only one of the cryptocurrency building blocks, albeit the most important.

In general, a blockchain serves as a public ledger for recording transactions. Transactions in Bitcoin are mainly exchanges of the cryptocurrency, from an account to another. From this perspective, a blockchain seems like just another kind of database. What makes blockchain interesting is its combination of technical facts and non-technical facts to achieve several critical properties for establishing a cryptocurrency.

In traditional banking, the transaction databases are maintained by organizations that obtained trust from the public, usually endorsed by the government or influential investors. Transaction databases are critical for banking and we need trusted organizations to operate them, because the transaction records are the only evidence of how the assets are located among millions of accounts. If the transaction database is secured, the assets owned by accounts are secured. The security of the transaction database depends on trusted organizations. However, Bitcoin adopted a security model without the trusted authority, which is referred to as a trusted third party in the majority of the security framework. The abandonment of the trusted third party is painful, but the solution provided by Nakamoto has not only solved the problem of establishing trust without trusted

parties but also given fundamental support to the value of Bitcoin.

The rest of this chapter illustrates the general framework of the Bitcoin-like cryptocurrencies, in order to explain what makes blockchain, a database technology, important and interesting to both industry and academia. As we will see, blockchain is a smart combination of existing technical and non-technical factors. This strong coexistence means that it is difficult to decouple blockchain from the cryptocurrency system for general purpose ledgers, and the performance challenges are hard to overcome. The definitions and terminology introduced in this chapter will support the discussion of later chapters.

2.2 The Bitcoin-like Blockchain

Blockchain is a distributed ledger technology (DLT). The Bitcoin blockchain in particular is based on peer-to-peer (P2P) networks in which every participating node, called mining node, has to keep the entire transaction database and update its local version as the data grows. As a result, multiple copies of the ledger exist in the network. If the copies disagree about some transactions, how do they resolve the issue and achieve agreement? This problem could refer to the consistency problem, the consensus problem, or, more generally, the Byzantine fault tolerance problem [3]. Inconsistent data is common in the Bitcoin-like blockchain. Suppose every mining node follows the protocol and never crushes. Despite this ideal execution environment, data inconsistency occurs due to the transfer latency across the network.

Fig. 2.1 illustrates this type of data inconsistency. Two transactions (i.e., transaction R and transaction B) enter the network from nodes with high transfer latency. Consider the two subnetworks. From the point of view of these subnetworks, the arriving order of these transactions is different. Another type of data inconsistency that may occur during valid execution of the protocol of

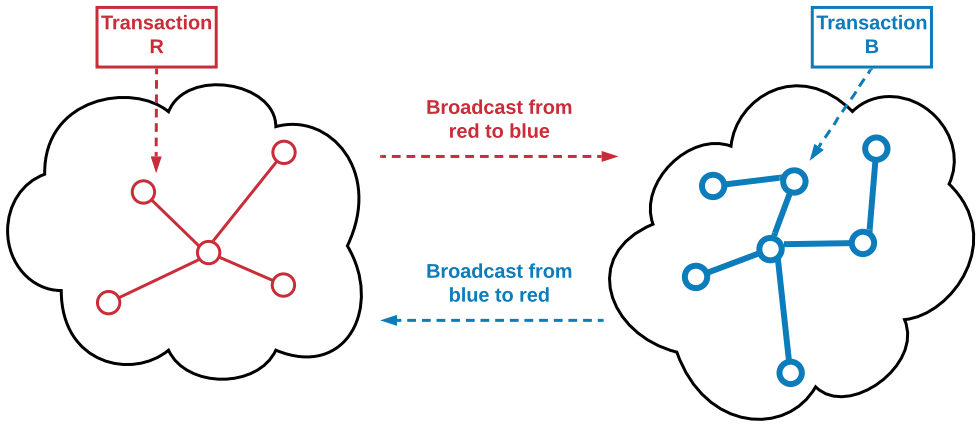

Figure 2.1: Transactions R and B enter the network from nodes with high transfer latency. From the point-of-view of the red nodes (the nodes on the left side), Transaction R arrived earlier than Transaction B, while blue nodes (the nodes on the right) consider Transaction B as earlier than Transaction R.

Bitcoin-like blockchain is the fact that mining nodes may get offline and back online, as every node could join and leave a P2P network anytime as they wish. As a result, there may always be some nodes with an outdated version of the database. The most important type of data inconsistency is caused by malicious nodes that may alter the database for their benefits. In a decentralized system without a trusted authority, it is difficult to identify if a copy of the database is without fraudulent transactions.

In order to achieve data consistency in such a complex environment, the Bitcoin-like blockchain relies on a protocol that tolerates Byzantine failure in that the mining nodes' behavior is, in essence, arbitrary and unpredictable, if the malicious behaviors are considered as a set of failure cases with arbitrary consequences. Nakamoto provided the original protocol in the Bitcoin white paper [4], which includes several smart and correlated design decisions that enable this scheme. Informally, the protocol has the following features:

- The transactions are recorded in a chained data structure, where the

transactions are arranged in blocks, and each block contains a field which is the hash of the previous block—then the transaction blocks are chained by this field.

- The value of a new coming transaction can be verified by the value of transactions resident in the existing blocks whose value could again be verified by earlier transactions.
- New blocks are generated through a competition process called mining. The winning node gets rewarded by obtaining the ownership of new Bitcoins accompanied by the new-born block.
- All the mining nodes must accept the longest chain as the valid version.

The ideas behind the blockchain are in fact older, and were formerly a method for time-stamping digital documents. They were introduced by Haber and Stonetta in a series of works beginning in the early 1990s [5, 6, 7]. By assembling data into blocks and chaining the blocks with hash pointers, this structure provides the feature of tamper-evident, as shown in Fig. 2.2. If an adversary had modified data in any block of the chain, it would invalidate the hash pointers of every following block. Technically, if we store the last valid hash pointer, the tampering could be detected even if an adversary modified the data and all the corresponding hash pointers in the chain. In centralized database architecture, the hash pointers could be stored by a trusted authority; however, it is not the case for decentralized systems like the blockchain.

The fundamental assumption behind Bitcoin-like blockchain is that there is no trusted authority or trusted third party in the system. An individual mining node has to keep its own copy of the blockchain. An observer cannot recognize which copy is the valid one, if multiple copies are displayed since, from the perspective of trust, all the copies are identical without a trusted authority.

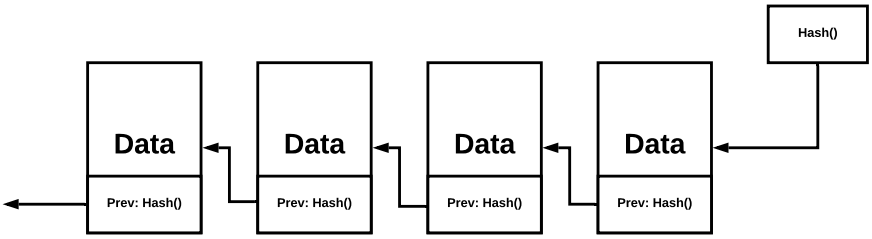

Figure 2.2: Every hash pointer depends on all the previous data and pointers. Once a piece of data is tampered, the following hash pointers will become incorrect. If an adversary attempts to modify the data, he has to re-compute every hash pointer influenced by the modification. If there is a trusted authority keeping the record of the correct value of the last hash, data tampering is always detectable, even if the adversary modifies the data and hash pointers appropriately.

Imagine, blockchain A and B store the same data except for a single disagreed data entry. Correspondingly, the hash pointers are differentiated between blockchain A and B after that data entry. It is possible that B modified the data, or A did the modification. However, there is no authentic version kept in a secure place; thus, one could not tell which is the valid chain without modification.

Nakamoto innovatively solved this problem. Firstly, the generation of hash pointers becomes a computationally difficult puzzle. As illustrated in Fig. 2.3, to generate the hash pointer for a data block, the mining node has to find a string of bit called nonce with which the generated hash pointer displays some required patterns. In the Bitcoin system, the required pattern for a hash pointer is some leading zeros in the generated hash. In order to find an answer to a puzzle for a specific block, the mining nodes have to randomly select a nonce, then calculate the resulted hash to verify if the hash meets the requirement. Repeat this cycle until a valid nonce is found and thus a valid hash. This technical decision makes the hash generation inefficient, and thus inefficient to regenerate the entire blockchain if a piece of data is modified. An adversary has to successfully solve as many puzzles as the number of blocks following the data he intends to change.

It is not enough to secure the valid blockchain by just raising the difficulty of

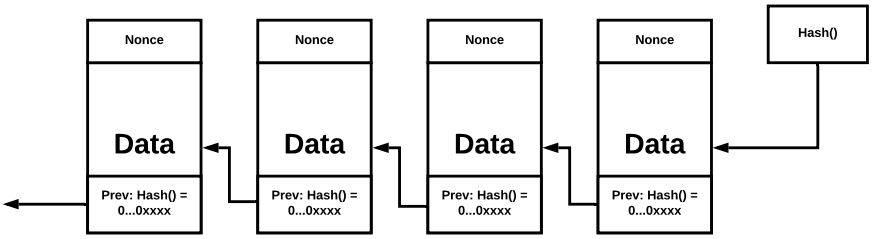

Figure 2.3: In Bitcoin like blockchain, there is a field called nonce with which the resulted hash of a block will demonstrate some required patterns. Currently, the required pattern is the number of leading zeros in the resulted hash pointer. The process to find a valid nonce is called mining.

its hash generation. Suppose an adversary with a supercomputer, then it is possible to recreate a blockchain in an acceptable time span. The second requirement to secure the valid blockchain is that the honest mining nodes control the majority of computation power in the network. Since new blocks are rapidly added to the blockchain, an adversary could not just recreate the historical blocks to tamper the data, but also generate the hash for new blocks correspondingly. To be successful, the adversary has to maintain computation resources stronger than the honest nodes; otherwise, the tampered blockchain could be recognized by the miss of the most recent blocks. Based on the assumption that the majority of computation resource is controlled by the honest nodes, the Bitcoin-like blockchain further requires the mining nodes to accept the longest chain as the currently valid version, since the probability is high that an adversary could not generate blocks faster than the honest nodes as the computation power owned by the adversary is weaker.

Interestingly, Nakamoto did not specify a strong technical method to protect the valid hash pointers. For instance, other distributed systems split a secure key among nodes, and no single node could access and modify critical data [8, 9, 10]. In Bitcoin, multiple different versions are allowed, and the authentication condition is simple—by the length of the chain. The critical factor enabling the entire approach is the assumption of honest majority computation power. In principle, if an

adversary has the computation resource to recreate a blockchain that is longer than generated by the honest nodes, there is no doubt that the tampered data will become the valid version.

Assumptions are often not reliable. The assumption under the centralized system is that the authorized third party is trustworthy, which is not always true, and that is why systems like blockchain will appear. Though the assumption under blockchain can become unreliable as well, there is a fair way to make the reliability probability high. The answer is to reward honest nodes by incentive. The honest miners are encouraged by rewarding them with financial benefits—the coins. In the Bitcoin protocol or other similar cryptocurrency scheme [4, 11], incentive comes in two ways:

- the nodes who find a new block will be automatically rewarded with Bitcoins;
- the owner of the transactions will pay the mining nodes transaction fee.

Technically, these two types of incentive could not prevent misbehavior from happening, since a successful adversary could get all the rewards. By misbehaviors such as forking the blockchain or selfish mining, dishonest miners could rubber the coins of honest miners by simply invalidating blocks generated by them. If the assumption had held, such an attack would fail in that the adversary is unlikely to generate more blocks than the honest majority. Forking the blockchain will increase the risk of mined blocks being discarded, which means the legal revenue is decreased.

There is another layer of incentive—the reputation of the entire system and, thus, the value of the rewards. If the adversary controls a dominant computation power and misbehaviors happen consistently, the market confidence will be impacted, the honest players and the public will quit, and the currency will become worthless [12]. An essential fact to the success of Bitcoin is the interest binding between the players and the system. The rewards are in the form of Bitcoin. In

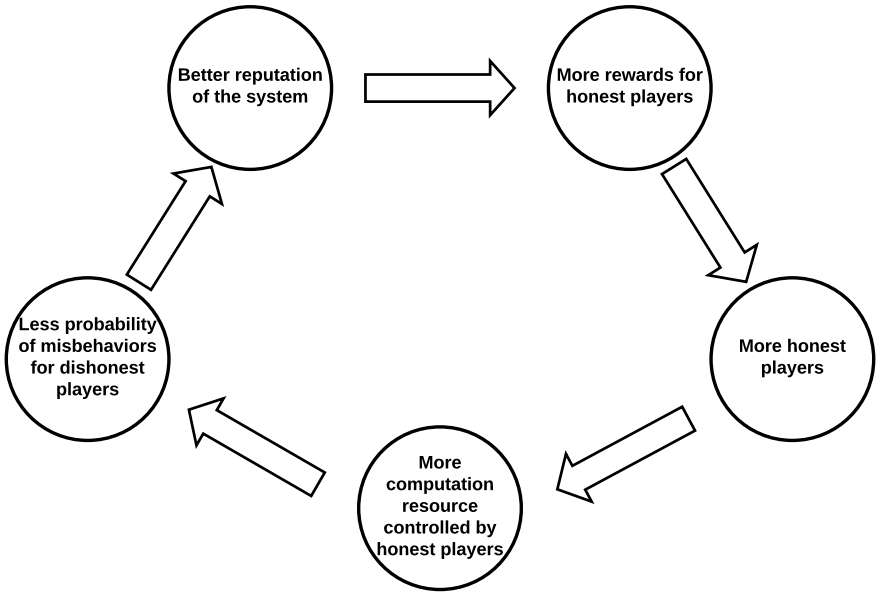

Figure 2.4: Reputation of the system guarantees the rewards to the participants, and thus the participants will voluntarily preserve the reputation of the system. The increase of honest players helps maintain the portion of the computation resource controlled by the honest players. Therefore, the probability of a successful attack is kept low, which in turn enhances the system reputation.

order to make the rewards they get worth more, the mining nodes have to preserve the reputation of the system in public, which forms a benign cycle, as demonstrated by Fig. 2.4:

- the more significant reputation of the system leads to the higher value of the coins and thus the value of the rewards;
- more rewards lead to more honest players;
- the presence of more honest players increases the computation resource controlled by the honest parties;
- it is difficult for the dishonest player to increase their portion of computation resource controlled
- the possibility of successful misbehavior remains low;

- the reputation of the entire system gets enhanced.

The success of Bitcoin, and any other cryptocurrency, depends also on less controllable factors. The system needs a bootstrap at the early stage. Early acceptance by the public is vital for the thriving of the entire ecosystem.

It is clear that the components of the Bitcoin system are integrated tightly. Without the incentive scheme, the blockchain is vulnerable to adversaries with strong computation power. The security scheme of blockchain is weak from the aspect of technology. The technical improvement from its original version is the hash puzzle. Nevertheless, the puzzle could not make the scheme completely secure, because it is a randomized computation. The adversary wins or loses by chance.

By this point, we have discussed the success of Bitcoin, not the success of blockchain. We observed that it is challenging to discuss blockchain without the background of cryptocurrencies because the attempt to decouple the blockchain from a binding cryptocurrency usually results in fundamental modifications to the underlying trust assumptions. The success of Bitcoin also depends on human factors. It is true that the blockchains of successful cryptocurrencies preserve important properties, especially the Byzantine fault tolerance in a decentralized anonymous network without a trusted authority. However, this property is technically quite expensive. For instance, more mining nodes generally lead to a greater extent of decentralization of computation resources, which is good for the security of the entire system, but also leads to more fierce competition into owning a new block. The result is that the performance of the entire system would not scale with the increase of computation resources. In fact, the performance is degraded in terms of energy consumption, since the rate of block generation is somehow locked by the protocol. More mining nodes simply increase energy consumption per block.

Hence, if we discuss blockchain without human factors, constrains become obvious. For example, to scale the blockchain, researchers introduced permissioned

blockchain, where the mining nodes could only join the network with permission. In such a system, miners are more easily identified and traceable, in order to regulate the behavior of miners, while miners could participate or quit as they wish and anonymously. The difference is subtle comparing to a system enabled by a trusted authority.

This observation inspires our introduction of the model of incentive-based trust. It is important to understand the degree of trust provided by the incentive-based approach and the scenarios in which such an approach is applicable. In the following section, the pros and cons of blockchain as public ledger will be discussed in detail.

2.3 Blockchain as Public Ledger

The blockchain serves as the ledger of transactions in Bitcoin. Comparing to its predecessors [5, 6, 7], Nakamoto's blockchain is characterized by the following remarkable features:

- **Distributed ledger**—each mining node maintains its local copy of the blockchain with the requirement that mining nodes have to update their local copy once a version with longer chain length is found.
- **Public accessibility**—any user can view the transactions published on the blockchain network.
- **Immutability**—transactions are confirmed and accepted by all nodes in the network through the mining process in which the transactions are added into blocks that are finalized by proof of work (by solving the hash puzzle); once a transaction is confirmed in this manner, the probability of malicious modification to the transaction is negligible, because the adversary has to resolve the hash puzzles for the following blocks influenced by the modification.

- **Public verifiability**—transactions could be verified by preceded transactions in the blockchain.

Due to these desirable properties, blockchain is a promising technique for public ledger providing auditability and transparency, such as decentralized information management, immutable record-keeping for possible audit trail, and robust and available system. Therefore, an increasing number of works aim to decouple the blockchain from cryptocurrency as a pure database technology—the distributed ledger technology [13]. As discussed in section 2.2, the immutability is essentially based on the assumption that the majority of computation resource is controlled by honest players. More precisely, a successful attacker has to control over 50% of the mining power in order to succeed in forking the blockchain with non-negligible probability [12]. However, this assumption also makes it difficult to realize these desired properties without the appearance of a build-in incentive strategy. In cryptocurrency, the primary source of reward is the coins mined during the process of block generation. Thus despite the cheap transaction fee, miners likely join and stay in the network, which provides a stable population base of active miners. If the rewards purely come from transaction fees, the cost for the ledger user would be unsustainable, considering the energy consumption of a blockchain network with the scale comparable to that of Bitcoin. The estimated electricity used per transaction of Bitcoin is reaching 200 kWh. Ethereum transactions seem much cheaper than that of Bitcoin, scoring an electricity consumption of 37 kWh. However, compared to the 0.01 kWh of Visa transaction, we can imagine 3700 times of processing fee, if we use a blockchain-based credit card [14].

For reducing the cost of blockchain application and scalability, researchers have introduced permissioned or private blockchain, contradictory to permissionless or public blockchain, i.e., the Bitcoin-like blockchain. A private blockchain is able to adopt different algorithms to scale the performance. Nodes cannot join a private

blockchain network by simply downloading the current version of the blockchain and broadcasting a joining announcement. Intentional nodes have to apply for permission from the blockchain owner. Thus the permission itself serves as a kind of a behavior limitation to the mining nodes, because to get permission the identity of joining nodes shall be verified, which implies a non-anonymous network. Adversaries to such a system do not enjoy a leisure equivalent to that of public blockchain—to join freely, try attacks, and then quit freely. If attackers get caught in a public blockchain, they can simply start using another identity.

Mining in private blockchain is organized. For example, in Hyperledger fabric [15], transactions would be firstly ordered by the *Ordering Service Nodes*, and then send to some peers designated by an endorsement policy for execution (instead of mining nodes, nodes in Hyperledger are called peers to indicate that their function is not mining blocks, but executing code in transactions). The result would be endorsed by the designated peers, and other peers need only validate the endorsement instead of re-executing the transaction. This treatment eliminates mining competition and avoids blockchain forking. It further reduces the computation workload by the endorsement scheme.

Despite the performance gain of a private blockchain, it indeed changes the essential trust model of the original blockchain, which, for some advocates of blockchain technology, is the spirit of this technology. The decentralized trust model is suspectable in permissioned blockchain in that the owner of the blockchain is the only entity to release the permission. The behaviors of participants are not independent of the influence of the owner; therefore, the immutability and integrity of data are under potential threat. The idea to eliminate the trusted authority comes from a fundamental question: is the trusted authority actually trustworthy? It is similar to the political problem of how to prevent the government from abusing its unlimited power.

This dissertation inclines to the public or permissionless blockchain mode for the following reasons:

- Technically, the difference between private and public blockchain mainly reflects the tradeoff between performance and security features. Suitable decisions depend on application scenarios. There is no "one-size-fits-all" consensus protocol (Byzantine fault tolerance), which is a well-established understanding [16];
- Existing public blockchain still has potential with regard to scalability, when combined with other technologies, such as off-chain channels;
- When considering off-chain extensions, public and private blockchain are logically equivalent because they provide the same interfaces for off-chain applications.

The choice between private and public blockchain is mainly about the conflict between performance and immutability of data. We can choose private blockchain for better performance, vice versa, depending on the nature of application scenarios. Now we consider the feature of public verifiability, which is another primary reason for the adoption of blockchain as a public ledger.

2.3.1 Questionable Public Verifiability of Blockchain

Public verifiability and immutability of data of blockchain attract researchers looking for transparent and immutable record keeping for possible audit trail [17]. However, approaches often leverage the public accessibility instead of the verifiability feature. For instance, consider a blockchain-based financial statement publishment system, where companies anonymously publish their statements on a blockchain for audit. A questionable logic in such a system is that if a company

Algorithm 1: Ledger Object $\mathcal{L}$

```
1 Init:S ← ∅
2 function L.get()
3     return S
4 function L.append(r)
5     S ← S||r /* || is the concatenation operator */
6     return
```

tries to cheat, such misbehavior is detectable because of the public verifiability of the blockchain, and therefore the auditing result is provably correct. If there is no anonymous requirement, public accessibility may provide some degree of verifiability since it is relatively easy to build a connection between the financial statement and the company and its operational activities. Nevertheless, with anonymity, as long as the number of income and outcome matches, there is no way to identify a cheater by the use of blockchain. To understand this problem, we should examine the public verifiability of blockchain in detail.

Let us start with a formal definition of a ledger object for the convenience of further discussion. We borrow the form from Antonio F. Anta and Maurice Herlihy [18, 19].

Definition 1. *A ledger $\mathcal{L}$ is a sequence of records over a sequential history H defined as follows. The initial value of the sequence $\mathcal{L}.S$ is the empty sequence $\emptyset$. Suppose the value of the sequence in ledger $\mathcal{L}$ is $\mathcal{L}.S = V$ at the invocation of an operation π, then:*

1. *if π is an $\mathcal{L}.\texttt{get}_p()$ operation, then the response of π returns V, and*
2. *if π is an $\mathcal{L}.\texttt{append}_p(r)$ operation, then set the value of sequence in ledger $\mathcal{L}$ to $\mathcal{L}.S = V||r$.*

Operations `get()` and `append()` are defined in Algorithm 1.

This definition describes a ledger as an append-only list of records and does not capture the distributed nature of blockchain, but it is enough for the discussion

of the public verifiability of blockchain in this section. We extend it to a distributed version at the end of this chapter. Note, though the operation `get`() only returns the entire list S, other types of reading operation could easily be implemented on top of `get`(). This treatment is for the simplicity of the definition and reflects the essence.

In the work of Maurice Herlihy [19], a ledger with public verifiability is realized by adding a Boolean function `Valid`(). Precisely, `Valid`() is invoked by the operation `append`() with an input of a sequence of records S and returns *true* if and only if some semantics are preserved. Upon the return value of `Valid`(), `append`() fails or accepts the operation request. The formal definition of the validated ledger is extended from Definition 1, taking into account the influence of `Valid`() to the execution route:

Definition 2. *A **validated** ledger $\mathcal{VL}$ is a sequence of records over a sequential history H defined as follows. The initial value of the sequence $\mathcal{VL}.S$ is the empty sequence $\emptyset$. Suppose the value of the sequence in ledger $\mathcal{VL}$ is $\mathcal{VL}.S = V$ at the invocation of an operation π, then:*

1. *if π is a $\mathcal{VL}.\texttt{get}_p()$ operation, then the response of π returns V,*
2. *if π is a $\mathcal{VL}.\texttt{append}_p(r)$ operation and $\texttt{Valid}(V\|r) = true$, then set the value of sequence in ledger $\mathcal{L}$ to $\mathcal{L}.S = V\|r$ and respond with ACK, and*
3. *if π is a $\mathcal{VL}.\texttt{append}_p(r)$ operation and $\texttt{Valid}(V\|r) = false$, then keep the value of sequence in ledger $\mathcal{L}$ to $\mathcal{L}.S = V$ and respond with NACK.*

Operations `get`(), `append`(), and `Valid`() are defined as in Algorithm 2.

The semantics need to be preserved for Bitcoin-like blockchain by `Valid`() in Maurice Herlihy's work [19] and described as preventing double-spending. However, this is not a precise abstraction of the verification mechanism of Bitcoin-like blockchain. To the best of our knowledge, few works are aiming to understand and

Algorithm 2: Validated Ledger Object $\mathcal{VL}$

```
1 Init: S ← ∅
2 function VL.get()
3     return S
4 functionVL.append(r)
5     if Valid(S‖r) then
6         S ← S‖r
  /* ‖ is the concatenation operator                    */
7         return ACK
8     else return NACK
```

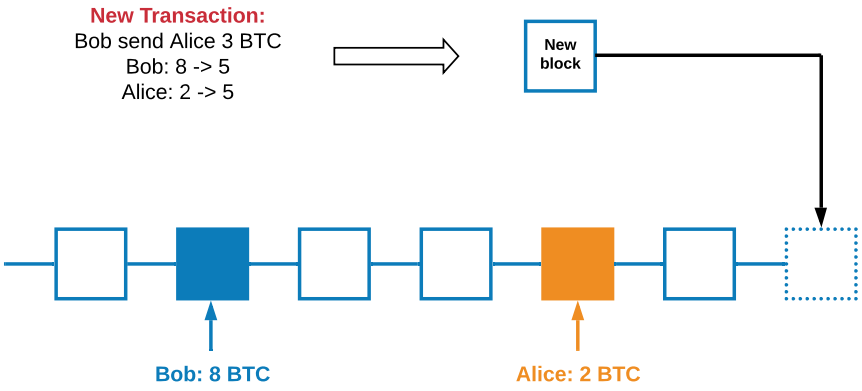

Figure 2.5: The transaction verification of Bitcoin-like blockchain relies on previous transactions. In this case, Bob intends to transfer 3 BTC to Alice. The mining nodes check if the account of Bob has enough balance to complete this trade by tracing back along the blockchain to find transactions that transfer more than 3 BTC to Bob and those Bitcoins have not yet been spent.

model the verification mechanism of blockchain, and thus lead to drawbacks of applications relying on the verification feature.

Fig. 2.5 displays how the verification mechanism works. Suppose Bob transfers Alice 3 units of Bitcoin,

1. a new transaction is broadcasted to the network announcing that Bob transfers Alice 3 units of Bitcoin;

2. the mining nodes check the blockchain to find if any transactions are indicating Bob has enough units to complete this transfer;

3. if Bob has enough Bitcoin in his account, the mining nodes include this new transaction in a block; and

4. when the block containing the new transaction is successfully mined, the transfer is completed.

What makes the verification process subtle is the structure of the account information. Bitcoin accounts do not have integrated data in the blockchain to describe their status, though there are Bitcoin wallet applications that collect and summarise necessary data to represent an account. Instead, only the transactions are listed in the blockchain. In order to check if an account has enough balance to finish a transaction, mining nodes traverse the blockchain to find one or more transactions whose combined balance is greater than the required value, and that have not been spent yet. A new transaction must refer to the transactions whose balance is spent in this new transaction, which means **a transaction with a balance is an incoming transaction and has no further transactions refer to it**.

Fig. 2.6 illustrates a case possible for the example in Fig. 2.5. The first incoming transaction is of balance 2 BTC and referred by two spending transactions that spend 1 BTC each. Thus when the Bob-to-Alice transaction is announced, the mining nodes would not include it into a candidate transaction for this transfer. When miners find the transaction with a balance of 7 BTC, they stop traversing the blockchain and pick this transaction as the one spent by the new transaction. In the new transaction, 3 BTC are transferred to Alice, and the rest 4 BTC are included in a self-transfer to Bob, and the transaction with a balance of 7 BTC is referred.

Clearly, there is a linked list of transactions related to every unit of Bitcoin, from the latest incoming transaction back to its origin. An important feature of Bitcoin is that the only source of Bitcoin is the mining activity. Whenever a block is mined, the miner who solves the puzzle could include a transaction that indicates a

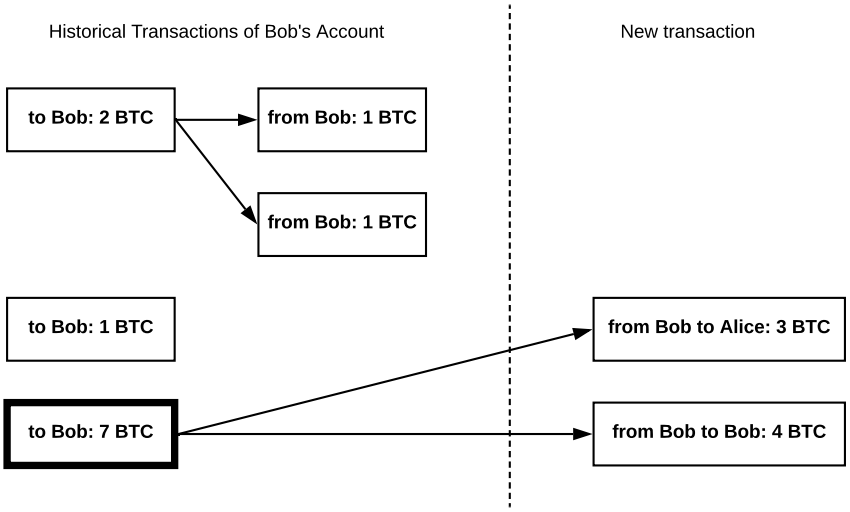

Figure 2.6: A possible situation of transactions corresponding to the example in Fig. 2.5. The first to-Bob transaction is a spent transaction which could be used for completing this trade. The second to-Bob transaction is a not-spent transaction, but indicates a balance not enough to complete the transfer. The third is a not-spent transaction and has enough balance, thus the new transactions refer to it transferring 3 BTC to Alice and the rest back to Bob.

certain amount of Bitcoin is transferred to the miner's account. Unlike the traditional banking system, there is no deposit transaction that injects balance from another source. This feature is important because, in essence, the "space" of Bitcoin is closed. Formally, we model the Bitcoin transaction system by the *Self-Verifiable transaction system*:

Definition 3. *A* ***self-verifiable transaction system*** $\mathcal{VT}$ *consists of three components:*

1. *an element base* $\mathcal{EB}$ *which is initialized as an empty set* $\emptyset$*—an element is noted as a tuple* $\{e_i, value\}$
2. *a label set* $\mathcal{ID}$ *which is defined over the space of strings with a fixed length,*
3. *a transaction forest* $\mathcal{TF}$ *whose trees are rooted in the element base* $\mathcal{EB}$*—the nodes in layers below the roots satisfy the requirement as follows,*

- *every node t contains a tuple $\{id, value, parent\}$, where $id \in \mathcal{ID}$,*
- *if a node t has child nodes, its value $t.value$ equals the sum of the values of its child nodes, $t.value = \sum_{i \in child} t_i.value$.*

Allowed operations in the system are defined as follows:

- ***Element generation**, `gen`(id)—add a new element e into the element base $\mathcal{EB}$, and create a new tree with one child node rooted with the newly generated element in the forest $\mathcal{TF}$,*
- ***Validation**, `valid`$(\mathcal{TF})$—take the transaction forest and the element base as the input, and return true if and only if the sum of the values at the root layer equals the sum of the values of all the leaves,*
- ***Transfer**, `transfer`$(id_{from}, id_{to}, value)$—take as input the ID of the sender id_{from}, the ID of the receiver id_{to}, and the value, find a set of leaves with $t.id = id_{from}$ satisfying $\sum_{i \in found} t_i.value \geq value$, add child nodes with $t.id = id_{to}$ satisfying $\sum_{i \in added} t_i.value = value$, and add a child node with $t.id = id_{from}$ satisfying $t.value = \sum_{i \in found} t_i.value - value$; invoke `valid`() with the transaction forest $\mathcal{TF}_{updated}$ with newly added nodes; if `valid`$(\mathcal{TF}_{updated}) = true$, accept the transfer. Otherwise, rollback the transaction forest.*

Operation `gen`(), `transfer`(), and `valid`() are defined as in Algorithm 3.

Per the model definition of the self-verifiable transaction system, we can provide a proof of the verifiability feature of Bitcoin-like blockchain to prevent double-spending attacks.

Theorem 1. Let $\mathcal{VT}$ be a self-verifiable transaction system, as defined in Definition 3. Then there is no double-spending transactions iff $\mathcal{VT}$.`valid`() returns true.

Algorithm 3: Self-verifiable transaction system $\mathcal{VT}$

Init: $\mathcal{EB} \leftarrow \emptyset$, $\mathcal{ID} \leftarrow \{0,1\}^n$, $\mathcal{TF} \leftarrow \emptyset$
function $\mathcal{VT}$.`gen`(id)
    $e =$ `new element`
    $\mathcal{EB} \leftarrow \mathcal{EB} \cup e$
    `new node` $t \leftarrow \{id, e.value, e\}$
    $\mathcal{TF} \leftarrow \mathcal{TF} \cup t$
function $\mathcal{VT}$.`valid`($\mathcal{TF}$)
    if `sum`$(t_{i \in roots}.value) =$ `sum`$(t_{j \in leaves}.value)$ **then**
        return *true*
    else return *false*
function $\mathcal{VT}$.`transfer`($id_{from}, id_{to}, value$)
    find a set of leaves C:
        `sum`$(t_{i \in C}.value) \geq value$
        $t_{i \in C}.id == id_{from}$
    create new set of nodes NC:
        `sum`$(t_{i \in NC}.value) = value$
        $t_{i \in NC}.id = id_{to}$
        $t_{i \in NC}.parent = id_{from}$
    create new node t:
        $t.value =$ `sum`$(t_{i \in NC}.value) - value$
        $t.id = id_{from}$
        $t.parent = id_{from}$
    $\mathcal{TF} \leftarrow \mathcal{TF} \cup NC \cup t$
    if $\mathcal{VT}$.`valid`($\mathcal{TF}$) **then**
        return *ACCEPT*
    else
        `rollback`
        return *NACCEPT*

Proof. By Definition 3, the fact that $\mathcal{VT}$.`valid`() returns true indicates that the sum of the values at the root layer equals the sum of the values of all the leaves in the transaction forest $\mathcal{TF}$. Another fact is that if a node t has child nodes, its value $t.value$ equals the sum of the values of its child nodes, $t.value = \sum_{i \in child} t_i.value$. Thus, we can divide each tree by the method below,

- for a node with only one child node, keep it the same,
- for a node t with multiple child nodes, create a dummy set of nodes in which

each dummy node corresponds to a child node and satisfies

$$t_{d_i}.value = t_{c_i}.value$$

$$\sum_{d_i} t_{d_i}.value = t.value$$

$$t_{d_i}.id = t.id$$

$$t_{d_i}.parent = t.parent$$

- abort if any of the conditions could not be satisfied.

Repeat this treatment from the leaves up to the roots, and then we get a dummy forest in which each tree is a linked list with every node that has the same value. In essence, we have established a 1-on-1 map between every not-spent transaction to its element base. Obviously, if this 1-on-1 map is established successfully, it is equivalent to the fact that the sum of the values at the root layer equals the sum of the values of all the leaves in the transaction forest $\mathcal{TF}$. This 1-on-1 map indicates no element is spent twice as well. Therefore, the problem is reduced to there is no double-spending transaction iff the 1-on-1 map could be created. We have shown $\Rightarrow$, we now prove $\Leftarrow$. Suppose there is a pair of double-spending transactions. Then there must be at least one node t_{ds} whose value is less than the sum of the values of its child nodes, $t_{ds}.value < \sum_{i \in child} t_i.value$. Thus, the dividing operation is aborted, and the map could not be created. □

We defined the operation `valid`() as comparing the sum of leaves and the sum of roots for the conciseness of discussion. From the basic definition, a formulation for quicker verification could be induced.

Theorem 2. For a new transaction t of `transfer`$(id_{from}, id_{to}, value)$ in a self-verifiable transaction system $\mathcal{VT}$, the operation `valid`$(t, t_{parent}, t_{sibling})$ is equivalent to the operation `valid`$(\mathcal{TF})$ in Definition 3, which is defined as below,

- return $true$, if the sum of the values of t and $t_{sibling}$ equals the value of t_{parent},
- return $false$ otherwise.

Proof. For simplicity, we only discuss the case with only one node involved. Proving the case with multiple nodes follows by separating them into several single node cases. From the structure of the forest, a transfer transaction might add at most two children to a node, with the constrain that the sum of the new child nodes equals their parent's value. Therefore, this operation would not change the sum of the leaves layer, and the equivalence holds. □

This model of self-verifiable transaction system helps understand the verifiability of Bitcoin-like blockchain. The insights we achieved from the study is that the verifiability of Bitcoin-like blockchain is actually provided by the self-verifiable transaction system. This transaction system is established on top of a distributed ledger object, and the feature provided by the distributed ledger object is the public accessibility.

It seems that this combination could provide public verifiability; however, we show the limitation of the self-verifiability transaction system, and when combining with Bitcoin-like blockchain, its verifiability is undermined.

From the proof of Theorem 1, the verification of a transaction is, in essence, to find the origin of the information contained in the transaction from the element base $\mathcal{EB}$. In other words, if the information does not have its origin in the element base $\mathcal{EB}$, then it could not be verified by the operation `valid`(). A double-spending transaction could not be validated because it indeed introduces information whose origin is out of the element base $\mathcal{EB}$. Apparently, the reliability of the self-verifiability transaction system relies on the generation of elements because **the verifiability of the system could not guarantee the operation `gen`() is valid**. An implicit feature in the self-verifiability transaction system is that the

generation of elements is always valid by default. In Bitcoin-like blockchain, the validity of generation is justified by proof of work, which is out of the transaction system itself.

Consider leveraging the Bitcoin-like blockchain (which is a combination of a ledger object and a self-verifiable transaction system) for general-purpose record keeping. We must firstly extend Definition 3 to support data that is different from cryptocurrency. We provide an extended definition after the necessary discussion. As mentioned previously, Bitcoin-like blockchain justifies the generation of its cryptocurrency by proof of work during the mining process. This method is a natural choice because the transaction system is exclusively designed for cryptocurrency, and the system pays the miners for their work with newly generated currency. However, proof of work could not justify arbitrary data. For instance, a service provider violates a privacy policy, but announces to the blockchain that it complies with the policy. This announcement, in this case, is incorrect information, probably misleading, even malicious. A blockchain with self-verifiability transaction system has two possible treatments when dealing with this information:

- treat the information as a transaction, then `valid`() returns false and reject the information, or
- treat the information as operation `gen`(), then add it into the element base without verification.

Assume the system supports such information by assigning a recognizable value. Then, the first treatment always rejects such information regardless its correctness since there is no element in the element base $\mathcal{EB}$ corresponding to the information. By definition, a transaction could be accepted only if a related element has been generated prior to the transaction; otherwise, `valid()` always returns false. On the other hand, the second treatment accepts the information without

justification because proof-of-work is unable to make a judgment on the correctness of specific information. Obviously, the first treatment is not applicable for general purpose data storage, because it prevents any data from being kept into a blockchain system. The second treatment is the only way to include arbitrary data into Bitcoin-like blockchain, but the reliability of data could not be ensured. Back to the example, the ledger includes the incorrect report regarding the policy enforcement performance of the service provider. The public might make misled decisions, if people trust this ledger as a reliable data source.

We extend the self-verifiable transaction system by a hybrid-verifiability transaction system, as defined below.

Definition 4. *A **hybrid-verifiability transaction system** $\mathcal{HV}$ consists of four components:*

1. *an internal element base $\mathcal{IE}$ which is initialized as an empty set $\emptyset$—an element is noted as a tuple $\{e_i, value\}$,*
2. *a label set $\mathcal{ID}$ which is defined over the space of strings with arbitrary length $\{0,1\}^*$,*
3. *an external element base $\mathcal{XE}$ which is initialized as an empty set $\emptyset$—an element is noted as a tuple $\{id, string | id \in \mathcal{ID}\}$,*
4. *a transaction forest $\mathcal{TF}$ whose trees are rooted in the internal element base $\mathcal{IE}$—the nodes in layers below the roots satisfy the requirements that follow,*
   - *every node t contains a tuple $\{id, value, string, parent\}$, where $id \in \mathcal{ID}$,*
   - *if a node t has child nodes, its value $t.value$ equals the sum of the values of its child nodes, $t.value = \sum_{i \in child} t_i.value$.*

Allowed operations in the system are defined as follows,

- ***internal element generation***, *`INgen`(value)—add a new element ie into the internal element base $\mathcal{IE}$, and create a new tree with one child node rooted with the newly generated element in the forest $\mathcal{TF}$,*

- ***external element generation***, *`EXgen`$(id, string)$—add a new element xe into the external element base $\mathcal{XE}$,*

- ***validation***, *`valid`$(\mathcal{TF})$—take the transaction forest and the element base as the input, and return true if and only if the sum of the value field at the root layer equals the sum of the value field all the leaves (the string field is not taken into account),*

- ***transfer***, *`transfer`$(id_{from}, id_{to}, value, string)$—take as input the ID of the sender id_{from}, the ID of the receiver id_{to}, and the value, find a set of leaves with $t.id = id_{from}$ satisfying $\sum_{i \in found} t_i.value \geq value$, add child nodes with $t.id = id_{to}$ satisfying $\sum_{i \in added} t_i.value = value$, add a child node with $t.id = id_{from}$ satisfying $t.value = \sum_{i \in found} t_i.value - value$, and invoke `EXgen`($id_{to}$,string); invoke `valid`() with the transaction forest $\mathcal{TF}_{updated}$ with new added nodes; if `valid`$(\mathcal{TF}_{updated}) = true$, accept the transfer, otherwise rollback the transaction forest.*

Operation `INgen`(), `EXgen`(), `transfer`(), and `valid`() are defined as in Algorithm 4, 5.

This definition provides the functionality for general-purpose data storage. It captures the fact that the blockchain system is not reliable when serving as a general-purpose ledger, because it lacks the capability of verifying external information. The essential difference between the operations `INgen`() and `EXgen`() is that the `INgen`() is a trusted method, i.e., the information generated by `INgen`() is believed to be the truth, while `EXgen`() only generates an element for injected

Algorithm 4: Hybrid-verifiability transaction system $\mathcal{HV}$

Init: $\mathcal{IE} \leftarrow \emptyset$, $\mathcal{XE} \leftarrow \emptyset$, $\mathcal{ID} \leftarrow \{0,1\}^n$, $\mathcal{TF} \leftarrow \emptyset$
function $\mathcal{HV}$.`INgen`($value$)
    $e =$ `new element`
    $e.value = value$
    $\mathcal{IE} \leftarrow \mathcal{IE} \cup e$
    `new node` $t \leftarrow \{id, e.value, \varnothing, e\}$
    $\mathcal{TF} \leftarrow \mathcal{TF} \cup t$
function $\mathcal{HV}$.`EXgen`($id, string$)
    $e =$ `new element`
    $e.value = value$
    $e.id = id$
    $\mathcal{XE} \leftarrow \mathcal{XE} \cup e$
function $\mathcal{HV}$.`valid`($\mathcal{TF}$)
    if `sum`($t_{i \in roots}.value$) = `sum`($t_{j \in leaves}.value$) **then**
        return *true*
    else return *false*

external information, but does not guarantee its authenticity. This model is a reasonable abstraction of the Bitcoin-like blockchain. The block mining process functions as the `INgen`() in which the coins are generated as elements, and the proof-of-work justifies the generation of coins as truth. The Bitcoin transactions are monetary only, and the verification process only determines whether an address owns enough coins to complete a transfer. There are no transactions that transfer coins which are not derived from mining. The "coin", or its equivalent, is the only verifiable object in the ledger.

Though the Bitcoin-like blockchain allows users to upload some data by attaching a string of bit in transactions, it accepts this data field without any verification. In our definition, the operation `EXgen`() corresponds to this feature. Defining the inclusion of external data as a type of generation of elements is a natural choice, as mentioned previously. There is an additional benefit of this decision: we could retain the definition of allowed operations as an option for users, and a self-verifiable transaction system could be defined on top of $\mathcal{XE}$ without too

Algorithm 5: Hybrid-verifiability transaction system $\mathcal{HV}$

```
1  function VT.transfer(id_from, id_to, value)
2      find a set of leaves C:
3          sum(t_{i∈C}.value) ≥ value
4          t_{i∈C}.id == id_from
5      create new set of nodes NC:
6          sum(t_{i∈NC}.value) = value
7          t_{i∈NC}.id = id_to
8          t_{i∈NC}.parent = id_from
9      create new node t:
10         t.value = sum(t_{i∈NC}.value) − value
11         t.id = id_from
12         t.parent = id_from
13     TF ← TF ∪ NC ∪ t
14     HV.EXgen(id, string)
15     if HV.valid(TF) then
16         return ACCEPT
17     else
18         rollback
19         return NACCEPT
```

much effort. Ethereum [11] allows its blockchain network to execute arbitrary user scripts as a smart contract with a fee, which enhanced the ability of verification. However, the validity of external information still depends on the reliability of the information source. In a smart contract implementation, Hawk [20], an additional private layer is introduced to verify the external proof of contract compliance, where its blockchain only takes currency transfer as transactions. This mechanism could be effectively modeled as a distributed ledger combined with a special extension of the hybrid-verifiability transaction system.

To conclude this section, the data from the world outside blockchain (which is modeled as the combination of distributed ledger and hybrid-verifiability transaction system) is not reliable due to a lack of applicable verification mechanism. More precisely, the reliability of data in blockchain depends on the reliability of the data source. Data in the blockchain is differentiated by the properties of the related generation process `INgen`() and `EXgen`() in which `INgen`()

represents reliable data source guaranteed by the mining process, and `EXgen`() represents unreliable source respectively. Therefore, to leverage blockchain as a reliable public ledger, we should add another layer of verification to the `EXgen`() in order to create a reliable data source. We discuss how to establish an additional verification mechanism in chapter 3.

2.3.2 Distributed Ledger Object

In the preceding section, we discussed the ledger and its verifiability on the base of the centralized model. One reason for this treatment is that it simplifies the discussion of verifiability without impacting the generality, because the verifiability of the transaction system does not depend on the ledger object, as demonstrated by the analysis. As a consequence, it is unnecessary to consider the verification when defining the distributed ledger object. A more important observation is that even the consensus does not influence the verifiability of the transaction system; thus, we did not introduce the operation `valid`() as part of the distributed ledger object as Antonio F. Anta, etc. did in their work [19].

Another technical decision is that our definition of distributed ledger is not based on generic deterministic atomic broadcast service [21] since it does not capture the nature of the consensus protocol of blockchain. The models with atomic broadcast, in essence, replace the Byzantine fault tolerance consensus protocol with consensus in the crash-prone system like the one considered in [22].

We describe the distributed ledger object in Algorithms 6, 7, and 8. In the pseudo-code, the mining nodes are modeled as servers in which each node maintains a local copy of the entire blockchain. The consensus algorithm only depends on the length of the blockchain, i.e., the longest version must be accepted unconditionally. Thus, an interesting observation is that although the verifiability of the transaction system prevents the double-spending attack, this attack is still possible in the entire

Algorithm 6: Client code of a distributed ledger $\mathcal{DL}$ executed by process p

```
1  seq ← 0
2  select L ⊆ S : |L| ≥ f + 1
3  function DL.get()
4      seq ← seq + 1
5      send request(seq, p, GET) to the servers in L
6      wait response(seq, p, GET, V) from some i ∈ L
7      return V
8  function DL.append(r)
9      seq ← seq + 1
10     send request(seq, p, APPEND, r) to the servers in L
11     wait response(seq, p, APPEND, res) from some i ∈ L
12     return res
```

Algorithm 7: Consensus algorithm of a distributed ledger $\mathcal{DL}$

```
1  function DL.propose(v)
2      if r ∉ S_i.DL.V then
3          DL.append(v)
4      V_i ← DL.get()
5      if receive(CONSENSUS, V_j) then
6          V_received ← V_j
7      if V_received ≥ V_i then
8          DL.V = V_received
9      else DL.V = V_i
10     broadcast(DL.V)
```

blockchain system setting with a probability depending on the distribution of computational power. The double-spending attack is realized by a rollback attack in essence.

Another problem related to the verification of the transactions is the append-only feature. This feature is related to immutability, which is a desirable property for a ledger in that the finalized records are supposed to be reliable as long as the data source is reliable. However, it could be problematic regarding the flexibility of general-purpose data storage. For instance, a relational database requires a more flexible data structure. The append-only feature limits the range of application scenarios of blockchain. The immutability of blockchain sometimes

Algorithm 8: Server code of a distributed ledger $\mathcal{DL}$ executed by server $i \in S$

init: $S_i \leftarrow \emptyset$
receive(seq, p, GET)
    `send` response(seq, p, GET, S_i) to process p
receive $(seq, p, APPEND, r)$
    $\mathcal{DL}$.`propose`(r)
    `send` response$(seq, p, APPEND, ACK)$ to process p

causes a problem: any operation other than `get`() increases its size (e.g., attributes like address could be modified directly, but it must add a new entry to represent an address change). A possible way of reasonable data modification is to announce a replacement of the incorrect information. This method is flawed in two aspects: 1) it wastes the expensive storage resource in the BC system (i.g both the previous information and its modification are permanently kept in the BC and duplicated among the mining nodes); 2) It is not efficient to retrieve the current status of a specific attribute.

In addition, general purpose recording requires a system with scalability, because of the volume of data entries and frequent incremental and inquiry operations. When putting all the records into blocks, it generates highly frequent transactions which could dysfunction a public blockchain network. As a matter of fact, it takes 10 minutes on average for a Bitcoin block to be mined. For a Bitcoin transaction to be confirmed, it is expected to take an hour, and the overall throughput of the network is limited to around 7 transactions per second [23]. Existing solutions for scaling blockchains consider structures with off-chain databases and on-chain hashes, in which the reliability of the off-chain database remains questionable.

2.4 Other Related Work

Kaiwen Zhang et al. structured the requirements for dependable, scalable, and pervasive distributed ledgers with blockchains, and identifies research challenges to achieve this objective [24]. One of the particular difficulties besides the scalability problem is transaction privacy. Because of the transparency of the ledger, it is possible to construct an activity graph for a particular address. zkLedger [17] attempted to solve this problem for a public auditable ledger by hiding plain information via *Pedersen commitment* and *non-interactive zero-knowledge proofs*. Z. Shae and J. Tsai proposed an approach to transform blockchain into distributed parallel computing architecture for precision medicine [25]. Though different from the purpose of our scheme, it shares the same extension, i.e., to coordinate on-chain and off-chain computation. The challenge of the problem is how to keep the on-chain workload lightweight. Their work depends on an off-chain control node that could help the on-chain program call off-chain arbitrary code to execute the main computation. It works when the entire computation is owned by a single organization. In the presence of collaborators, the output from other participants could be questionable, and thus a verification process is necessary. There are discussions about the incentive mechanism underlying the blockchain-based cryptocurrencies [26, 27]. However, their analysis focuses on the explanation of how the existing schemes work rather than provide a quantified method to dedicatedly design incentive frameworks for different application scenarios.

Chapter 3

VOCMC for Verification and Scalability

3.1 Off-chain Payment Channel and State Channel

Public blockchains insist on the permissionless property of their consensus protocols. As a consequence, the scalability of a public blockchain is limited to a magnitude (about 10 transactions per second) that is not comparable to the transaction system of the traditional centralized ledger (thousands of transactions per second) [28, 29]. One category of scaling solutions [30, 31, 32, 33, 34, 35] is alternative consensus protocols, which fall short of lacking backward compatibility or fundamentally altering the decentralized security assumption. Off-chain payment channels (that are referred to as *layer-two protocols in [36]*) aim to improve the scalability of Blockchain-based cryptocurrencies for fast and frequent payment processing [37, 38] by reducing the transaction load on the underlying blockchain.

According to off-chain payment channel protocols, two parties deposit coins into a shared multi-signature address to open an off-chain payment channel. After opening, the two parties can make payments to each other by agreement on the distribution of the deposit coins without generating any on-chain transaction. The agreement is in the form of a committed transaction. For example, if Alice wants to initiate a payment to Bob via an off-chain payment channel, she signs a transaction indicating the resulted balance. This transaction is a commitment transaction that is not broadcasted to the network immediately. The new payment replaces the preceded commitment transaction. At closure, the blockchain network takes the latest commitment transaction and redistributes the deposit coins.

Unless one party could successfully forge the signature of its counterpart, a dishonest party is only able to cheat by posting an expired commitment transaction to close the payment channel. Therefore, any transaction for channel closure should

be finalized after a timeout enough for the counterpart to react to the dispute. A proven cheater will be punished economically in some way defined by the contract. A disadvantage of this original form of payment channels is that the payment channel is pairwise. Only the participants of the transaction that setup the payment channel could pay each other through it. Thus, the mainstream of research on this area is to connect existing payment channels into an off-chain payment network. If Alice wants to pay Bob, they do not have to set up a payment channel between them, as long as there is a user (we may call her Chris) having a payment channel with both Alice and Bob. Alice pays Chris, then Chris pays Bob. Any number of nodes can be added to the chained payment; thus, any pair of users in this network could pay each other with established additional channels.

Another direction of payment channel research is to generalize it into a so-called state channel [39]. The participants of a state channel monitor and operate with some states in concern. In payment channels (a specific state channel), the interesting state is the deposit paid by the participants. The generalized state channel could accept any variables as the states. This scheme has the potential to broaden the adoption of blockchain applications in areas other than cryptocurrencies, but the potential has not been deeply explored by the researchers. *Raiden* is the most prominent project that implements state channel, but currently, it focuses on the implementation of payment channels via this generalized framework.

Besides the scalability improvement, the off-chain payment channel, or state channel, in fact, guarantees the reliability of external information by probable economic loss resulted from being caught behaving dishonestly. Therefore, we can build a protocol based on this observation. In the next section, we will discuss an underlying trust concept, and we call it *incentive-based trust*, then propose an information verification framework on top of it.

3.2 Incentive-based Trust

There are two folds of security concerns related to off-chain payment channels. The first is that the deposit balance should be correct; the second is that the publishing of the transaction indicating correct, current balance state should be guaranteed. State-of-art research has focused on the publishing problem, i.e., how to guarantee the honest participants could publish the valid commitment transactions by the normal closure or by the dispute process when the counterparts attempt to publish an outdated transaction to rollback the state of the balance. On the other hand, the correctness of the state has not attracted enough attention because the correctness of the states seems a natural property in such an environment. However, correctness is not guaranteed in general. Suppose Alice pays Bob 10 USD for a sandwich, which is described in the transactions, as shown in Fig. 3.1. Although the value of the payment is correct, the additional information provided by the comment could be correct, ambiguous, or absolutely ignored. This scenario is a common case when we examine any type of transaction description. Other examples can be easily found from the online banking applications. We can identify two types of information from the example in Fig. 3.1: I. the payment value whose correctness could directly impact the interest of the participants, II. comment whose correctness may be irrelevant to the benefit of each participant in this case. We call type I as incentive-associated information, if we generalize the financial interest. For off-chain payment channel, the protocol naturally ensures the correctness of the balance state, because the deposit balance is an incentive-associated information. The participants will automatically take care of the balance when reaching an agreement.

Similar to the cryptocurrencies, an important factor for the overall security of the off-chain payment channel is the incentive, which may be more influential than estimated. Though technically the security of the blockchain network requires that

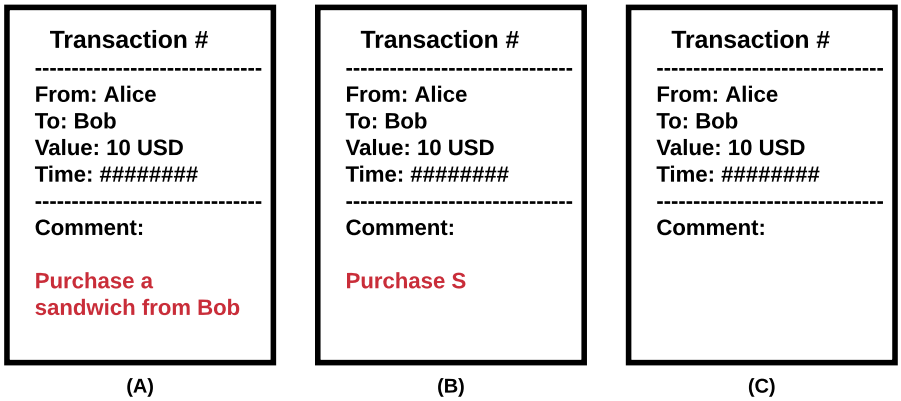

Figure 3.1: A: a transaction with comment describing the behavior related to this payment; B: the comment with partial information; C: comment is ignored.

none of the participants controls a dominant majority of the mining resource, in practice, the formation of mining pools indeed have put threats to this fundamental security assumption. Mining pools are groups of cooperating miners who agree to share block rewards in proportion to their contributed mining hash power [40]. By joining a mining pool, the miners are able to reduce the variance of their mining rewards. At the time of writing, the top 3 leading mining pools hold over 51% share of the computing resource; the biggest pool, BTC.com controls 29.6%; and notably, most of the mining pools are concentrated in China where is estimated 81% of the network hash rate [40]. With the presence of mining pools, Eyal and Sirer proposed the selfish mining strategy that allows a pool with 1/3 of the overall hash power to obtain more revenue than its ratio of the total hash power [41].

Nevertheless, such an attack has not been observed. The pools have been benign and followed the protocol so far [41]. The assumption is that the majority miners may avoid strategies that earn more bitcoins but decrease the expected value of their future mining rewards since a substantial share requires a large amount of investment to maintain [12]. Different from the measurement of traditional security solutions, currently the security or trust strength supported by incentive has not been measured by quantified models, thus how to effectively choose the

reward-penalty combination remains an open question. Despite the lack of precise models, we believe the successful applications demonstrate the feasibility of the frameworks based on this type of security.

In cryptocurrency, the incentive is more rewards than penalties in that the reputation of the system supports the value of the coins. For the off-chain payment channel, the penalty has the main influence. Imagine Bob pays to Alice but files a transaction with less value than expected. Alice double-checks the transaction, then closes the channel once she finds out about the fraud. Bob may lose part of his deposit according to the contract. There is another factor that contributes to the correctness of the states: the direct interest conflict between Alice and Bob. If the state is incorrect, either the interest of Alice or Bob would be damaged. Therefore, to protect their benefit from pillage, Alice and Bob have a strong motivation to carefully monitor the state.

To summarize, we identify three properties that could support effective verification of external information (states), and ensure the reliability of the information published on the blockchain, the public ledger:

- there is an incentive that could effectively encourage honest behaviors;
- the value of the states could alter the distribution of interest, which could lead to direct conflict between the participants; and
- if the incentive strategy is a zero-sum game, the information should be neutral, i.e., an approximation of the truth with negligible error.

For our work, an inspiring observation is that users and service providers may hold opposite interests on policy compliance, if the states of policy compliance are related to the payment process. In consequence, the *incentive-based trust* can be applied. Note that, for other application scenarios, the chance is high that we can set up an environment where the external information can be related to some

incentives and situations with interest conflict, but it is not clear if such an environment always exists. Solving this problem needs a refined model describing its underlying mechanism, which is out of the scope of this dissertation.

3.3 Defining VOCMC

We discuss the two-party variant of VOCMC in this section, where we assume the two parties are users of a blockchain system in which a type of cryptocurrency is defined, and each party holds enough balance of the cryptocurrency in some private address. Upon the establishment, these parties have to transfer some units of the cryptocurrency form their private addresses into a 2-of-2 multi-signature address as the deposits, i.e., any transfer from the 2-of-2 address requires 2 signatures to authorize. This framework could be simply extended to n-party cases by applying an n-of-n multi-signature address and adjusting protocol synchronization, respectively.

We follow the sketch used in [42] to describe the VOCMC protocol. The protocol essentially relies on the composition of protocols, and thus we breakdown the entire protocol into some component protocols universally composable [43], i.e., a synchronous version of the UC framework [44]. More precisely, the parties who run the protocol are modeled as interactive poly-time Turing machines (ITM) [45] whose inputs are from the *environment* $\mathcal{E}$, which is modeled as an ITM as well. The environment $\mathcal{E}$ represents anything "external" to the execution of the current protocol instance, e.g., the inputs from the users, other running instances of protocols, etc. Further, we define ideal functionalities for the on-chain executed modules and define the entire protocol as running in a hybrid model with the ideal functionalities. We will discuss the security implications of this hybrid model in section 3.4.

In the rest of this section, we will first discuss the ideal functionalities handling cryptocurrency; second, discuss the VOCMC by dividing it into

Algorithm 9: Ledger Functionality $\mathcal{F}_{\mathcal{L}}$

Data: a **balance** vector$(x_1, ..., x_n) \in R_+^n$
Input: message from environment $\mathcal{E}$
Upon **receive** message(`add`, sid, $\{(p_{i_j}, y_{i_j})_{j=1}^t | t \in Z_{n+}, y_{i_j} \in R_+\}$):
begin
  forall $j \in \{1, ..., t\}$ **do**
    $x_{i_j} := x_{i_j} + j_{i_j}$
  end
end
Upon **receive** message(`remove`, sid, $\{(p_{i_j}, y_{i_j})_{j=1}^t | t \in Z_{n+}, y_{i_j} \in R_+\}$):
begin
  if *(*$x_{i_j} \leq y_{i_j} \forall j \in 1, ..., t$*)* **then**
    `reply` message(`NOFUNDS`, cid)
    `stop`
  else
    forall $j \in \{1, ..., t\}$ **do**
      $x_{i_j} := x_{i_j} - j_{i_j}$
    end
  end
end

subroutines including channel creation, contract registration, off-chain update, and channel closure. Each discussion contains its on-chain ideal functionality and off-chain local components.

3.3.1 Ledger Functionality and Incentive Function

This functionality handling cryptocurrency is separated from other on-chain functionalities because it is a build-in mechanism offered by the public blockchain system, and thus not a component of the VOCMC but a critical dependence.

Following [46], we model the cryptocurrency mechanism as a special functionality $\mathcal{F}_{\mathcal{L}}$ by which the balance owned by each party is tracked, and currency transfer is made via operations `remove` and `add` (see Algorithm 9). The "transfer" operation is represented by two separate operations `remove` and `add` to simplify the expression, though the underlying execution logic of blockchain is, in effect,

Algorithm 10: Incentive function $\texttt{Inc}(ev_1, ..., ev_n)$

Data: a vector of party identifiers$(p_1, ..., p_n)$, a vector of initial balances$(b_1, ..., b_n)$, a balance pool bp, pool identifier(p_0)
Input: a vector of evidences $(ev_1, ..., ev_n)$
begin
  create a reward vector$(r_1, ..., r_n | \sum_i^n r_i == bp)$ according to $(ev_1, ..., ev_n)$
  forall $j \in \{1, ..., n\}$ **do**
    `send` message$(\texttt{remove}, sid, p_0, r_j)$ to ledger functionality $\mathcal{F}_\mathcal{L}$
    `send` message$(\texttt{add}, sid, p_j, r_j)$ to ledger functionality $\mathcal{F}_\mathcal{L}$
  end
end

integrated.

Note, the balance vector $(x_1, ..., x_n)$ includes n elements, where n is the number of existing parties $(p_1, ..., p_n)$. The state of the functionality $\mathcal{F}_\mathcal{L}$ is always public, i.e., the environment $\mathcal{E}$, the parties $(p_1, ..., p_n)$, and any potential adversary $\mathcal{A}$ have free access to its contents. The vectors $(x_1, ..., x_n)$ and $(p_1, ..., p_n)$ are abstractions of account and balance in the blockchain concept since they are not maintained explicitly in the blockchain ledger. A party p_i corresponds to a set of m addresses $(add_1^i, ..., add_m^i)$ where m could be any positive integer. The balance x_i for a party p_i is collectively represented by all the transactions on the ledger, related to the party p_i. The underlying detail is omitted in the rest of the discussion. The balance vector $(x_1, ..., x_n)$ and the party vector $(p_1, ..., p_n)$ possess capable expressive power for the description of this protocol.

The incentive function $\texttt{Inc}(ev_1, ..., ev_n)$ is part of the VOCMC protocol (see Algorithm 10). Since it encapsulates the calling of the ledger functionality $\mathcal{F}_\mathcal{L}$ and is the only interface for the protocol accessing the ledger functionality $\mathcal{F}_\mathcal{L}$ after channel creation, we describe it together with $\mathcal{F}_\mathcal{L}$.

It holds an initial balance vector $(b_1, ..., b_n)$ which is different from the balance vector $(x_1, ..., x_n)$, i.e., the deposit the parties put into the VOCMC channel space, which is frozen by the channel from the ledger balance $(x_1, ..., x_n)$. The frozen

balance will finally be redistributed according to some reward or penalty policy. An evidence vector $(ev_1, ..., ev_n)$ is needed to call this function, for the reward policy to generate the reward vector $(r_1, ..., r_n)$. Since `Inc()` could only redistribute the frozen balance, the reward vector should satisfy the constraint, $\sum_i^n r_i = \sum_i^n b_i = bp$, where bp indicates the total value of the frozen balance. After the decision of the reward vector, the balance will be transferred to the parties' accounts respectively from the frozen account (denoted as p_0 in Algorithm 10).

3.3.2 Channel Creation

In order to create a new VOCMC channel, the environment $\mathcal{E}$ has to send message$_{\mathcal{E}}$ (`CREATE`, cid, pid_a, pid_b) to the instance of local protocol $\Pi(\mathcal{L})$ (see Algorithm 11) run by party p_{pid_a} to initiate the creation process. cid indicates the global identifier of the channel to be created, and pid_b indicates the other party that the channel will be created in between. The protocol instance of the initiating party p_{pid_a} sends to the on-chain ideal functionality $\mathcal{F}_{ch}$ the message (`CONSTRUCT`) in which `CODE_INC` contains the incentive function code, the reward policy, and the required initial value for the balance vector, to request construction of the VOCMC contract. The ideal functionality (see Algorithm 12) freezes the required initial balance in the account identified by cid on the ledger and send message$_{\mathcal{F}}$(`INITIALIZING`) to the other party identified by p_{pid_b}.

Upon receiving the message$_{\mathcal{F}}$(`INITIALIZING`) and message$_{\mathcal{E}}$ (`CREATECONF`, cid, pid_a, pid_b), the party p_{pid_b} responds with the message(`CONFIRM`) to confirm the initialization. Then the ideal functionality $\mathcal{F}_{ch}$ freezes the required initial balance for party p_{pid_b} and sends message$_{\mathcal{F}}$ to both parties. Otherwise, there is a timeout triggered in the protocol instance of party p_{pid_a} who thereafter has the option to refund the initial balance frozen during the initiating process.

Note, the communication between the environment $\mathcal{E}$ and the other entities

Algorithm 11: VOCMC Protocol $\Pi(\mathcal{L})$: Create channel

Input: $message_{\mathcal{F}}$ from on-chain functionality $\mathcal{F}_{ch}$, $message_{\mathcal{E}}$ from environment $\mathcal{E}$
Upon **receive** $message_{\mathcal{E}}(\texttt{CREATE}, cid, pid_a, pid_b)$:
begin
  `send` message$(\texttt{CONSTRUCT}, cid, pid_a, pid_b, \texttt{CODE_INC})$ to on-chain functionality $\mathcal{F}_{ch}$
  `wait` up to 2Δ:
  begin
    if ***receive*** $message_{\mathcal{F}}$*(INITIALIZED*$, cid$*)* **then**
      `send` message$(\texttt{CREATED}, cid, pid_a, pid_b)$ to $\mathcal{E}$
      `stop`
    else
      if ***receive*** $message_{\mathcal{E}}$*(REFUND*$, cid$*)* **then**
        `send` message$(\texttt{REFUND}, cid, pid_a)$ to $\mathcal{F}_{ch}$
        `stop`
      end
    end
  end
end
Upon **receive** $message_{\mathcal{E}}(\texttt{CREATECONF}, cid, pid_a, pid_b)$:
begin
  if ***receive*** $message_{\mathcal{F}}$*(INITIALIZING*$, cid, pid_a, pid_b$*)* **then**
    `send` message$(\texttt{CONFIRM}, cid, pid_a, pid_b)$ to $\mathcal{F}_{ch}$
    `wait` up to Δ:
    if ***receive*** $message_{\mathcal{F}}$*(INITIALIZED*$, cid$*)* **then**
      `send` message$(\texttt{CREATED}, cid, pid_a, pid_b)$ to $\mathcal{E}$
      `stop`
    end
  end
end

in the protocol is modeled as taking no time, or this can be achieved by defining the allowed communication time Δ between the parties other than the environment as long enough to neglect the communication with the environment. Therefore, the initiating party p_{pid_a} has to wait up to 2Δ to let the party p_{pid_b} complete the communication for confirmation.

Algorithm 12: On-chain Functionality $\mathcal{F}_{ch}$: Create channel

Data: a vector of party identifiers$(p_1, ..., p_n)$, initializing list, active list
Input: message from the parties
Upon **receive** message(`CONSTRUCT`, cid, pid_a, pid_b, `CODE_INC`):
begin
  if *($vocmc_{cid} \notin$ initializing list)* && *($\mathcal{F}_{\mathcal{L}}.x_{pid_a} \geq$ CODE_INC.b_{pid_a})* **then**
    `send` message(`REMOVE`, $sid, p_{pid_a}, b_{pid_a}$) to ledger functionality $\mathcal{F}_{\mathcal{L}}$
    `send` message(`ADD`, sid, cid, b_{pid_a}) to ledger functionality $\mathcal{F}_{\mathcal{L}}$
    `add` the pair $(vocmc_{cid}, \tau_0)$ into initializing list
    `send` $message_{\mathcal{F}}$(`INITIALIZING`, cid, pid_a, pid_b) to p_{pid_b}
  end
end
Upon **receive** message(`CONFIRM`, cid, pid_a, pid_b):
begin
  if *($vocmc_{cid} \in$ initializing list)* && *($\mathcal{F}_{\mathcal{L}}.x_{pid_b} \geq$ CODE_INC.b_{pid_b})* && $(\tau - \tau_0 \leq \Delta)$ **then**
    `send` message(`REMOVE`, $sid, p_{pid_b}, b_{pid_b}$) to ledger functionality $\mathcal{F}_{\mathcal{L}}$
    `send` message(`ADD`, sid, cid, b_{pid_b}) to ledger functionality $\mathcal{F}_{\mathcal{L}}$
    `remove` the pair $(vocmc_{cid}, \tau_0)$ from initializing list
    `add` $vocmc_{cid}$ into active list
    `send` $message_{\mathcal{F}}$(`INITIALIZED`, cid) to p_{pid_a}, p_{pid_b}
  end
end
Upon **receive** message(`REFUND`, cid, pid_a):
begin
  if *($vocmc_{cid} \in$ initializing list)* && $(\tau - \tau_0 > \Delta)$ **then**
    `send` message(`REMOVE`, sid, cid, b_{pid_a}) to ledger functionality $\mathcal{F}_{\mathcal{L}}$
    `send` message(`ADD`, $sid, p_{pid_a}, b_{pid_a}$) to ledger functionality $\mathcal{F}_{\mathcal{L}}$
    `remove` the pair $(vocmc_{cid}, \tau_0)$ from initializing list
  end
end

3.3.3 Local Contract Instance Update

Denote any input to the VOCMC at time t before the expected registration time T_E as $m_{t,0\leq t\leq T_E}$. The corresponding output to the blockchain network is generated from $\Phi(m_0, m_1, \cdots, m_t)$ where the input to the method $\Phi(\cdot)$ is the input sequence till t. For convenience, we denote the output at time t as Φ_t, unless ambiguity occurs.

The protocol is described in Algorithm 13. Similar to the channel creation,

Algorithm 13: VOCMC Protocol $\Pi(\mathcal{L})$: Contract update

```
1  Data: round number br, current output Φ_t
2  Input: message_E from environment E, message from other party
3  Upon receive message_E(UPDATE, cid, pid_a, pid_b):
4  begin
5      compute S_a ← sign(Φ_t, pid_a, pid_b, br + 1)
6      send message(UPDATE, sid, br + 1, S_a) to p_{pid_b} at τ_0
7      wait up to Δ:
8      begin
9          if receive message(UPDATEOK, sid, br + 1, S_b) then
10             if valid(S_b) then
11                 br = br + 1, create TemTran(t, Φ_t, br, S_a, S_b)
12         else if receive message(NEGOTIATE, m_nr, sid, nr) then
13             if nr < N then
14                 m_{nr+1} ← message_E
15                 send message(NEGOTIATE, m_{nr+1}, sid, nr + 1)
16                 goto line 7
17             finalize negotiation
18         else
19             m_0 ← message_E
20             send message(NEGOTIATE, m_0, sid, 0)
21             goto line 7
22 Upon receive message_E(UPDATECONF, cid, pid_a, pid_b):
23 begin
24     if (receive message(UPDATE, sid, br + 1, S_a)) &&(message_E : OK)
        &&(valid(S_a)) then
25         compute S_b ← sign(Φ_t, pid_a, pid_b, br + 1)
26         send message(UPDATEOK, sid, br + 1, S_b)
27         br = br + 1, create TemTran(t, Φ_t, br, S_a, S_b)
28     else
29         if message_E : NOTOK then
30             m_0 ← message_E, send message(NEGOTIATE, m_0, sid, 0)
31             wait up to Δ:
32             if (receive message(NEGOTIATE, m_nr, sid, nr))&&(nr < N) then
33                 m_{nr+1} ← message_E
34                 send message(NEGOTIATE, m_{nr+1}, sid, nr + 1)
35                 goto line 31
36             finalize negotiation
```

the environment $\mathcal{E}$ sends message$_{\mathcal{E}}$(UPDATE) and message$_{\mathcal{E}}$(UPDATECONF) in order to instruct party p_{pid_a} to initiate the update process and party p_{pid_b} to respond, respectively. The initiating party p_{pid_a} then signs the bit string comprising the output Φ_t, the identifier of both parties, and a round number br (the round number indicates the current round of update which is in effect increased upon a successful update) via a public key encryption system, then sends the output and the signature S_a by an update request.

When receiving the update request, the party p_{pid_b} needs another environment input to determine if the output Φ_t matches her local record. If there is no disagreement on current output, and the signature S_a is valid, then party p_{pid_b} will sign the same bit string and send the signature S_b in an UPDATEOK response. Both parties will create and hold a temporary on-chain transaction $\texttt{TemTran}(t, \Phi_t, br, S_a, S_b)$.

Different from the off-chain payment channel, there is a particular difficulty when applying state channel for data verification: a participant may intentionally refuse to create a transaction with output Φ_t that may impact its benefit. To do so, the participant could refuse to respond to the latest UPDATE request and close the channel immediately. Thereafter, the latest output Φ_t will never be published to the ledger. We introduce a negotiation procedure to address this issue. Once party p_{pid_b} refuses to respond to an UPDATE request, the counterpart could initiate negotiation by sending a request message(NEGOTIATE, m_{nr}, sid, nr), where nr is the round number of negotiation. A deadline Δ is set for the other party to respond to the negotiation. The negotiation could also be initiated if there is a disagreement on the output, in which case the environment $\mathcal{E}$ indicates the disagreement by NOTOK message.

If a negotiation request is ignored, a $\texttt{TemTran}(t, \texttt{NEG}_t)$ will be automatically created with default signatures. If the negotiation is responded to, but without

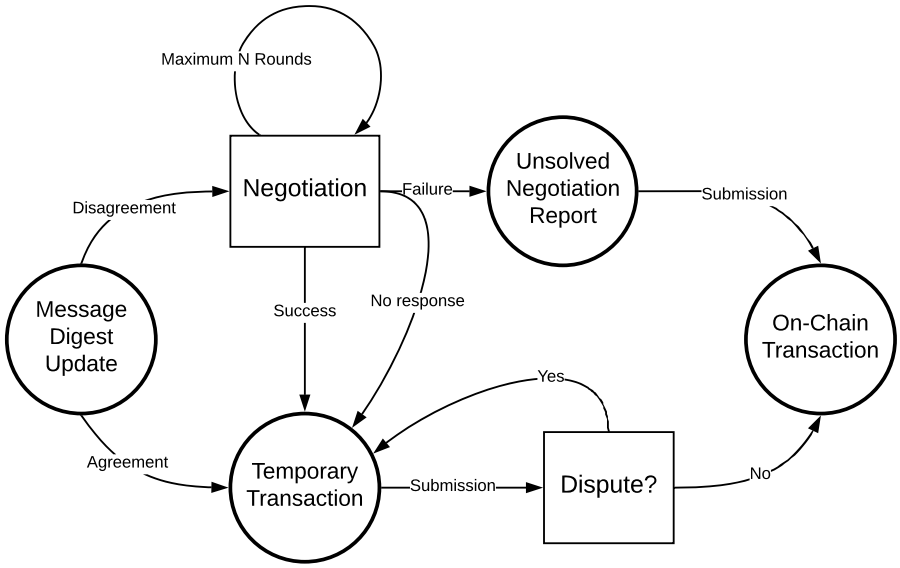

Figure 3.2: State transition in VOCMC. Negotiation reports indicate that the information might be unreliable, but the details could help the public make decision by their own judgement.

reaching an agreement, the negotiation will stay open with a new deadline for further communication, and thus a VOCMC can not close with any negotiation still ongoing. If the parties could not reach an agreement after N rounds (the maximum of rounds allowed) of negotiation, a transaction with information of the negotiation proceedings, $\texttt{TemTran}(t + N\Delta, \texttt{NEG}_{N\Delta})$, will be automatically created and submitted to the public ledger. This transaction serves as a report of unsolved negotiation. Fig 3.2 illustrates the state transitions during the execution of the VOCMC. The transaction of the negotiation report would not serve as the evidence for a reallocation of the deposit and may be unreliable because it is unwise to accept the explanation from any one of the parties with disagreement. Nevertheless, the negotiation report is still a valuable information source in that it provides more details than regular transactions.

Algorithm 14: VOCMC Protocol $\Pi(\mathcal{L})$: Contract registration

Data: list of `TemTran`
Input: $message_F$ from on-chain functionality $\mathcal{F}_{ch}$, $message_E$ from environment $\mathcal{E}$
Upon **receive** $message_{\mathcal{E}}(\texttt{REGISTER}, cid, pid_a, pid_b)$:
begin
  `send` message$(\texttt{REGISTER}, cid, pid_a, pid_b, \texttt{TemTran}^T_{pid_a})$ to $\mathcal{F}_{ch}$ at τ_0
  `wait` up to Δ:
  begin
    if ***receive*** $message_{\mathcal{F}}(\textit{REGISTERING}, cid, \textit{TemTran}^T_{pid}.br)$ **then**
      `wait` up to Δ:
      if ***receive*** $message_{\mathcal{F}}(\textit{REGISTERED}, cid, \textit{TemTran}^T_{pid}.br)$ **then**
        `mark` $\texttt{TemTran}^T_{pid}$ as `REGISTERED`
      else
        `send` message$(\texttt{FINALIZE}, cid, pid_a, \texttt{TemTran}^T_{pid_a})$ to $\mathcal{F}_{ch}$
      end
    end
  end
end
Upon **receive** $message_{\mathcal{E}}(\texttt{REGISTERCONF}, cid, pid_a, pid_b)$:
begin
  if ***receive*** $message_{\mathcal{F}}(\textit{REGISTERING}, cid, \textit{TemTran}^T_{pid_a})$ **then**
    `send` message$(\texttt{REGISTER}, cid, pid_a, pid_b, \texttt{TemTran}^T_{pid_b})$ to $\mathcal{F}_{ch}$
    `wait` up to Δ:
    if ***receive*** $message_{\mathcal{F}}(\textit{REGISTERED}, cid, \textit{TemTran}^T_{pid}.br)$ **then**
      `mark` $\texttt{TemTran}^T_{pid}$ as `REGISTERED`
    end
  end
end

3.3.4 Contract Instance Registration

The local protocol instance (see Algorithm 14) maintains a list of temporary transactions in its local storage. Recall that temporary transaction $\texttt{TemTran}(t, \Phi_t, br, S_a, S_b)$ is a contract instance containing the output Φ_t that could be uploaded to the public ledger (thereafter, a temporary transaction will be denoted as $\texttt{TemTran}^T$). The procedure to submit a $\texttt{TemTran}^T$ to be mined and appear on the ledger and thus available to the public is called *"contract instance*

Algorithm 15: On-chain Functionality $\mathcal{F}_{ch}$: Contract registration

Data: registering list
Input: message from the parties
Upon **receive** message($\texttt{REGISTER}, cid, pid_a, pid_b, \texttt{TemTran}^T_{pid_a}$):
begin
  if *valid*$(signature_{pid_a})$ && *valid*$(signature_{pid_b})$ **then**
    if $\textit{TemTran}^T_{pid_b} \in$ *registering list* **then**
      if $\textit{TemTran}^T_{pid_b}.round < \textit{TemTran}^T_{pid_a}.round$ **then**
        replace ($\texttt{TemTran}^T_{pid_b}$, τ_0) with ($\texttt{TemTran}^T_{pid_a}$, $\hat{\tau}_0$)
        wait up to Δ
      end
    else
      add ($\texttt{TemTran}^T_{pid_a}$, τ_0) into registering list
      send $message_{\mathcal{F}}(\texttt{REGISTERING}, cid, \texttt{TemTran}^T_{pid_a})$ to p_{pid_a} and p_{pid_b}
      wait up to Δ
    end
    remove $\texttt{TemTran}^T_{pid}$ from registering list
    register $\texttt{TemTran}^T_{pid}$
    send $message_{\mathcal{F}}(\texttt{REGISTERED}, cid, \texttt{TemTran}^T_{pid}.br)$ to p_{pid_a} and p_{pid_b}
  end
end
Upon **receive** message($\texttt{FINALIZE}, cid, pid_a, \texttt{TemTran}^T_{pid_a}$):
begin
  if $(\textit{TemTran}^T_{pid_a} \in$ *registering list)* && $(\tau - \tau_0 \geq 2\Delta)$ **then**
    remove $\texttt{TemTran}^T_{pid}$ from registering list
    register $\texttt{TemTran}^T_{pid}$
    send $message_{\mathcal{F}}(\texttt{REGISTERED}, cid, \texttt{TemTran}^T_{pid}.br)$ to p_{pid_a} and p_{pid_b}
  end
end

registration" [42]. Registration could be invoked anytime the newest $\texttt{TemTran}^T$ is unregistered. An enforced registration point is the channel closure, where the latest $\texttt{TemTran}^T$ must be registered.

The environment $\mathcal{E}$ instructs the party p_{pid_a} to initiate the registration process and the party p_{pid_b} to respond, respectively. Upon the initiating instruction, party p_{pid_a} sends to the on-chain functionality $\mathcal{F}_{ch}$ the REGISTER message, which contains the latest $\texttt{TemTran}^T$, i.e., if there are multiple un-registered temporary

transactions, take the latest and ignore the rest.

The on-chain registration functionality (Algorithm 15) verifies the validity of the signatures. If valid, the temporary transaction $\texttt{TemTran}^T$ will be added to the list of transactions under the registration process (whose members are currently *unregistered*), and a `REGISTERING` will be sent to both parties. Party p_{pid_a} resets the timer upon receiving the `REGISTERING` message, while party p_{pid_b} immediately sends its own copy of the latest temporary transaction $\texttt{TemTran}^T_{pid_b}$ as a reaction. If both the copies from party p_{pid_a} and party p_{pid_b} contain the same round number br, the functionality $\mathcal{F}_{ch}$ then removes the temporary transaction from the registering list and proceeds it into the mining process. Then, it sends the `REGISTERED` message to both parties. A different round number indicates a dispute situation, which has to be settled by determining the largest round number br. The temporary transaction with the largest br is submitted for mining. In case party p_{pid_b} did not respond to the registering request, party p_{pid_a} could force the registration of her own version of the temporary transaction, if a timeout is triggered after receiving the `REGISTERING` message from the on-chain functionality $\mathcal{F}_{ch}$, by sending a `FINALIZE` message.

3.3.5 Channel Closure

In order to close a VOCMC channel, the latest temporary transactions must be registered. Therefore upon receiving the message$_{\mathcal{E}}$(`CLOSE`) from environment $\mathcal{E}$, the protocol instance (see Algorithm 16) of party p_{pid_a} will firstly check if the $\texttt{TemTran}^{T_E}$ is marked as registered. If true, then send message(`CLOSE`, cid) to on-chain functionality $\mathcal{F}_{ch}$ to initiate the closure process.

The functionality $\mathcal{F}_{ch}$ sends message$_{\mathcal{F}}$(`CLOSING`, cid) to party p_{pid_b} and waits up to Δ for response. Upon receiving the `CLOSING` message, if her own version of the latest temporary transaction is NOT registered, party p_{pid_b} should reply with a `NOTCLOSE` message to stop the closure process for the chance to register the

Algorithm 16: VOCMC Protocol $\Pi(\mathcal{L})$: Close channel

Data: list of `TemTran`
Input: $message_E$ from environment $\mathcal{E}$
Upon **receive** $message_{\mathcal{E}}(\texttt{CLOSE}, cid, pid_a, pid_b)$:
begin
  if $\mathit{TemTran}^{T_E}$ *marked* `REGISTERED` **then**
    `send` message(`CLOSE`, cid) to $\mathcal{F}$
    `wait` up to 3Δ:
    if ***receive*** $message_{\mathcal{F}}$*(*`CLOSED`$, cid$*)* **then**
      `send` message(`CLOSED`$, cid, pid_a, pid_b$) to $\mathcal{E}$
    end
  end
  `stop`
end
Upon **receive** $message_{\mathcal{E}}(\texttt{CLOSECONF}, cid, pid_a, pid_b)$:
begin
  if ***receive*** $message_{\mathcal{F}}$*(*`CLOSING`$, cid$*)* **then**
    if $\mathit{TemTran}^{T_E}$ *marked* `REGISTERED` **then**
      `wait` up to Δ:
      if ***receive*** $message_{\mathcal{F}}$*(*`CLOSED`$, cid$*)* **then**
        `send` message(`CLOSED`$, cid, pid_a, pid_b$) to $\mathcal{E}$
      end
    else
      `send` message(`NOTCLOSE`, cid) to $\mathcal{F}$
      `stop`
    end
  end
end

temporary transaction. Otherwise, party p_{pid_b} just sets a timer to wait for additional messages.

If a timeout is triggered after sending the `CLOSING` message, the functionality $\mathcal{F}_{ch}$ will execute the incentive function to reallocate the frozen balance according to the evidence vector $(ev_1, ..., ev_n)$, remove $vocmc_{cid}$ from the active channel list, and send message$_{\mathcal{F}}$(`CLOSED`$, cid$) to both parties. When received the `CLOSED` message, the protocol instances of the parties then signal the environment $\mathcal{E}$ the success of channel closure for further instructions.

Note, we do not inline the registration process into the channel closure

Algorithm 17: On-chain Functionality $\mathcal{F}_{ch}$: Close channel

Data: active list, a vector of evidence$(ev_1, ..., ev_n)$
Input: message from the parties
Upon **receive** message(`CLOSE`, *cid*):
begin
    `send` message$_{\mathcal{F}}$(`CLOSING`,cid)
    `wait` up to Δ:
    if ***receive*** *message(`NOTCLOSE`, cid)* **then**
        `stop`
    else
        `execute` `Inc`$(ev_1, ..., ev_n)$
        `remove` vocmc$_{cid}$ from active list
        `send` message$_{\mathcal{F}}$(`CLOSED`,cid)
    end
end

protocol for a concise description. It is the users' responsibility for settling any unregistered temporary transactions and open negotiations. The functionality only prevents premature channel closure.

3.4 Security Analysis of the VOCMC

3.4.1 Sketch of Universally Composable Security

It is necessary to understand the *universally composable security* [43] framework in order to thoroughly examine the security properties of the VOCMC protocol. This paradigm significantly simplified the discussion of protocol security in case complex patterns of subroutine and communication are adopted for rich enough representation of more realistic adversarial behaviors. In such cases, a proof based on the single execution of a protocol instance cannot ensure its security under parallel and concurrent composition, e.g., Oblivious Transfer [47, 48, 49] whose initial definition could not guarantee security when multiple instances executed concurrently.

Instead of extending the security requirements of the composed protocol (as

in [50, 51, 52, 53]) which results in ever-growing complexity of definitions, the universally composable security framework adopts protocols that are secure in isolation with the additional requirement of *universally composability*: if a protocol is secure under isolated execution and universally composable, its combination with other universally composable protocols is secure as well.

Therefore, we could securely compose complex protocols by combining secure subroutine protocols through a general composition process. In this section, we only consider the subroutine composition, which is the primary composing type adopted in the VOCMC protocol. The crux of this approach is to define the subroutine protocol in a way that is secure in an interactive environment, i.e., the subroutine protocol should UC-emulate [43] a trusted ideal functionality that preserves all the security requirements for the subroutine. Let ϕ denote the subroutine, and π denote the ideal functionality, respectively. The protocol ϕ **UC-emulates** π, if an interactive environment $\mathcal{E}$ could not distinguish if it interacts with the protocol ϕ with the presence of an adversary $\mathcal{A}$ in the real world, or interacts with the ideal functionality π with the presence of an adversary $\mathcal{S}$.

Given a protocol ϕ that UC-emulates π, any party including the adversary could not gain more information from an instance of ϕ than an instance of the ideal functionality π, since adversary $\mathcal{A}$ here takes the place of the interactive environment $\mathcal{E}$. To define universally composable protocols, besides carefully protecting the secrets by the ideal functionality, the communication sequence and timing should be preserved in both the real and ideal world, which is the main effort when sketching the constructive security proof. Typically, the ideal world adversary $\mathcal{S}$ should be constructed capable of relaying the communication flow between the corrupted parties and the environment and determine the delay of the communication channel.

3.4.2 Security of VOCMC

The security proof of VOCMC is relatively straight forward since the subroutines related to security requirements mostly rely on on-chain functionalities, thus avoid the difficulty of handling secure computations coordinated with the corrupted parties. The ideal functionality could be directly replaced by on-chain functionality without an impact on the security of protocols since the on-chain computation is supposed to be executed in a secure environment. The assumption of a secure execution environment of blockchain is based on its fundamental assumption of decentralized computation resources. It is equivalent to run the secure core of the protocol by a trusted third party (note, there is no trusted mining node in the blockchain network, but the network as an entity is trusted). As a consequence, the security proof is reduced to relaying the communication between the environment and the parties, which is straight forward. The detailed construction of the message relay could be found in [42]. This section will discuss some remarkable features of the functionalities.

Fairness of update. Since there is no secret involved in the message exchange process except the encryption key, we do not consider the fairness of information closure in this case. Remember that the initiating party of the update process will send its own signature along with the initiating message to its counterpart, who could intentionally not respond with the other signature required for creating the temporary transaction. As a result, the initiating party could not create the latest temporary transaction, while its counterpart could. In the current setting, this flaw of fairness does no harm the protocol, because only when a party intends to publish the output, she would initiate the update process. This flaw would not lead to blockage of information publication. The fairness of updates could be achieved by introducing an on-chain functionality for fair signature disclosure.

Registration security. Different from off-chain payment channel, registration of temporary transaction would not always lead to deposit reallocation, and thus is balance secure. The dispute-like procedure adopted in the protocol is for confirmation of the agreement on the published information and provides an opportunity to identify data inconsistency. Note that the submitted transaction should contain signatures of both parties and has a greater round number, and the channel would not be closed after a registration; therefore, the registration of an older temporary transaction would not prevent the latest from registration.

Channel closure and balance security. The fairness of channel closure is critical in the VOCMC protocol because, upon the closure of a channel, any unpublished information would be lost, and the frozen balance would be distributed by the incentive function. If there is any unregistered temporary transaction or unsolved negotiation, it is possible that the incentive function is executed based on incomplete or outdated evidence vector and thus incorrect balance distribution. In the current setting, the channel closure could be stopped directly by the environment intervention or known unregistered temporary transactions. This provides an opportunity for malicious prevention of channel closure, e.g., prevent any channel closure if a party is not satisfied with the current potential outcome of the incentive function. In order to solve this problem, the channel closure process could depend on the completeness of registration and negotiation. But, a malicious party could initiate an infinite negotiation process to achieve the same effect. To avoid this problem, we could introduce an on-chain functionality to freeze the creation of negotiation. Only the negotiation open at the initiating time of the channel closure would be taken into account.

3.5 Other Related Work

The literature on off-chain protocols is rich. A remarkable large portion of the work focuses on scaling the blockchain in order to enhance its throughput of payment transactions. Payment channel networks (PCNs) [37, 54] enable the users to route payment through a network of existing payment channels, thus avoid unnecessary channel creation among every pair of channel users. PCNs require routing protocols [55, 56, 57, 58] to optimize the path searching and secure the chained payment. Commit chain [59, 60] solved this problem without a network of payment channels but established a pool that money is freely transferred among the participants. [61, 62, 63, 64] further improved the privacy of the off-chain payment channel, which is important because the off-chain channel naturally leads to information concentration related to a small group of users. Arbitrum [65] and TrueBit [66] scaled smart contract execution by only verifying the signature of endorsers through on-chain execution, and smart contract execution and verification are purely off-chain, which is similar to the methodology of our work. Augur [67] also relies on human knowledge, but allows the users to open a public decision-making process where any number of participants could join in and provide inputs to help the process initiator decide the output and collect their payouts.

Another related research area is the *oracle* system, which aims to provide data to a blockchain in a more reliable way. However, proposed systems largely differ in assumptions and applicability. TLS-N [68] focused on data authenticity and integrity by a modification to the transportation layer security protocol. Similarly, TownCrier [69] guarantee authenticated data feeding to the blockchain by employing trusted execution environments [70, 71]. Astaea [72] and Shintaku [73] leverage on participants' domain knowledge and a voting scheme to ensure reliable data feeding to smart contracts.

From the perspective of game theory, literature analyzed the incentive mechanism using Nash equilibrium and proposed alternatives depending on different game constructions. [74, 75] examined incentive compatibility in the presence of selfish mining behavior of mining pools. [76] improved the proof-of-stake consensus protocol by incentivizing miners to propagate blocks as soon as possible using the Stackelberg equilibrium [77]. The Stackelberg game was as well used in [78] to design a privacy-aware data sharing protocol via blockchain against the security attacks from the external world.

Chapter 4

System Design and Implementation

4.1 Introduction

Determining whether an organization complies with the policies is a challenging task, given the complexity of the current application environment. A data provenance view is required for dealing with the cross-boundary problem, which is introduced by the ever frequent storage and process outsourcing. Fig. 4.1 illustrates a relatively "simple" scenario where a single medical application is described. The imaginary application employs a social robot that could recognize oral instructions from human speech and take actions to complete tasks such as searching for music or traffic conditions. The robot is connected to a PC client and a mobile client for local configuration, control, and application interface. The robot provider maintains a data warehouse for storage of the user data, and outsource the speech recognition to a SaaS provider who is specialized in natural language processing. The training data for speech recognition model is acquired from the data warehouse. When the recognition model is ready, the robot could directly send realtime speech record to the SaaS for analysis.

In order to leverage the VOCMC, one of the critical facts is the agreement. Hence, our proposed scheme requires the user, and the service provider could make an independent judgment on policy compliance. We assume policy compliance inference engines are deployed at both the user side and the provider side. The scheme should ensure that the required evidence id sent to both the inference engines. To achieve this goal, we recommend a *sticky-policy*-based framework to track the data footprint and enforce the policy application.

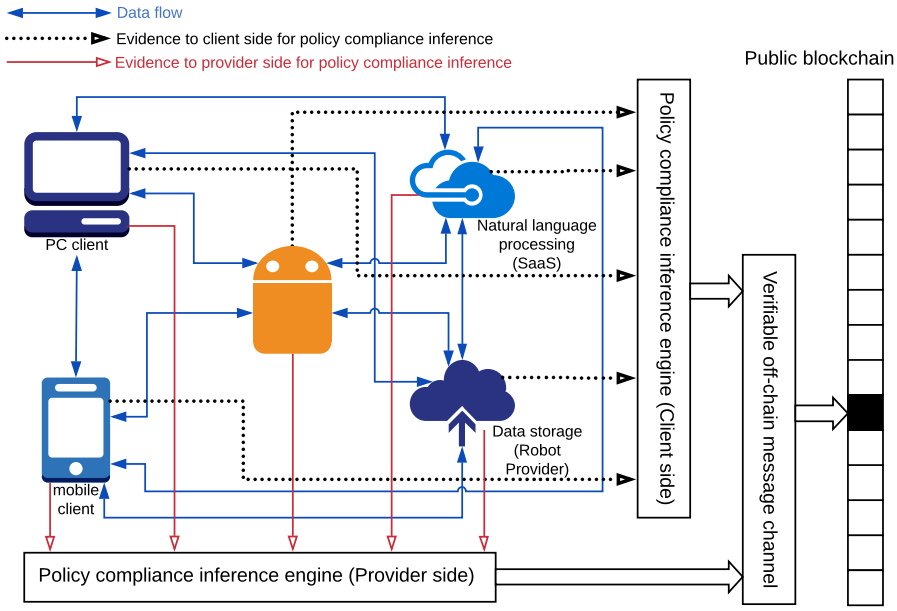

Figure 4.1: High level description of the proposed scheme with example

4.2 Background: Sticky Policy

Policies can stick to data to define allowed usage and obligations as it travels across multiple parties, platforms, or administrative domains, enabling users to improve control over their personal information [79]. The *sticky policy* is a potential approach for accountable and enforceable policies [80, 2], which is firstly introduced by Casassa-Mont *et al.* [80]. Here, the data owner specifies management constraints that would be attached to the data by establishing contractual relationships between data owners and service providers. In order to enforce the owner-specified policies, a trusted authority (TA) is employed to keep the decryption keys of the encrypted data. To obtain the decryption key, a party must agree to enforce the policies associated with the data. Typically, the policies are enforced at the application level and are tend to be enforced at particular points, e.g., at administrative boundaries. Since the data is released as the decryption key is released, the data owner loses the control and track over her data after the release

point. This mechanism is extended by the following work [81, 82, 83].

A similar idea has been used for tracking data flow within cloud infrastructures [84, 85, 86]. These works adopted the perspective of *Information Flow Control* (IFC), a concept that emphasizes more on policy enforcement on lower level data transfer, e.g., data sharing among applications within a cloud infrastructure. Particularly, the proposed IFC is a Linux kernel implementation, which indicates a data-centric paradigm for data management. Traditionally, user data is controlled by the processing application and invisible to the operating system, thus in an OS, when data is shared by multiple applications, each application has to implement its own data control mechanism, which leads to difficulties in consistent policy enforcement even though they are in the same system. The IFC, however, applies data control policies at the OS level, which enables system-wide consistent policy enforcement beyond isolation and application borders (within an IFC-enforcing world). This methodology indeed separates the responsibility of data control from applications. How an application accesses and processes the data is determined and monitored by the OS according to the policies associated with the data.

Another related technique is *Taint Tracking* (TT) [87]. An example of TT used for privacy preservation is to taint sensitive information, e.g., a list of contacts in mobile phones, and track it through this closed system [88]. Data leaving the system (i.e., the phone) is analyzed to ensure it does not contain sensitive information. The tainting mechanism is to propagate tags through any entities that may contain sensitive information. Since TT is an OS-level mechanism, it could not know exactly how the data would be used by a program or process. If the sensitive data is accessed by an instance of an application, the aroused processes will acquire the data's tag(s), which taint the processes as sensitive. The resulting output or outgoing connections may be tainted as well. By monitoring the tainted objects, the

occurrence of the issues (e.g., a data leakage, an attack) could be detected.

From the perspective of execution model, the *sticky policy* actually allows a data-centered approach that encloses allowed methods with the data object. This scheme could be further extended to define any allowed operation in the policy descriptor. Sundareswaran et al. proposed a logging system for data sharing in this paradigm [89]. In their work, the users' data is encapsulated with executable code in JAR files. There is a tradeoff between storage overhead and universal applicability of the *sticky policy*. For instance, a system for information flow control uses a tagging mechanism to identify the policies applied to specific user data within a PaaS cloud [90, 91] . Since the policy recognition and enforcement system is embedded in the cloud infrastructure, the tag associated with the user data is quite lightweight, but this policy enforcement could not be applied when data has to travel across the boundary of cloud infrastructures (e.g., from EC2 to Azure). If the enforcement code is attached to the data as [89] or [92], the policy application could be ensured as the data travel through different cloud infrastructures at the cost of overhead probably higher than the data in concern. Ideally, the most effective way to implement the *sticky policy* mechanism is by a protocol standard in which a header is defined as the policy descriptor, and the processing methods are defined for the policy agent. We take this paradigm in this work by assuming there is a standard policy descriptor attached to the user data, and the corresponding policy agent is deployed through the entities involved in the service provision.

4.3 System Design Overview

4.3.1 Sticky Policy Based Evidence Collection

Our *sticky policy* mechanism can be illustrated by Fig. 4.2, where part of the data flows in Fig. 4.1 is sketched. The user data is accompanied by a policy descriptor

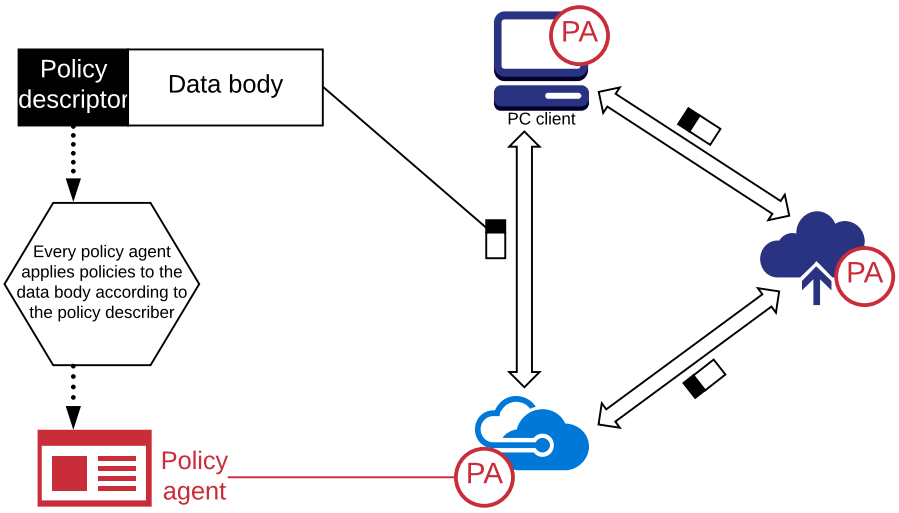

Figure 4.2: The sticky policy mechanism that enables policy application and evidence collection in Fig. 4.1

during its entire life cycle, including transmission, storage, duplication, processing, and deletion. The policy descriptor defines the applied policies, or allowed treatments. A policy agent parses the policy descriptor to determine the policies and configure the execution environment. We do not require that the policy descriptor should carry the execution module but assume that the mechanism for policy application is deployed within the policy agent at any cloud infrastructure involved in the service provision. In addition, the policy agent is required to have a communication mechanism that sends logging updates to the data owner and the service providers when the data is touched by any program. This log serves as the evidence feeding into the inference engines in Fig. 4.1.

Because of its inherent flexibility, the *sticky policy* enabled scheme is potential to be adjusted to the ever-growing set of policies. Here, we focus our scheme on following particular problems.

- **Consistent cross-boundary policy application**. Since any copy of the user data is accompanied by a policy descriptor, it is straight forward to maintain a consistent set of applied policies when the data is moved to

another *sticky policy* enabled domain.

- **Global view of data distribution**. A central problem for auditable policy compliance is to track all the duplication of the user data through the cloud in that the data is probably copied by various purposes (e.g., backup, buffering, process outsourcing, etc.) and some are generated in unexpected ways. Therefore, the violations might happen by not only malicious behaviors but also by misconfiguration or failure of exception processing. The *sticky policy* enabled logging mechanism could help to discover all the intentional and unintentional duplications.

- **Fair availability of information for policy compliance inference**. VOCMC works when all the participants can make informed decisions to reach an agreement. Thus, the fairness of information acquisition is a critical feature. Since the protocol allows lag for the participants to make delayed decisions, the fairness here is in the sense of a unique view of available information. The *sticky policy* mechanism should guarantee an independent "push" notification for all the participants.

Fig. 4.1 outlines the high-level design of our proposed public ledger of policy compliance. The logging mechanism independently provides evidence to policy compliance inference engines on both the user side and the provider side. According to the outputs of the inference engines, the user and the provider reach an agreement on compliance or violation. Finally, the agreement will appear on the blockchain (may not in the form of the original record).

4.3.2 Public Blockchain Versus Private Blockchain

The public ledger is recommended to adopt the public blockchain system. The reason is that the major benefit of blockchain is its immutability and no trusted

authority involved. This property depends on the assumption of the independence of the miners and the fact that no (group of) miner controls a dominant computing power. This assumption probably holds for public blockchain, but not for private (or permitted) blockchain.

The difficulty in adopting private blockchain as the ledger is that it essentially introduces a trusted authority. The private chain is owned by, an organization and the miners have to get permission from the owner. Ricardo N. et al. [93] pointed out that the only feasible solution for the blockchain-based database with public auditability is to utilize private BC for the recording, and public blockchain for the checkpoint. We hold a similar point of view, but realize that the private BC is a replaceable component, as long as the checkpoint in the public blockchain is reliable. In a setting where a blockchain is controlled by a trusted authority, the performance of the blockchain has to be compared with other distributed database systems.

Note that there are scenarios in which permissioned blockchain is applicable and probably a better solution than its permissionless counterpart. For instance, a blockchain-based multi-bank settlement system may adopt permissioned blockchain because if the participating banks play the role of miners, it is natural that the miners are independent. In such a system, the population of the miners is under control and will not decrease and increase frequently. Because of the non-anonymity, the behavior of the miners is identifiable and detectable.

An observation here is that the assumed attackers in the blockchain-based system are some malicious block miners. Therefore, if the ledger users are mainly the miners as well, it is unnecessary to adopt the permissionless blockchain in practice because they have a direct incentive to guarantee the reliability of the data and detect any misbehaviors of other miners. For the scenario of this dissertation, we assume the public is the main data user of the ledger; thus, the interest of the

miners may be inconsistent with the ledger user. Therefore, we choose the public blockchain for the ledger.

4.4 Demonstration Cases

This section provides four implementations of policy cases to demonstrate the sticky policy mechanism as follows,

1. data duplication discovery,
2. guaranteed data deletion,
3. data sharing black and white list, and
4. consent grant and withdraw.

The first two are related to the problem of creating a global view of cross-boundary data distribution, and the rest are related to the problem of updating policy across the global data distribution. The sticky policy mechanism is flexible, because it resembles a container of modules, thus open to extensions. We do not include access control into the demonstration set in that we attempt to show the power of this mechanism for cross-boundary policy compliance. Cross-boundary access control relies on the services in our showcase and could be implemented by extension.

4.4.1 Shared Communication Protocol and Execution Model

In order to achieve cross-boundary policy compliance, the sticky policy framework attaches policy descriptors to each applicable file, which makes it difficult to update the policy descriptors for all the copies of the applicable files at the same time. Imagine data copies in a single cloud infrastructure. Data might be duplicated due

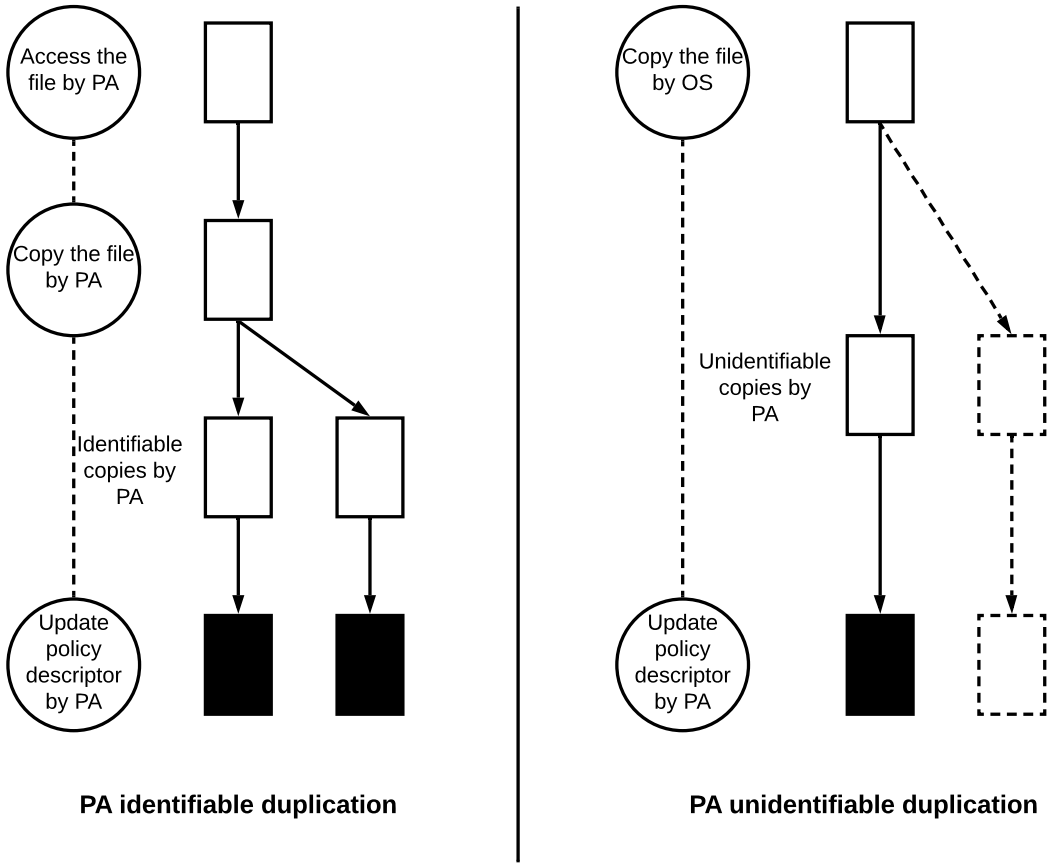

Figure 4.3: If the policy descriptor is updated whenever a policy update request arrives, some of the duplication can not be discovered by this process. There are unidentifiable duplications caused by lower level system operations without the participation of the PA.

to caching, keeping backups, or load balancing. The system is usually unaware of such "unintentional" duplications, e.g., hard drives might be copied at the operating system level, and thus there is no opportunity to execute the policy agent at all. Unintentional copies are generally unidentifiable at the copy time. Fig. 4.3 illustrates this problem.

Lowering the implementation level of the policy agent could mitigate but not eliminate the unidentifiable duplication, because virtual machines are broadly applied in present cloud infrastructures. It is impossible to apply fine granularity data management in such application scenarios, e.g., servers in an IaaS cloud.

Suppose we implement the policy agent with a list of controlled files. It is possible to add corresponding entries for copies generated by reaching the policy agent, which is the PA identifiable duplication in Fig. 4.3. With the presence of the controlled file list, policy updates could be sent directly to every file in the list. On the other hand, we could not add the PA unidentifiable duplication into the list of controlled files, and thus they are not reachable when users update policies.

Another consideration is the performance implication of the policy updating process. If the policy agent updates the policy descriptors of the controlled files once it receive a request for policy updating from the users, it will generate frequent random read/write requests for the file system, which will significantly degrade the performance of the entire system. One reason is that the files may not be actually accessed by the processes, and the policy updates will cause additional accesses. Another reason is that the user data and their duplications may not be stored consecutively in the storage medium.

Alternatively, we consider fetching policy updates at the access time. First, any user request for policy updates will be applied to a policy database. Second, whenever the user data is accessed at the policy agent level, the policy agent will pull the newest policy set from the policy database, and update the policy descriptor of the accessed files, as illustrated in Fig. 4.4. Note that the policy database in Fig. 4.4 is logically resident in the user domain in order to offer a cross-boundary policy application. The cloud provider could cache part of this policy database into the local domain of the cloud infrastructure to boost performance.

Another critical problem caused by unintentional copies is duplication discovery. As mentioned previously, the unintentional copies are unidentifiable by the policy agent at the copy time and thus not included in the global view of data distribution. This problem is directly related to the guaranteed deletion service, because the system could not delete a file without the awareness of its existence.

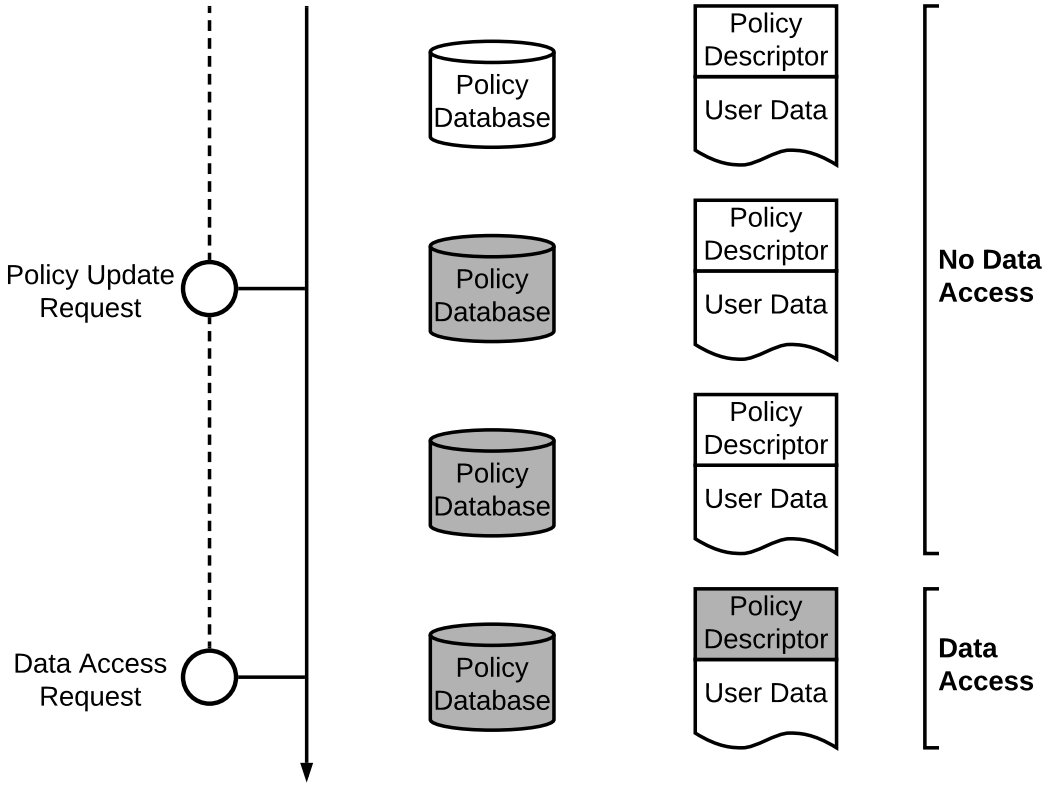

Figure 4.4: The policy update strategy: the policy database will be updated when policy update requests arrive, but if there is no data access request, the policy descriptor attached to that data file will not be updated. The policy descriptor will only be updated at access time.

Similar to the solution of the policy update problem, we expect to discover duplication only at the access time. The underlying logic is that if a file is never accessed, it is equivalent to a non-existent file. Hence, we need the policy agent to send a message to the user domain with information that is able to distinguish every duplication. We will provide a detailed discussion in a later section. The point here is that we could combine the message for duplication discovery and policy updates in a single round communication, and we aim to limit the message communication to a single round protocol in order to minimize its performance impact. Algorithms 18 and 19 describe the entities' behavior in this communication and execution model.

We introduced an entity data owner agent or user agent UA, which is a

Algorithm 18: Pseudo code of a policy agent PA

```
1 receive request(id_owner, id_PA, pid, fid, ACCESS)
2     send request(id_owner, pid, fid, STATUS) to user agent UA_owner
3     wait response(id_owner, pid, fid, UPDATE) from user agent UA_owner
4     upon receive response(id_owner, pid, fid, UPDATE)
5         update the policy descriptor of (id_owner, fid) with UPDATE
6         send response (id_owner, pid, fid, ALLOWED_OPERATION)
7             to the process p_pid
```

Algorithm 19: Pseudo code of a data owner agent UA

```
1 receive request(id_owner, id_PA, pid, fid, STATUS)
2     update(fid,STATUS) to file list
3     read(fid,UPDATE) from policy database
4     send response(id_owner, pid, fid, UPDATE) to policy agent UA_{id_PA}
```

container of the policy database and the global view of data distribution. Especially, the $STATUS$ contains the necessary information to recognize if a previously unknown data copy is the target file.

A data file is the minimum object that a policy set is applied. All the copies of the data file share the same policy set. We use `fid` to identify a group of files, which are copies of the same file, and share the same applicable policies. Additional information provided by $STATUS$ distinguishes the copies in a filegroup.

The `pid` indicates the entity that requests the data access. Be aware that `pid` here is a generic identifier that is better to be considered as an information container with a globally unique identifier. It may contain information that determines the policies to apply. In practice, this field may include domain identifier, user identifier, application identifier, system information, etc. The `pid` is used here for concision and capture the essence.

4.4.2 Case 1: Data Duplication Discovery

It is critical to discover any unintentional copy in order to create a global view of data distribution. As described in Section 4.4.1, the system discovers duplication at the accessing time via the information carried by the $STATUS$ field in the request. In the meantime, intentional and unintentional duplication should be treated in a uniform way. It is clear that if the policy agent does not report copy operation to the user agent at the copy time, there will be no difference between intentional copy and unintentional copy. Therefore, it is unnecessary to differentiate these two types of duplication and specify corresponding system behaviors.

The data duplication discovery subsystem takes into account the following design points:

- **A copy carries exactly the same information as its source.** At the copy time, both the data file and its policy descriptor will be duplicated. As a consequence, from the view of the user agent, any duplication is indistinguishable from its source, and the system could not rely on the information of the file's initial policy descriptor, such as the `fid` field.

- **This process should be a single-round protocol.** This requirement is to minimize the communication overhead introduced by the mechanism. Also, the system should minimize the size of the communicated message.

- **Messages could carry invalid information.** The system should filter out invalid duplication information. For instance, an attacker could inject fortified copy information into the data distribution view without actually possessing the data file. When the user requests data deletion according to the distribution view, it will trigger a policy violation.

Algorithm 20: Pseudo code of user agent UA for duplication discovery

init: $L \leftarrow \emptyset$
function: `fileupload`($file$)
    $HASH_{\texttt{fid}} \leftarrow \texttt{hash}(file)$
    $G_{\texttt{fid}}.\texttt{add}(HASH_{\texttt{fid}})$
    $L.\texttt{add}(\texttt{fid}, G_{\texttt{fid}})$
    `send` $(id_{owner}, file, \texttt{fid}, HASH_{\texttt{fid}})$ to policy agent $PA_{id_{PA}}$
receive request$(id_{owner}, id_{PA}, \texttt{pid}, \texttt{fid}, STATUS)$
    if $L.\texttt{search}(fid).G.\texttt{search}(STATUS.prehash)$ **then**
        if `verify`$(file, STATUS.timestamp,$
                $STATUS.rnd, STATUS.curhash)$ **then**
            $L[fid].G[STATUS.prehash].HASH \leftarrow STATUS.curhash$
        else return *INVALID*
    else $L[fid].G.\texttt{add}(STATUS.curhash)$
    `send` response$(id_{owner}, \texttt{pid}, \texttt{fid}, ACC)$ to policy agent $PA_{id_{PA}}$

The resulted process is described in Algorithms 20 and 21, when the user agent initializes a list of controlled files. When uploading files to cloud providers by invoking `fileupload`(), the function will generate an initial hash and create a file group for the uploaded file. The hash is used as the identifier for the initial copy in the filegroup.

Since the policy descriptor will be copied at the copy time, the system has to create a copy identifier at the access time (see Algorithm 21). More specifically, the policy agent will hash the data file with the current timestamp and an additional random number. The random number is introduced corresponding to the situation in which multiple copies get accessed at the same time. The timestamp, random number, newly generated hash value, and previous hash value will be included in the $STATUS$ field of the request sent to the user agent.

Upon receiving a request, the user agent will search the controlled file list L by `fid`. If an entry is found, then search the corresponding copy group $G_{\texttt{fid}}$ using $STATUS.prehash$ (at this point, the user agent does not know the newly generated hash). If the corresponding entry is found for $STATUS.prehash$, update that entry with $STATUS.curhash$, which is the treatment for a previously known copy. Before

Algorithm 21: Pseudo code of policy agent PA for duplication discovery

```
1  receive request(id_owner, id_PA, pid, fid, HASH_fid, ACCESS)
2      STATUS.prehash ← HASH_fid
3      STATUS.rnd ← random()
4      STATUS.timestamp ← now()
5      STATUS.curhash ← hash(file, STATUS.rnd, STATUS.timestamp)
6      send request(id_owner, pid, fid, STATUS) to user agent UA_owner
7      wait response(id_owner, pid, fid, ACC, UPDATE) from user agent UA_owner
8      upon receive response(id_owner, pid, fid, ACC, UPDATE)
9          update the policy descriptor of (id_owner, fid) with UPDATE
10         HASH_fid ← STATUS.curhash
11         send response
                   (id_owner, pid, fid, HASH_fid, ALLOWED_OPERATION)
                   to the process p_pid
```

the update, the $STATUS.curhash$ should be verified to guarantee its validity. According to the way of hash generation, this process could filter out the fortified request without actually holding a copy of the file.

If there is no entry in the copy group $G_{\texttt{fid}}$ for the $STATUS.prehash$ (this will happen when a copy is accessed after its source file was accessed, vice versa), it indicates a previously unknown duplication has been found. Therefore, the system will add a new entry to the filegroup $G_{\texttt{fid}}$ with $STATUS.curhash$. Note that, at the copy time, the source file and its copy are equivalent, and the system has to update the file's hash at every access time to differentiate each copy. Essentially, each duplication is identified by its access pattern.

The policy agent PA could only update the $HASH_{\texttt{fid}}$ for a particular copy upon receiving the ACC response in order to avoid inconsistency between policy agent and user agent. A subtle point is when the ACC response is missing. It may indicate two situations:

- the request is not received by the user agent, and
- the ACC response is not received by the policy agent.

To settle this problem, we could set a timer on the UA side when sending the

ACC response. If the same request is received before the timer is expired, the *UA* will just resend the response. The *PA* will also set a timer when sending the request. If the *ACC* response is not received before the timeout, the *PA* has to resend the request. This mechanism is not included in the algorithm description for concision.

4.4.3 Case 2: Guaranteed Data Deletion

Guaranteed data deletion could also refer to secure data deletion [94], data being forgotten [95], sanitized [96, 97], etc. Most of the works focus on physically erasing data from the storage media. In their setting, data can be considered as securely deleted from a storage system, if an adversary with access to the system is not able to recover the deleted data [94, 98, 99]. Some researchers further require that data is assuredly deleted when it becomes inaccessible to anyone permanently after it has been deleted [100, 101, 102, 103].

Incomplete data deletion is common because typical file deletion only updates metadata of the "deleted" file (e.g., mark the related storage space as allocatable), while usually leaving the media unchanged until that space has been overwritten by new files. For instance, research showed that MSDOS disc formatting operation only overwrites 0.1% of the data [104]. In the cloud environment, this issue becomes an even more complex problem due to the unintentional duplication. Particularly, at the time when the data owner sends a request for data deletion, it is possible that some unintentional copies are unidentifiable. If the data has been outsourced to other cloud entities, guaranteed deletion requires WLAN wide search for the data file, which is impractical.

We did not pursue the enforcement of physical erasion of the data upon request. Supported by the duplication discovery (see Section 4.4.2), we take a surveillant approach. Remember that a copy is considered as non-existent, if it has never been accessed. Therefore, immediate deletion is not required in this system.

Algorithm 22: Pseudo code of user agent UA for guaranteed deletion

```
1  init: L ← ∅
2  function: filedelete(fid)
3      L[fid].DEL_MARK ← DEL_PENDING
4      send request(id_owner, file, fid, DEL) to policy agent PA_{id_PA}
5  receive request(id_owner, id_PA, pid, fid, STATUS)
6      if L.search(fid).G.search(STATUS.prehash) then
7          if verify(file, STATUS.timestamp,
                     STATUS.rnd, STATUS.curhash) then
8              L[fid].G[STATUS.prehash].HASH ← STATUS.curhash
9              if L[fid].DEL_MARK == DEL_PENDING then
10                 if L[fid].G[STATUS.curhash].DEL == false
11                     L[fid].G[STATUS.curhash].DEL ← true
12                 else return POLICY_VIOLATION
13         else return INVALID
14     else
15         L[fid].G.add(STATUS.curhash)
16         L[fid].G[STATUS.curhash].DEL ← true
17     send response(id_owner, pid, fid, DEL) to policy agent PA_{id_PA}
```

Instead, our goal is auditable and transparent data deletion, i.e., the system could monitor the data deletion operation and discover undeleted copies in the future. Physical deletion is irrelevant in this system, because any data access is reported to the user agent by the policy agent. If supposedly already "deleted" data gets accessed, the data owner will catch this event, which will trigger a policy violation.

Algorithms 22 and 23 describe the logic of this component. Similar to duplication discovery, the deletion takes place at the access time in effect. The method `filedelete`(fid) only marks the copy group $G_{\texttt{fid}}$ as $DEL\_PENDING$. When a request arrives for accessing copies in a $DEL\_PENDING$ group, the user agent checks whether this is the first access request towards this copy after the `filedelete` has been called. If it is the first time, then set its deletion mark as `true`; otherwise, issue a $POLICY\_VIOLATION$ event.

Since it is possible that unidentifiable copies exist after a file is "deleted", the system should be open to newfound copies. Whenever a new copy is recognized, add

Algorithm 23: Pseudo code of user agent UA for guaranteed deletion

```
1  receive request(id_owner, id_PA, pid, fid, HASH_fid, ACCESS)
2      STATUS.prehash ← HASH_fid
3      STATUS.rnd ← random()
4      STATUS.timestamp ← now()
5      STATUS.curhash ← hash(file, STATUS.rnd, STATUS.timestamp)
6      send request(id_owner, pid, fid, STATUS) to user agent UA_owner
7      wait response(id_owner, pid, fid, ACC, UPDATE) from user agent UA_owner
8      upon receive response(id_owner, pid, fid, DEL)
9          update the policy descriptor of (id_owner, fid) with DEL
10         HASH_fid ← STATUS.curhash
11         send response (id_owner, pid, fid, HASH_fid, DEL) to the process p_pid
```

it to the group and mark it as deleted. For all these situations, the user agent should uniformly respond to the access request with DEL to notice the deletion to the policy agent.

Upon receiving a DEL response, the policy agent has to send the request process a DEL response. We have not defined the behavior of request processes upon receiving DEL responses, but since any access has to take place via the policy agent, access to "deleted" files will be detected and trigger policy violations.

Though this model does not actively delete files upon request and delay the deletion to the next access request, it will not conflict with the common usability practice. The user interface could invoke access requests right after the `filedelete`() method has been called. This treatment will delete known duplications immediately.

4.4.4 Case 3: Black/White List for Data Sharing

Black/White list is the showcase of the customizable policy set of the system. We have to first differentiate two types of data sharing because of the discrepancy between access patterns:

- A local process p draws and sends part of the data to the request process.

This access pattern could occur when the user data can be viewed as a list of entries, and only a few entries will be used at a time, e.g., machine learning algorithms.

- Send a copy of the target file to the request process. This access pattern usually occurs when sharing multimedia data, or for load balancing purposes.

The first type could be implemented inside the policy agent by a built-in attribute-based access control mechanism. PA returns required data entries according to the applied polices. Since the request process could not acquire full accessibility to the data, it is not file-level data sharing. This type of data access could be considered as a procedure in which a process with file-level accessibility serves as an access agent providing data service to other processes.

The focus of this section is the second type, file-level data sharing, which is usually related to cross-boundary data sharing. The policy agent has to send the entire file, including the policy descriptor, to the request process. In essence, file-level data sharing will create a new copy for the domain who initializes the sharing request, so the overall method heavily depends on the duplication discovery mechanism. Note that the policy descriptor could only regulate the behaviors assisted by the policy agent; thus, the duplication discovery is critical for cases in which the function of policy agent is bypassed in some way.

The black/white list is not enforced by the system, however, the system will detect policy violations. Data shared with a third party domain has to be accessed through the policy agent resident in that domain. Thus the user agent could detect any access request from domains that are blacklisted by the policy configuration.

The behavior of the user agent for data sharing follows the general treatment described in Algorithm 19 (see section 4.4.1), and the policy agent is defined in Algorithm 24. Notably, the user agent will not react to the data sharing request

Algorithm 24: Pseudo code of a policy agent PA for data share

```
1 receive request(id_owner, id_PA, id_domain, pid, fid, SHARE)
2     send request(id_owner, pid, fid, STATUS) to user agent UA_owner
3     wait response(id_owner, pid, fid, UPDATE) from user agent UA_owner
4     upon receive response(id_owner, pid, fid, UPDATE)
5         update the policy descriptor of (id_owner, fid) with UPDATE
6             if isblacklisted(id_domain) then
7                 send response (id_owner, id_domain, pid, fid, REJECTED) to
                      p_pid
8             else send response (id_owner, id_domain, pid, fid, ALLOWED) to
                      p_pid
```

differently from other requests, though data sharing will lead to additional duplications.

In fact, the user agent does not distinguish the type of requests except for the request for consent (see section 4.4.5). The black/white list is only part of the policy set and will be returned to the policy agent though the policy update $UPDATE$. Therefore, the user agent will not create any representative for the possible new duplications at this moment; however, if the request regards some unknown duplications that have already existed, the UA will treat it through the duplication discovery service.

4.4.5 Case 4: Consent Grant and Withdraw

User consent for data usage is an important feature for data service not only because of the increasing awareness of privacy protection of the public, but also regulations are established to enforce the transparency of data usage and allow the data owners to decide how their data will be used.

The grant and withdrawal of consent are different from other policy requests in that it is a process initiated by the service provider to require additional privilege from the data owner. In previous cases, the policy agent could only passively check the policy database for allowed operations or data sharing domain. The consent

Algorithm 25: Pseudo code of a policy agent PA for consent grant

receive request(id_{owner}, id_{PA}, `pid`, `fid`, $CONSENT$)
    `send` request(id_{owner}, id_{PA}, `pid`, `fid`, $STATUS, CONSENT$) to UA_{owner}
receive response(id_{owner}, id_{PA}, `pid`, `fid`, $CONSENT$)
    `notify` the process p_{pid} with (id_{owner}, `fid`, $CONSENT$)

Algorithm 26: Pseudo code of a user agent PA for consent grant

receive request(id_{owner}, id_{PA}, `pid`, `fid`, $STATUS, CONSENT$)
    `update`(`fid`, $STATUS$) to file list
    `consent` $\leftarrow$ *user_input*
    `update`(`consent`) to policy database
    `send` response(id_{owner}, id_{PA}, `pid`, `fid`, $CONSENT$) to $PA_{id_{PA}}$

mechanism allows the policy agent to purpose more operations intended to apply but not yet allowed by the current policy set.

The consent withdraw could be implemented by mechanisms we have already defined; thus, we have not defined a specific procedure for consent withdrawal. It can be done in two ways:

- **Reduce the allowed operations though the policy updates**. This method is recommended for fine granularity consent control. Since the user agent could update the policy database anytime without interaction with the cloud provider, consent withdrawal is under control by the data owner.
- **Delete data from the cloud infrastructure.** This method removes data from the cloud storage space and blocks any operation request from the process in the cloud.

For consent grant, a specific request and corresponding behaviors are defined as Algorithm 25 and Algorithm 26.

Because the request for consent grant needs human interference, the policy agent will not wait on response right after the request is sent, but asynchronously process the response. We use `notify`() to differentiate this process from the ones

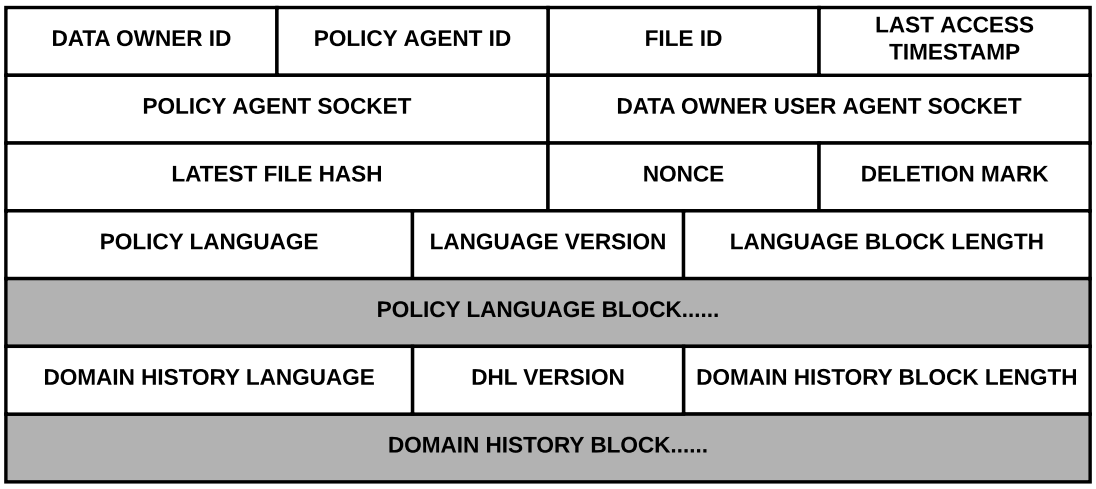

Figure 4.5: Data structure of the policy descriptor prototype. Except for the policy language block and the domain history block, every field is with fixed length. The policy language block is a variable length field whose length is determined by the language block length field. the domain history block is a variable length field whose length is determined by the domain history block length field.

that wait on responses. On the side of the user agent, standard duplication discovery will be conducted, which is represented by `update`(`fid`, $STATUS$) in Algorithm 26. Then, prompt the data owner for input and update the policy database according to *user_input*. Therefore, the cloud provider has to send requests for consent grant and check the policy database later on. We can consider any policy update is a process of implicit consent grant or withdraw.

How the policy agent handles the response is not essential for the explicit consent request and left to the request process the responsibility.

4.5 Policy Descriptor Prototype

Based on the policy demonstrations cases, we construct the policy descriptor prototype whose fields are described below (illustrated in Fig. 4.5).

- **Data owner ID**. Possible usage of this field is to assist management on the cloud side. On the other hand, it helps user agents filter out messages that

received from the socket but aims to other data owners. This field is recommended to be globally unique, e.g., addresses on a public blockchain.

- **Policy agent ID**. This field is used to distinguish policy agents, especially in cases that data is shared by multiple data processors in a single domain.
- **File ID**. This field is the identifier of a group of duplications belongs to a data file, i.e., copies with the same file ID contain the same data except for the policy descriptor.
- **Latest file hash**. The hash value generated by taking as input the combination of the data file, the timestamp of the last access, and a random number. This field serves as the identifier to distinguish copies in a filegroup. This field will be updated every time it has been accessed, thus this value has to be synchronized on both sides.
- **Last access timestamp**. This field is used to record the timestamp of the last access in order to help verify the file hash. This field has to be updated whenever the file hash has been updated.
- **Nonce**. This nonce is the random number that is used to generate the file hash. The purpose of introducing a random number in the hash generation is to avoid hash conflict when multiple copies of a single file have been accessed at the same time.
- **Policy agent socket**. This field is used for the user agent to create a communication connection with the policy agent.
- **Data owner user agent socket**. This field is used for the policy agent to create a communication connection with the user agent.

- **Deletion mark**. This field is used to indicate a copy has to be deleted, which has two main effects: as the instruction for deletion of the data file on the policy agent side; as an indicator of policy violation if an access request is received for a copy marked as deleted.

- **Policy language, language version, language length, and policy language block**. The proposed system allows flexibility for policy representation. Since the policy set is a variable-length list of variable length content, we do not specify a fixed length for each entry. Instead, the user of this system could specify a language parser by the policy language and version field. The language length indicates the offset of the following policy block in bytes.

- **Domain history language, DHL version, domain history block length, and domain history block**. These fields are used to track the history of data sharing across domains, which is useful, especially for identifying the origin of a duplication of a data file. Since the request message is part of the evidence for inference of policy compliance, messages have to be forwarded to policy agents along the domain history path for fair information availability. Similar to the policy representation, the user of this system could specify a language parser by the domain history language and version field. The domain history block length indicates the offset of the following domain history block in bytes.

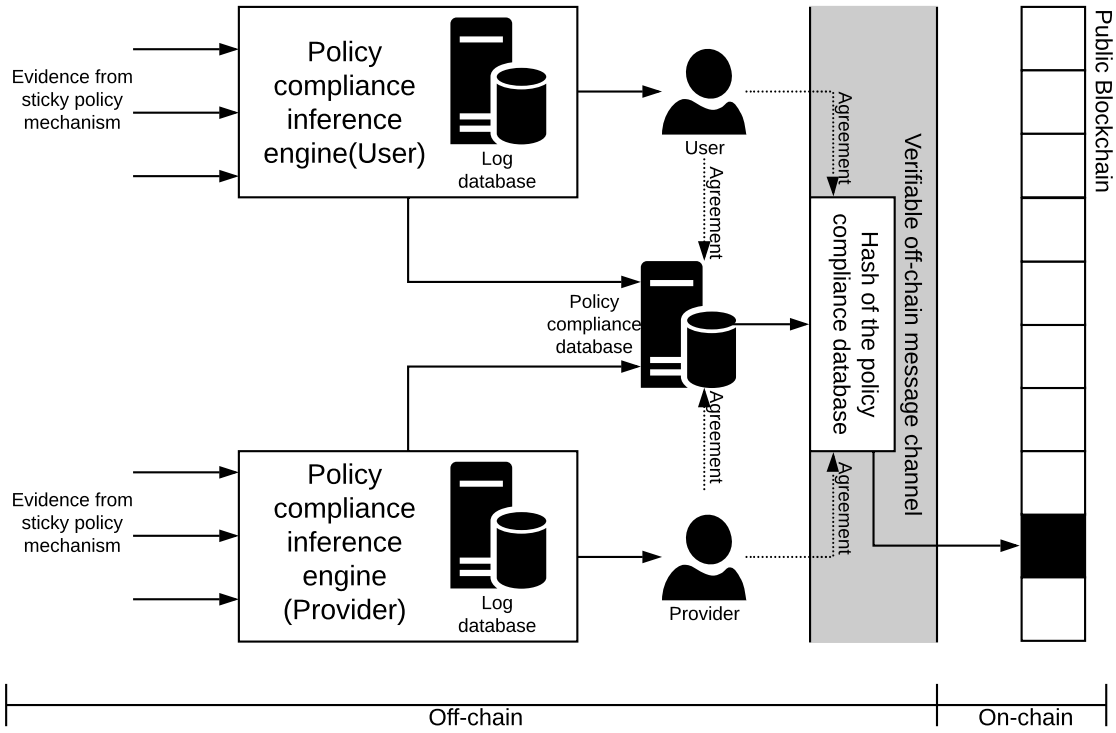

Figure 4.6: The illustration of the concept demonstration prototype

4.6 System Prototype

4.6.1 Off-chain Databases: Evidence and Inference

We have built a prototype system for the public ledger of policy compliance on top of the demonstration policy set. The policy violations under consideration regard the guaranteed deletion, black/white list, and consent grant and withdraw. Data duplication discovery serves as a critical supporting technique. The communication for policy requests between the user agent and the policy agent is recorded as the evidence to infer policy violations. Fig. 4.6 illustrates the structure of the prototype of the ledger of policy compliance.

The off-chain database is comprised of two levels and should be maintained on both the data owner side and the cloud provider side:

- **The evidence database**. It keeps the most detailed records regarding policy

compliance. Particularly, the prototype system records the request and response messages between the user agent and the policy agent. Note that the request and response mechanism could not generate exactly the same view on both sides. To understand, imagine the scenario of cross-boundary data sharing. From the data owner's point of view, there are requests from policy agents of multiple domains. However, each policy agent only holds its own requests and response. As a result, the policy agent could not detect some violations that could be detected by the user agent. For fairness of information availability, requests and responses should be forwarded to policy agents along the domain history path.

- **The policy compliance database**. It keeps the output of the policy compliance inference engine. The hash that will be submitted to the VOCMC for agreement is calculated from this database. Although the evidence database could create the same view on both sides, the two views are not the same bit-by-bit, which is not suitable for hash functions. The data owner and the cloud provider, in fact, reach an agreement on both the updates to the police compliance database and its renewed hash.

We use the on-chain hash as an immutable checkpoint of the police compliance database. Both sides confirm the content of the database and sign on-chain transactions through the VOCMC. When retrieving data from the compliance database, the ledger user could verify its integrity by comparing its on-chain checkpoint with calculated data hash.

4.6.2 VOCMC Configuration

Suppose the Ethereum blockchain or that with equivalent expressiveness of Turing completed language is the ledger for the on-chain transactions. The user and the

provider transfer deposits into a 2-of-2 multi-signature address to open a VOCMC for pairwise recording. The deposit could be part of the payment for the service provider, which would be transferred to the provider's address at the end of the service provision according to the incentive model. The incentive model is implemented by the smart contract. Every time an update to the database arrives, all the parties should sign a temporary transaction that contains the hash of the renewed view of the database when reaching an agreement on the updated states.

Regarding the finalization, our design choice is that the channel would be closed when finalizing a record, with the redistribution of the deposit. The reason for the decision is first that the on-chain cost is almost the same no matter what kind of transaction is processed, and secondly, that finalization could be caused by dispute where deposit must be redistributed. The current design processes the finalization uniformly. If the finalization is triggered by the timeout T_E and the service continues, the deposit will be transferred to another 2-of-2 address as opening another VOCMC. Figure 4.7 illustrates the ideal case in which all the participants are honest. The VOCMC output to the BC periodically to set checkpoints on-chain. During the finalization, the smart contract for the incentive model would be executed by the miners at the cost of the service provider[1].

The objective of the proposed public ledger is to serve as a credit record of service providers to provide a guide to the customer for informative decision making, just as the personal credit record helps banks make decisions on loan. A lesson learned from the current credit records is that mistake is not absolutely unacceptable, no correction is. The commercial bank would not report a default once it happens, but provide to the customer a time period to make a late payment or take other acceptable actions. Similarly, GDPR does not require the data

[1]This is not required. If the policy compliance database is maintained by a third party, the cost for on-chain computation could be paid by the third party as it joins the VOCMC by a 3-of-3 multi-signature address. Moreover, any operation cost will finally be transferred as part of the service charge to the consumer, thus this is not a essential requirement.

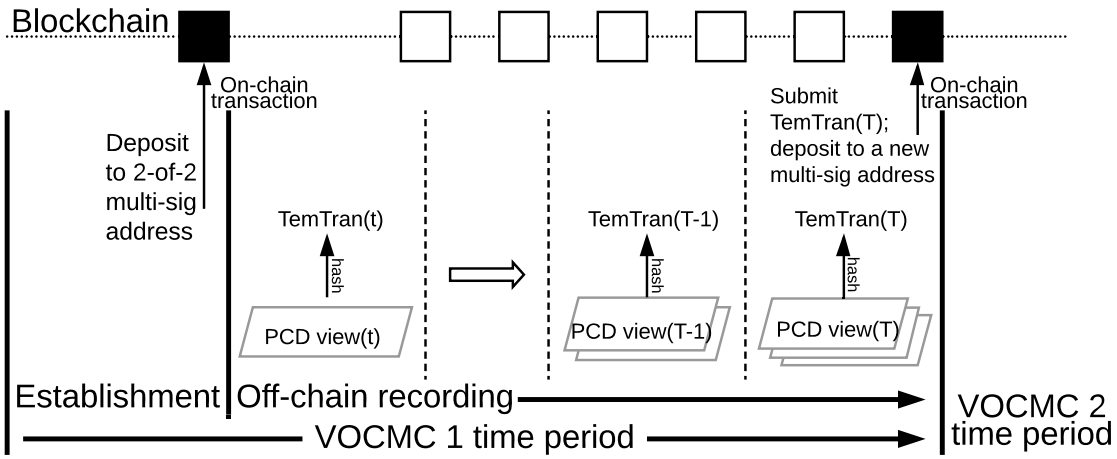

Figure 4.7: High level description of the public ledger with example of continuous service

processor to be absolutely secure. The organizations are allowed time to react to the event and notify the data leakage to the users in order to minimize the damages. The negotiation functionality supports this critical feature for a practical solution. Data or privacy leakage could take place during the process not only because of the intentional malicious behavior of the organization but also because of misconfiguration or attacking from outside. Besides providing an approach to address disagreement, the negotiation of the VOCMC could also serve as a buffering mechanism to allow the data processing organizations to take corrective actions or restoring their defensive system.

4.6.3 Anonymity

The BC community promotes anonymity as the default for any BC application in order to protect the privacy of the users. Because of public visibility, the users' detailed activities could be reconstructed, if identities are not properly treated. The basic approach for anonymity in a BC system is to strictly abandon address reuse, that is, an address can only be used for exactly one transaction, regardless of

representing a receiver or a sender. Note that anonymity provides on-chain privacy [20], that is, transactional privacy is provided against the public unless the contractual parties themselves voluntarily disclose information.

In our scheme, we embrace the anonymity of the data owner, i.e., the user of the data service. However, to enable auditability, we intentionally allow the reuse of service provider addresses. A service provider benefits from reusing an address for all activities related to a certain business since a trackable history record of good performance helps marketing. On the other hand, if a potential customer is given a new address with no publicly available historical record, the customer could suspect misbehavior.

Chapter 5
Evaluation and Conclusion

5.1 Experiment Setting

The system scales blockchain-based application by reducing unscalable on-chain operation without sacrificing the information reliability we desired from the blockchain system. Therefore, the purpose of the experiments is to demonstrate the capability of off-chain computation to replace on-chain computation. We ran the policy agent on a MacOS based machine, which is equipped with a 64-bit 3.1 GHz dual-core Intel Core i7-5557U processor, 16 GB 1867 MHz DDR3 RAM, 512 GB SSD, and 100 Mbit/s ethernet connection. The user agent is run on a MacOS based machine which has a 64-bit 2.7 GHz dual-core Intel Core i5 processor, 8 GB 1867 MHz DDR3 RAM, 256 GB SSD, and 100 Mbit/s ethernet connection. The use of CPU core is set to be the single-core mode for consistent measurement. The on-chain operations are tested on Ropsten testnet, whose behavior is supposed to be similar to the Ethereum main network.

5.2 Baseline Spacial Overhead

The cost of the entire system includes the overhead introduced by the sticky policy mechanism to data access and data process and the cost of the execution of the VOCMC protocol. In this section, we discuss the spacial overhead of sticky policy and VOCMC message.

5.2.1 Policy Descriptor

The prototype relies on the *Recursive Length Prefix* (RLP) [105] to create the headers and messages; thus, the length of each field of the header could be of

Header Field	**Length (bytes)**
Data Owner ID	8
Policy Agent ID	8
File ID	8
Last Access Timestamp	8
Policy Agent Socket	10
Data Owner User Agent Socket	10
Latest File Hash	32
Nonce	4
Deletion Mark	1
Policy Language	8
PL Version	8
Language Block Length	8
Policy Language Block	NA
Domain History Language	8
DHL Version	8
Domain History Block Length	8
Domain History Block	NA
Total	137+

Table 5.1: The length of the fields in policy descriptor of sticky policy

flexible length. For each field, the RLP will prefix two bytes to set the boundary and indicate the length of the field. For a payload of more than 55 bytes long, the RLP encoding adds 2+ bytes to indicate its length. Therefore, for the prototype, we construct the policy descriptor described in section 4.5 for the cost of 137+ bytes payload (see Table 5.1) and 34+ bytes for the cost of RLP. The uncertainty of the header length is because of the variable length fields, *Policy language block*, and *Domain history block*. The fields for IDs, timestamp, and length are of the long integer, which is of the size of 8 bytes. Note that big integer could be used to replace the long integer, if necessary. All the fields are determined to create a prototype with a reasonable setting in which the size of each field is by no means the only option.

The *Policy language block* and *Domain history block* are arbitrary strings.

Field	Length (bytes)
Requestor Address	20
Responser Address	20
Message Type	1
Digest of Policy Compliance/Negotiation	32
Round Number	8
Requestor Sign Mark	1
Requestor Signature	41
Responser Sign Mark	1
Responser Signature	41
Negotiation Block	NA
Total	165+

Table 5.2: The length of the fields in VOCMC message

We assume the policy block only contains a B/W list of domain names, and the B/W list consists of 10 items. As well, the domain history block is assumed to comprise 10 domain names. To estimate a realistic overhead of the policy descriptor, we pad these block with bytes of the average length of domain names, which is approximately 10 characters [106], and neglect the cost of the imaginary languages. The resulted policy descriptor is of length 377 bytes ($137 + 34 + 100 + 3 + 100 + 3$, the 3 bytes is for the RLP cost of string longer than 55 bytes). If the user data file is a jpeg image of size 2MB, the spacial overhead introduced by the policy descriptor approximate 0.019%. We use this pseudo-header for further performance experiments.

5.2.2 VOCMC Message

Table 5.2 illustrates the minimum construction of VOCMC message fields to implement the required behaviors described in chapter 3, but a particular message only carries a subset of these fields.

The possible message type with the largest size is the negotiation message,

which includes the addresses of requestor and responder, message type, digest of negotiation history, round number, and a variable-length field negotiation block. The total number of fixed bytes for a negotiation message is $81 + 10 + 3 = 94$ bytes, in which the extra 10 bytes are the RLP cost for regular length byte array, and the other 3 bytes are for byte array longer than 55 bytes. All the other message types are with a fixed message length.

The most frequently used message type is the messages for off-chain contract instance update, which consist of a pair of request and response. The request message comprises 140 bytes, including the addresses of requestor and responder, message type, digest of policy compliance history, round number, requestor sign mark, requestor signature, and responder sign mark. In addition to the fields of the request message, the response message has the responder signature field, thus results in 183 bytes in total.

The message for contract instance registration is an on-chain transaction. We only consider the payload of transaction data involved. A valid registration message contains all the fields other than the negotiation block and sign marks; thus, the total message length is 177 bytes no matter the contract registration is for policy compliance or negotiation.

5.3 Baseline On-chain Cost

On-chain cost consists of transaction fee and transaction confirmation time. We examine the on-chain cost in the Ropsten [107] network (one of the Ethereum test-nets). The Ropsten network executes exactly the same code of the Ethereum main net; therefore, we could expect quite similar behaviors compared to the main net execution.

Operation	**Cost (Gas)**
Functionality deployment	1455,250
Successful channel creation	123,047
Unsuccessful channel creation	157,378
Contract instance registration	237,420
Contract instance registration timeout	212,812
Successful channel closure	142,630
Unsuccessful channel closure	90,231

Table 5.3: The gas cost of on-chain operations

5.3.1 Transaction Fee of On-chain Operations

The transaction fee is measured by *Gas*, which is kind of the "fuel" to execute code in the Ethereum network. Every instructions and data storage has a Gas consumption value. The transaction fee is based on the total Gas consumption of the transaction. Gas is priced in Gwei (the most-used denomination of Ether, the cryptocurrency of Ethereum). The transaction fee equals
gas consumption $\times$ **gas price**. For execution, the gas consumption is fixed, but the user could announce a relatively high gas price to raise the priority of her transaction. Therefore, we only use gas consumption as an indicator of the operation cost. Table 5.3 lists the cost of complete execution of the on-chain operations of VOCMC.

To provide a more realistic sense of the on-chain execution cost, consider 1-Gwei gas price and $0.00000014 ether price. Contract instance registration, the most-used on-chain operation, is priced 0.0332 in US dollar. Thus, frequent on-chain transaction submission results in high execution cost, which is another constrain other than performance when building smart contract applications on the public blockchain.

By design, the on-chain operations consume a fixed number of gas because each operation only requires a single round protocol. If multiple rounds are

necessary (e.g., channel closure) for system security, the users should invoke the operations multiple times and manage it by off-chain logic instead of relying on the on-chain execution to handle the multiple-round protocol. The reason is that there is a maximum gas limit imposed by the block miners, non-deterministic execution risks out-of-gas exceptions.

5.3.2 Ropsten Transaction Latency

The latency of on-chain operation of VOCMC consists of three components, the transaction confirmation latency, computer-human interaction latency, and transaction communication latency. We used a message relay contract to measure the latency of protocol communication through the Ropsten network. The local client sends a registration transaction and its timestamp to the relay contract. Upon receiving the transaction, the contract sends the transaction data and timestamp to the responder address. The responder client uses the timestamp to measure the latency. We record an average latency of 83.3 ms.

We used the registration operation to measure the transaction confirmation latency of the Ropsten network. There are two reasons for this decision: 1) contract instance registration is the most-used operation, 2) contract instance registration requires no computer-human interaction. This measurement is used to estimate the scalability of the prototype system. The transaction confirmation latency is supposed to be 15s to 20s, which is adjusted dynamically by the mining difficulty [108].

The tester sent 1 transaction per minute for 1000 minutes, and query the transaction pool to monitor each transaction's appearance in the pool and take note of the time. Finally, query the Etherscan [109] to determine the timestamp of when the transaction eventually gets incorporated into a block. Fig. 5.1 shows how the transaction confirmation time varies over 100 minutes. In total, we recorded an

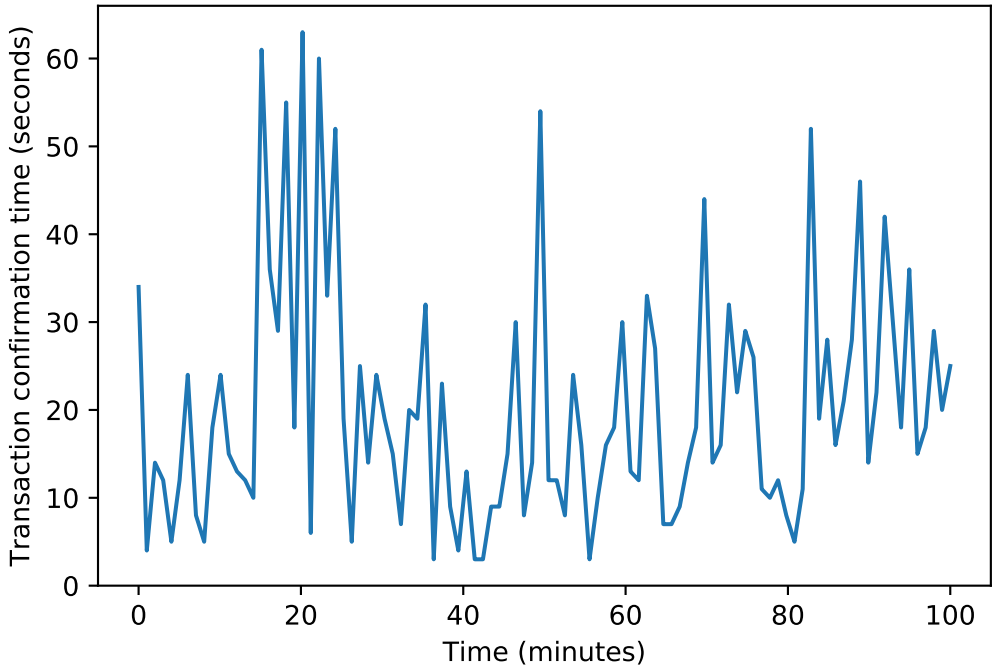

Figure 5.1: Ropsten testnet transaction confirmation time over a 100-minute period.

average confirmation time of 21.25 seconds and a standard variance of 13.89. Dynamically, the confirmation time varies from less than 10 seconds to around 70 seconds, as seen in Fig. 5.1. Based on this observation, a wide timeout is recommended for application logic that relies on transaction confirmation.

5.4 Data Access Delay

We measured the data access delay introduced by the sticky policy mechanism. Intuitively, there are two types of componential delay: 1) the communication delay between the policy agent and the user agent, and 2) the time to prepare the message and list checking. Given a pair of communication parties, most of the delays tend to be consistent, and fluctuation comes from the communication and execution environment. Only the time complexity of the hash function is $O(n)$ where n is a parameter related to the size of the data file, i.e., the execution time of

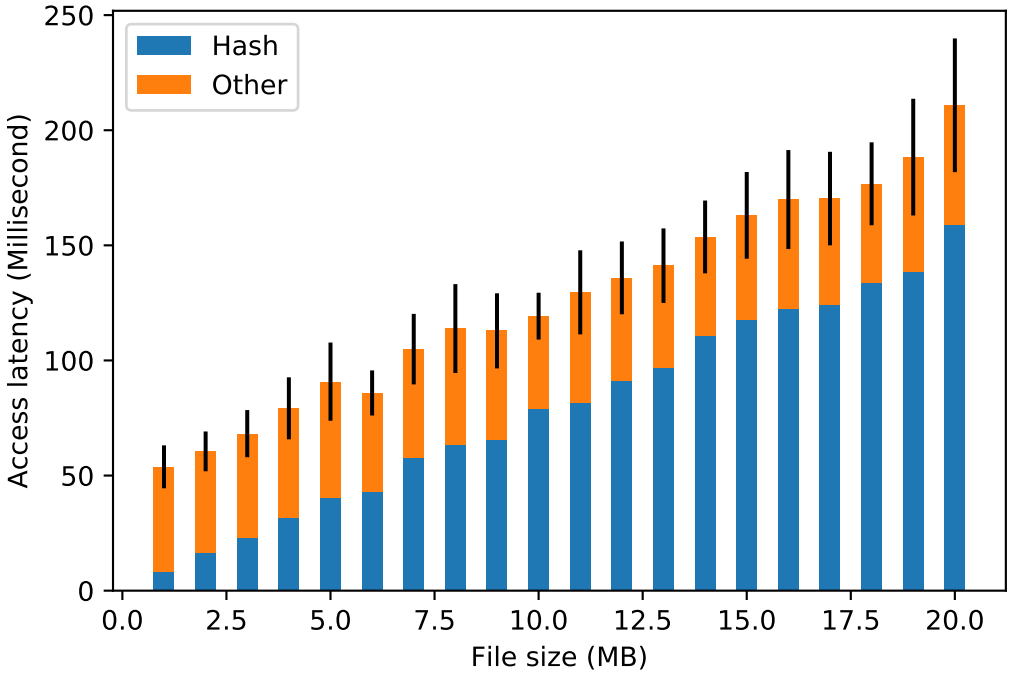

Figure 5.2: Data access latency introduced by sticky policy.

the hash function increases as the input file size increases. Therefore, besides measure the overall access delay, this experiment aimed to examine how user file size impacts the performance.

We traversed different file sizes from 1 KB to 20 MB. For each file with a given size, the tester accessed the file 1,000 times to record the average delay and the standard variance. Both the overall delay and the execution time of the hash function were recorded in order to get the statistics of the consistent component and the variable component. Fig. 5.2 shows the result of file sizes from 1 MB to 20 MB with 1 MB step length.

It can be observed from Fig. 5.2 that the execution time of the hash function starts to dominate the delay around file size of 7 MB. The consistent component is ranging from 40 to 53 milliseconds, and the variable part is linear to the file size as expected. The variance of the overall delay is around 9% of the total. The most

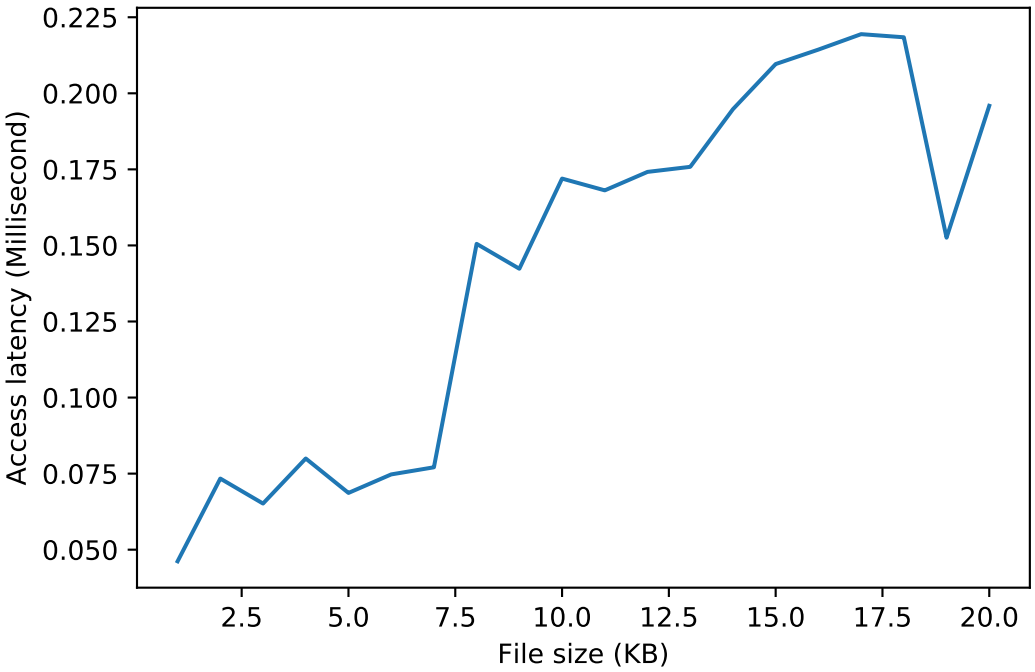

Figure 5.3: Data access latency introduced by sticky policy for small file.

acceptable range of delay is for multimedia files such as images or short video streams. It might need to be optimized to adjust to larger files. For the files with relatively small size (e.g., a text file of some KBs), the hash function overhead is negligible compared to the consistent component, as seen in Fig 5.3.

5.5 On-chain Strategy and Scalability

This section studies the VOCMC's impact on the scalability of blockchain-based general purpose ledger. Due to the lack of data access trace from real productive cloud infrastructure, we could only examine a simulated scenario in order to catch some insights into the properties of the prototype system. Table 5.4 lists the simulation configurations for the experiment.

The simulation is designed to capture system behavior similar to our application scenario, i.e., the public ledger for the policy compliance record of cloud

Simulation config	**Value**
Total policy compliance report	1,000
Gap between report generation	60-180 sec.
Size of policy compliance report	640 Bytes
Policy violation rate	5%
Pure on-chain recording	Upload per report
On-chain off-chain combination 1	Periodic
On-chain off-chain combination 2	Violation driven
On-chain off-chain combination 3	Combine 1 & 2

Table 5.4: Simulation Configuration of Scalability Test

Strategy	**Uploads**	**Time (s)**	**Gas cost**
Pure on-chain recording	1000	22,183	4.25×10^8
Periodic (1200 s)	100	1972	2.37×10^7
Violation driven	39	919	9.26×10^6
Periodic & violation driven	116	2466	2.75×10^7

Table 5.5: Simulation results of 4 different strategies

service providers. The policy compliance is assumed to be a non-frequent record. As an analog, a credit statement is reported on a monthly basis. We choose a 60-180 second gap and 1,000 records for the balance between low frequency and experiment efficiency. The 640 bytes is a reasonable choice for an informative policy compliance report. Notably, as the experiments show, these assumptions on original records are not essential for the performance of the off-chain boosted public ledger.

This experiment compares the on-chain-checkpoint-off-chain-database structure with pure on-chain data recording. Though the theoretical upper bound of transaction data is around 780 KB recently, it is not suitable for our scenario because it will take quite a long time to buffer 780 KB data of only the policy compliance report. However, it is simple to fill the block with evidence data. Hence, it is not necessary to reach the upper bound of uploaded data. For the pure on-chain recording, the tester uploaded every report once it arrived through transaction data.

We tested 3 strategies for the on-chain-off-chain combination in which only the hash value of the current accumulated records is uploaded through the contract instance registration of the VOCMC. The periodic strategy is useful for the most general-purpose database that set on-chain checkpoint over a specific period of time. Violation driven strategy is a special variety of event-driven strategies suitable for our application scenario. The policy violation is the most concerned records for the ledger user. This strategy guarantees that violations would be published upon their occurrence. We assume a violation rate of 5%, i.e., every policy compliance report has a 0.1 chance to be a violation report. The third strategy combines the periodic and violation driven strategies for both the merits, which is probably the optimized option for our scenario. Periodically uploading checkpoint is necessary for the daily operation of a database, and the violation driven offers on-time information exposure. Specifically, every time when a violation driven uploading is triggered, the current periodic cycle will be aborted and start a new cycle, i.e., the timer for periodical uploading will be reset to zero.

Table 5.5 shows the results, where *time* is the on-chain transaction confirmation time. The decrease of on-chain cost is not significantly remarkable due to the relatively small scale of our experiment (still decrease by orders of magnitude), and the parameter setting (small period). However, this scheme has more potential because of the decoupling of the on-chain operation and off-chain recording. Especially, the performance of the periodic strategy is completed irrelevant to the generation rate of the records. It only depends on the predefined time period. For example, if the uploading rate were set to be 1/24 hour, there would only be 2 uploads during the experiment (the run seen in Table 5.5 took a totally 119,110 seconds, 33 hours). On the other hand, if we had a 100× record generation rate, the cost of the periodic strategy would still stay the same, as seen in Fig. 5.4. Once the uploading rate is determined, the number of uploads would be

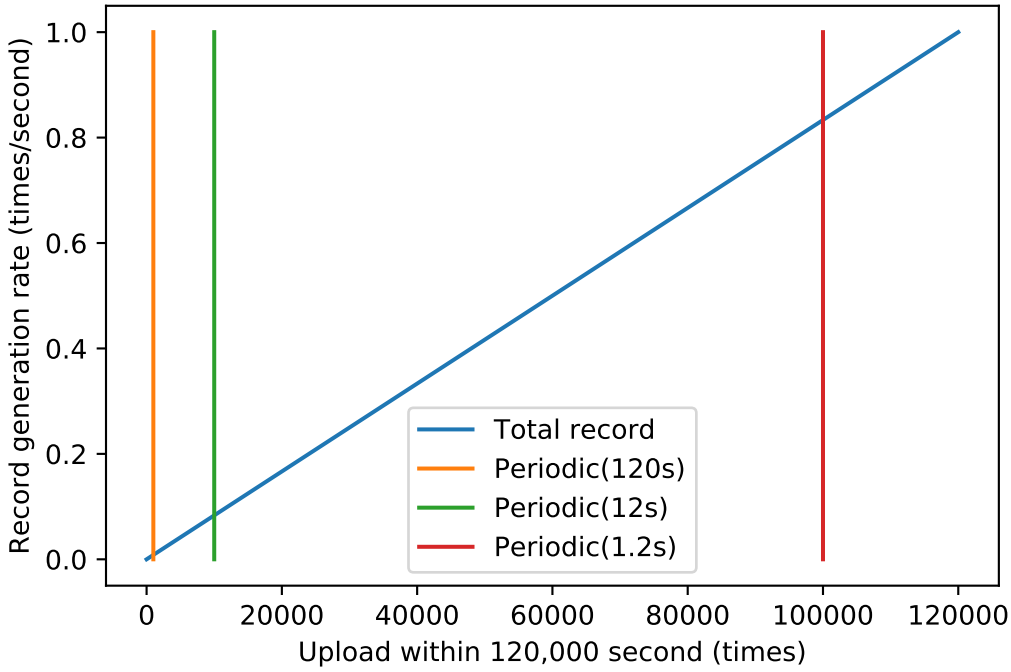

Figure 5.4: Periodic upload is not proportional to the record generation rate. Given a time period, the number of uploads remains constant, which is determined by an independent uploading rate.

a constant independent of the number of the records (but related to the uploading rate).

Though the violation driven strategy outperformed the periodic strategy in this particular case, the cost is related to the record generation rate. Therefore, scenarios with a high frequency of events in concern degrade this strategy. The cost of the combination of these two strategies was more than both the cost of the strategies, respectively. In fact, all the violation driven uploads were preserved and did replace part of the periodic uploads, so the combined cost should be greater than the cost of periodic strategy alone and less than the sum cost of these two strategies.

To conclude, the VOCMC effectively removes the scalability limit when developing blockchain-based general purpose ledger. The on-chain cost is decoupled from the scale of the records in concern. The developer could determine the portion

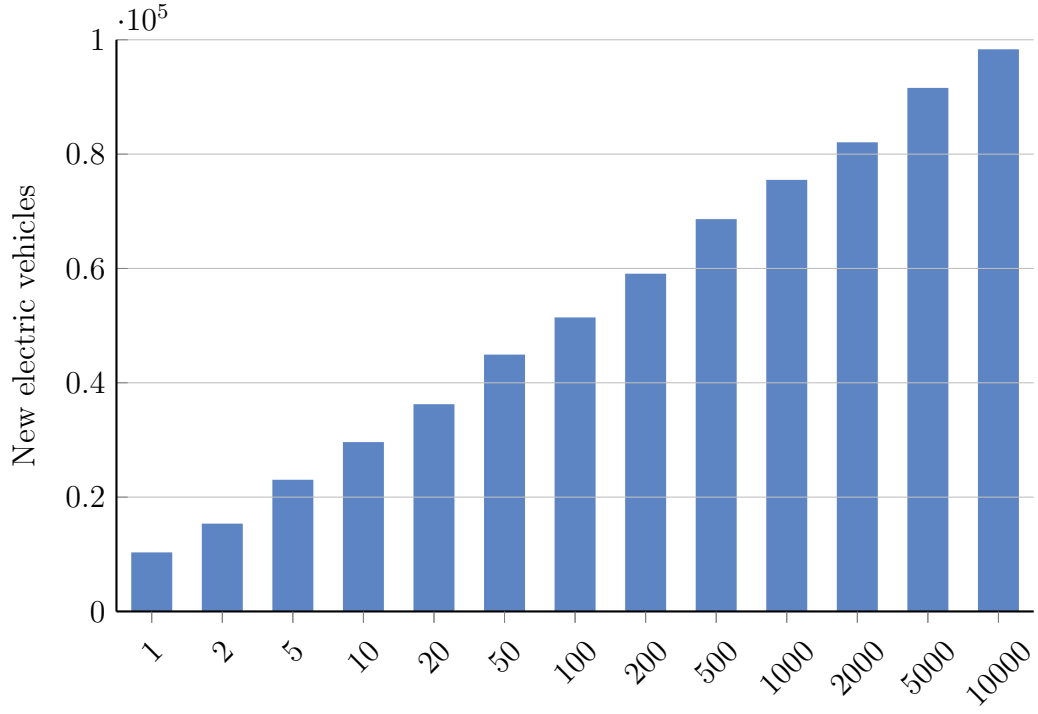

Figure 5.5: Number of accesses to completely deleted all the copies of file.

of on-chain operation purely according to the acceptable performance and finance constraints.

5.6 Guaranteed Deletion

We use this case study to examine the effectiveness of the policy application case supported by our prototype system because guaranteed deletion could also reflect the performance of the duplication discovery and the consent withdraw. Recall that a data file will be deleted at the first access after the user agent marks this file as deleted. As a result, the deletion depends on the access pattern. In this experiment, we only test the performance under a uniformly random access pattern. Since, in practice, files may be accessed at very low frequency, even never be accessed, we counted the number of access between when the deletion was issued and when the deletion was completed. We assume several copies of the file that would be deleted

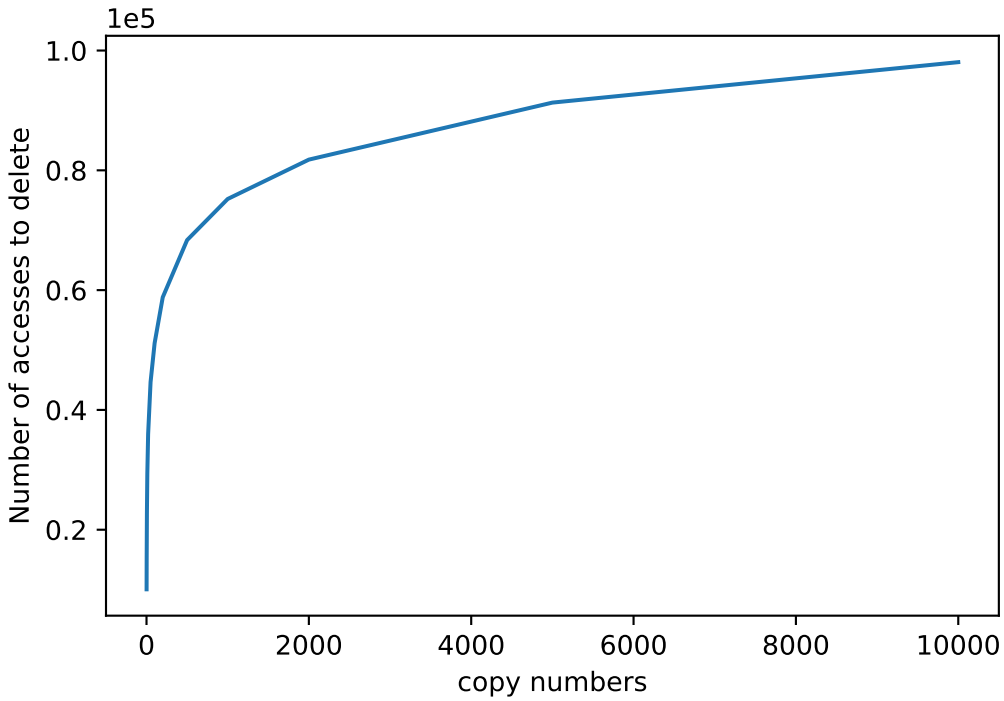

Figure 5.6: The access number required to complete deletion is nonlinear to the number of copies, growing rapidly for small copy numbers.

are spread in a file system with 10,000 files. We tested the guaranteed deletion on different numbers of copies, 13 cases from 1-10,000 copies. The total file number in the file system remained the same, i.e., 10,000 copies means 100% of the files in the system were copies of the file supposed to be deleted.

Fig. 5.5 shows the results. Notice the labels of the x-axis that the required number of accesses is not linear to the number of copies. Its functional relationship can be seen in Fig. 5.6. Under uniformly random access, even just 1 copy needed 10,052 access to hit the copy and complete the deletion. Though uniformly random access is far from the realistic access pattern in a file system, this is still an interesting observation, because the unknown duplications could only be touched by somehow random access if there is no regular scan of all the files in a system. The access number increased very fast when the copy number was very small, and thus for most of the realistic scenarios (usually, files would not be duplicated thousands

of times), the system should erase as many copies as possible through an initial deletion, i.e., access all the known copies right after the deletion instruction is issued. Decreasing from 5 copies to just 2 copies could significantly increase the chance of complete deletion. If a deadline is posted for data deletion, we recommend scanning the file system regularly to ensure unknown copies could be discovered before the deadline.

5.7 Conclusion

An innovative approach is proposed for constructing a public ledger of policy compliance in this dissertation. Particularly, VOCMC is introduced to address the verification of external information as a critical phase to build a public ledger based on blockchain technology. We stress that the blockchain is reliable in the sense of the immutability of on-chain transactions after finalization. However, the blockchain is lacking instruments to effectively verify external information; that is, it will trust whatever data provided as input. Therefore, when considering the combination of on-chain and off-chain architecture, the key is the verification of the information from the off-chain world. Moreover, the VOCMC displays the potential to further limit on-chain computation. A reliable verification mechanism for external information allows depending more on off-chain computations without impacting the reliability of the outcome.

Particular contributions of this dissertation are as follows,

- We formalized the ledger model and the verification mechanism (self-verifiable transaction system) of blockchain and proved blockchain's capability of internal information verification and its inability to verify information from the external world, based on our formalization.

- The VOCMC is proposed as the generalized and formalized model of the

off-chain channel for general purpose recording. Through the concept of incentive-based trust, we supported the justification of the information verification mechanism and analyzed the security properties of the VOCMC under the universally composable security framework.

- A prototype system was designed, in which on-chain hash is leveraged for the reliable checkpoint, and the VOCMC is used for information verification and scalability improvement. The sticky policy is employed as a general framework for policy enforcement and evidence tracking.

- We implemented the concept demonstration prototype with 4 policy application cases: 1) data duplication discovery, 2) guaranteed data deletion, 3) consent grant and withdrawal, and 4) data sharing black/white list.

- The prototype was evaluated on the Ropsten testnet for the Ethereum blockchain application. We explored some metrics which confirmed our expectation on performance and effectiveness. In particular, the decoupling of the on-chain and off-chain performance allows developers to scale blockchain-based applications through off-chain components without sacrificing the reliability of the information.

BIBLIOGRAPHY

[1] C. Breazeal, "Socially Intelligent Robots," *Interactions*, vol. 12, no. 2, pp. 19–22, Mar. 2005. [Online]. Available: http://doi.acm.org/10.1145/1052438.1052455

[2] S. Li, T. Zhang, J. Gao, and Y. Park, "A sticky policy framework for big data security," in *2015 IEEE First International Conference on Big Data Computing Service and Applications*, March 2015, pp. 130–137.

[3] K. Driscoll, B. Hall, H. Sivencrona, and P. Zumsteg, "Byzantine fault tolerance, from theory to reality," in *SAFECOMP*, 2003.

[4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[5] S. Haber and W. S. Stornetta, "How to time-stamp a digital document," in *Conference on the Theory and Application of Cryptography*. Springer, 1990, pp. 437–455.

[6] D. Bayer, S. Haber, and W. S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," in *Sequences Ii*. Springer, 1993, pp. 329–334.

[7] S. A. Haber and W. S. Stornetta Jr, "Method for secure time-stamping of digital documents," Aug. 4 1992, uS Patent 5,136,647.

[8] C. Lai, H. Li, R. Lu, and X. S. Shen, "Se-aka: A secure and efficient group authentication and key agreement protocol for lte networks," *Computer Networks*, vol. 57, no. 17, pp. 3492–3510, 2013.

[9] L. Harn, "Group authentication," *IEEE Transactions on computers*, vol. 62, no. 9, pp. 1893–1898, 2012.

[10] Q. Wu, Y. Mu, W. Susilo, B. Qin, and J. Domingo-Ferrer, "Asymmetric group key agreement," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2009, pp. 153–170.

[11] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014.

[12] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "Sok: Research perspectives and challenges for bitcoin and cryptocurrencies," in *2015 IEEE Symposium on Security and Privacy*, May 2015, pp. 104–121.

[13] R. Maull, P. Godsiff, C. Mulligan, A. Brown, and B. Kewell, "Distributed ledger technology: Applications and implications," *Strategic Change*, vol. 26, no. 5, pp. 481–489, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/jsc.2148

[14] J. Truby, "Decarbonizing bitcoin: Law and policy choices for reducing the energy consumption of blockchain technologies and digital currencies," *Energy Research and Social Science*, vol. 44, pp. 399 – 410, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214629618301750

[15] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, and et al., "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys ?18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3190508.3190538

[16] A. Singh, T. Das, P. Maniatis, P. Druschel, and T. Roscoe, "Bft protocols under fire." in *NSDI*, vol. 8, 2008, pp. 189–204.

[17] N. Narula, W. Vasquez, and M. Virza, "zkledger: Privacy-preserving auditing for distributed ledgers," *auditing*, vol. 17, no. 34, p. 42, 2018.

[18] M. Herlihy, "Blockchains and the future of distributed computing," in *Proceedings of the ACM Symposium on Principles of Distributed Computing*, 2017, pp. 155–155.

[19] A. F. Anta, K. Konwar, C. Georgiou, and N. Nicolaou, "Formalizing and implementing distributed ledger objects," *ACM SIGACT News*, vol. 49, no. 2, pp. 58–76, 2018.

[20] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 839–858.

[21] X. Défago, A. Schiper, and P. Urbán, "Total order broadcast and multicast algorithms: Taxonomy and survey," *ACM Computing Surveys (CSUR)*, vol. 36, no. 4, pp. 372–421, 2004.

[22] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of the ACM (JACM)*, vol. 43, no. 2, pp. 225–267, 1996.

[23] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, and R. Wattenhofer, "On scaling decentralized blockchains," in *Financial Cryptography and Data Security*, J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, and K. Rohloff, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 106–125.

[24] K. Zhang and H. Jacobsen, "Towards dependable, scalable, and pervasive distributed ledgers with blockchains," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, vol. 00, Jul 2018, pp. 1337–1346. [Online]. Available: doi.ieeecomputersociety.org/10.1109/ICDCS.2018.00134

[25] J. Tsai, "Transform blockchain into distributed parallel computing architecture for precision medicine," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 1290–1299.

[26] J. Chiu and T. V. Koeppl, "Incentive compatibility on the blockchain," 2018.

[27] E. Benos, R. Garratt, and P. Gurrola-Perez, "The economics of distributed ledger technology for securities settlement," 2017.

[28] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer *et al.*, "On scaling decentralized blockchains," in *International conference on financial cryptography and data security*. Springer, 2016, pp. 106–125.

[29] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 3–16.

[30] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol," in *13th {USENIX} symposium on networked systems design and implementation ({NSDI} 16)*, 2016, pp. 45–59.

[31] R. Pass and E. Shi, "Hybrid consensus: Efficient consensus in the permissionless model," in *31st International Symposium on Distributed Computing (DISC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[32] L. Luu, V. Narayanan, K. Baweja, C. Zheng, S. Gilbert, and P. Saxena, "Scp: A computationally-scalable byzantine consensus protocol for blockchains," *See https://www. weusecoins. com/assets/pdf/library/SCP*, vol. 20, no. 20, p. 2016, 2015.

[33] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *25th {usenix} security symposium ({usenix} security 16)*, 2016, pp. 279–296.

[34] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 475–490.

[35] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.

[36] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Layer-two blockchain protocols."

[37] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments."

[38] A. Miller, I. Bentov, R. Kumaresan, and P. McCorry, "Sprites: Payment channels that go faster than lightning," *CoRR*, vol. abs/1702.05812, 2017. [Online]. Available: http://arxiv.org/abs/1702.05812

[39] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: ACM, 2018, pp. 949–966. [Online]. Available: http://doi.acm.org/10.1145/3243734.3243856

[40] J. Tuwiner. (2019) Bitcoin mining pools. [Online]. Available: https://www.buybitcoinworldwide.com/mining/pools/

[41] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," *Commun. ACM*, vol. 61, no. 7, pp. 95–102, Jun. 2018. [Online]. Available: http://doi.acm.org/10.1145/3212998

[42] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 949?966. [Online]. Available: https://doi.org/10.1145/3243734.3243856

[43] R. Canetti, "Universally composable security: a new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, Oct 2001, pp. 136–145.

[44] R. Canetti, A. Cohen, and Y. Lindell, "A simpler variant of universally composable security for standard multiparty computation," in *Annual Cryptology Conference*. Springer, 2015, pp. 3–22.

[45] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on computing*, vol. 18, no. 1, pp. 186–208, 1989.

[46] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment channels over cryptographic currencies."

[47] S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," *Commun. ACM*, vol. 28, no. 6, p. 637?647, Jun. 1985. [Online]. Available: https://doi.org/10.1145/3812.3818

[48] J. A. Garay and P. MacKenzie, "Concurrent oblivious transfer," in *Proceedings 41st Annual Symposium on Foundations of Computer Science*, Nov 2000, pp. 314–324.

[49] M. O. Rabin, "How to exchange secrets with oblivious transfer." *IACR Cryptology ePrint Archive*, vol. 2005, p. 187, 2005.

[50] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *Annual international cryptology conference*. Springer, 1993, pp. 232–249.

[51] R. Canetti and H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2001, pp. 453–474.

[52] D. Dolev, C. Dwork, and M. Naor, "Nonmalleable cryptography," *SIAM review*, vol. 45, no. 4, pp. 727–784, 2003.

[53] C. Dwork, M. Naor, and A. Sahai, "Concurrent zero-knowledge," *J. ACM*, vol. 51, no. 6, p. 851?898, Nov. 2004. [Online]. Available: https://doi.org/10.1145/1039488.1039489

[54] G. Avarikioti, G. Janssen, Y. Wang, and R. Wattenhofer, "Payment network design with fees," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2018, pp. 76–84.

[55] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Silentwhispers: Enforcing security and privacy in decentralized credit networks." in *NDSS*, 2017.

[56] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," *arXiv preprint arXiv:1709.05748*, 2017.

[57] V. Sivaraman, S. B. Venkatakrishnan, M. Alizadeh, G. Fanti, and P. Viswanath, "Routing cryptocurrency with the spider network," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, 2018, pp. 29–35.

[58] P. Prihodko, S. Zhigulin, M. Sahno, A. Ostrovskiy, and O. Osuntokun, "Flare: An approach to routing in lightning network," *White Paper*, 2016.

[59] R. Khalil, A. Gervais, and G. Felley, "Nocust-a securely scalable commit-chain," Cryptology ePrint Archive, Report 2018/642, Tech. Rep., 2018.

[60] J. Poon and V. Buterin, "Plasma: Scalable autonomous smart contracts," *White paper*, pp. 1–47, 2017.

[61] K. Atlas. (2017) The inevitability of privacy in lightning networks. [Online]. Available: https://www. kristovatlas. com/the-inevitability-of-privacy-in-lightning-networks/

[62] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "Tumblebit: An untrusted bitcoin-compatible anonymous payment hub," in *Network and Distributed System Security Symposium*, 2017.

[63] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 473–489.

[64] E. Heilman, F. Baldimtsi, and S. Goldberg, "Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions," in *International conference on financial cryptography and data security*. Springer, 2016, pp. 43–60.

[65] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, private smart contracts," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1353–1370.

[66] J. Teutsch and C. Reitwießner, "A scalable verification solution for blockchains," *arXiv preprint arXiv:1908.04756*, 2019.

[67] J. Peterson, J. Krug, M. Zoltu, A. K. Williams, and S. Alexander, "Augur: a decentralized oracle and prediction market platform," *arXiv preprint arXiv:1501.01042*, 2015.

[68] H. Ritzdorf, K. Wüst, A. Gervais, G. Felley, and S. Capkun, "Tls-n: Non-repudiation over tls enabling-ubiquitous content signing for disintermediation." *IACR Cryptology ePrint Archive*, vol. 2017, p. 578, 2017.

[69] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proceedings of the 2016 aCM sIGSAC conference on computer and communications security*, 2016, pp. 270–282.

[70] G. Arfaoui, S. Gharout, and J. Traoré, "Trusted execution environments: A look under the hood," in *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. IEEE, 2014, pp. 259–266.

[71] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: what it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1. IEEE, 2015, pp. 57–64.

[72] J. Adler, R. Berryhill, A. Veneris, Z. Poulos, N. Veira, and A. Kastania, "Astraea: A decentralized blockchain oracle," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 1145–1152.

[73] R. Kamiya, "Shintaku: An end-to-end-decentralized general-purpose blockchain oracle system," 2019.

[74] I. Eyal, "The miner's dilemma," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 89–103.

[75] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal selfish mining strategies in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 515–532.

[76] J. Kang, Z. Xiong, D. Niyato, P. Wang, D. Ye, and D. I. Kim, "Incentivizing consensus propagation in proof-of-stake based consortium blockchain networks," *IEEE Wireless Communications Letters*, vol. 8, no. 1, pp. 157–160, 2018.

[77] K. Basu, "Stackelberg equilibrium in oligopoly: an explanation based on managerial incentives," *Economics Letters*, vol. 49, no. 4, pp. 459–464, 1995.

[78] D. B. Rawat, L. Njilla, K. Kwiat, and C. Kamhoua, "ishare: Blockchain-based privacy-aware multi-agent information sharing games for cybersecurity," in *2018 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2018, pp. 425–431.

[79] S. Pearson and M. Casassa Mont, "Sticky policies: an approach for privacy management across multiple parties," *IEEE Computer*, vol. 44, no. 9, pp. 60–68, 2011.

[80] M. C. Mont, S. Pearson, and P. Bramhall, "Towards accountable management of identity and privacy: sticky policies and enforceable tracing services," in *14th International Workshop on Database and Expert Systems Applications, 2003. Proceedings.*, Sep. 2003, pp. 377–382.

[81] S. Bandhakavi, C. C. Zhang, and M. Winslett, "Super-sticky and declassifiable release policies for flexible information dissemination control," in *Proceedings of the 5th ACM Workshop on Privacy in Electronic Society*, ser. WPES '06. New York, NY, USA: ACM, 2006, pp. 51–58. [Online]. Available: http://doi.acm.org/10.1145/1179601.1179609

[82] D. W. Chadwick and S. F. Lievens, "Enforcing "sticky" security policies throughout a distributed application," in *Proceedings of the 2008 Workshop on Middleware Security*, ser. MidSec '08. New York, NY, USA: ACM, 2008, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/1463342.1463343

[83] M. Casassa-Mont, I. Matteucci, M. Petrocchi, and M. L. Sbodio, "Towards safer information sharing in the cloud," *International Journal of Information Security*, vol. 14, no. 4, pp. 319–334, Aug 2015. [Online]. Available: https://doi.org/10.1007/s10207-014-0258-5

[84] V. Pappas, V. P. Kemerlis, A. Zavou, M. Polychronakis, and A. D. Keromytis, "CloudFence: Data Flow Tracking as a Cloud Service," in *Research in Attacks, Intrusions, and Defenses*, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Oct. 2013, pp. 411–431.

[85] D. Muthukumaran, D. O'Keeffe, C. Priebe, D. Eyers, B. Shand, and P. Pietzuch, "FlowWatcher: Defending Against Data Disclosure Vulnerabilities in Web Applications," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: ACM, 2015, pp. 603–615. [Online]. Available: http://doi.acm.org/10.1145/2810103.2813639

[86] T. F. J. . Pasquier, J. Singh, D. Eyers, and J. Bacon, "Camflow: Managed data-sharing for cloud services," *IEEE Transactions on Cloud Computing*, vol. 5, no. 3, pp. 472–484, July 2017.

[87] I. Papagiannis, M. Migliavacca, and P. Pietzuch, "Php aspis: using partial taint tracking to protect against injection attacks," in *WebApps' 11: Proceedings of the 2nd USENIX conference on Web application development*. USENIX Association, 2011, pp. 13–24.

[88] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Trans. Comput. Syst.*, vol. 32, no. 2, pp. 5:1–5:29, Jun. 2014. [Online]. Available: http://doi.acm.org/10.1145/2619091

[89] S. Sundareswaran, A. Squicciarini, and D. Lin, "Ensuring distributed accountability for data sharing in the cloud," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 4, pp. 556–568, July 2012.

[90] J. Bacon, D. Eyers, T. F.-M. Pasquier, J. Singh, I. Papagiannis, and P. Pietzuch, "Information flow control for secure cloud computing," *IEEE Transactions on Network and Service Management*, vol. 11, no. 1, pp. 76–89, 2014.

[91] J. Singh, J. Powles, T. Pasquier, and J. Bacon, "Data flow management and compliance in cloud computing," *IEEE Cloud Computing*, vol. 2, no. 4, pp. 24–32, 2015.

[92] J. W. Holford, W. J. Caelli, and A. W. Rhodes, "Using self-defending objects to develop security aware applications in java?" in *Proceedings of the 27th Australasian conference on Computer science-Volume 26*. Australian Computer Society, Inc., 2004, pp. 341–349.

[93] R. Neisse, G. Steri, and I. Nai-Fovino, "A blockchain-based approach for data accountability and provenance tracking," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, ser. ARES

'17. New York, NY, USA: ACM, 2017, pp. 14:1–14:10. [Online]. Available: http://doi.acm.org/10.1145/3098954.3098958

[94] J. Reardon, D. Basin, and S. Capkun, "Sok: Secure data deletion," in *2013 IEEE Symposium on Security and Privacy*, May 2013, pp. 301–315.

[95] D. Boneh and R. J. Lipton, "A revocable backup system," in *Proceedings of the 6th Conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6*, ser. SSYM'96. USA: USENIX Association, 1996, p. 9.

[96] R. Kissel, A. Regenscheid, M. Scholl, and K. Stine, *Guidelines for media sanitization*, 2014.

[97] G. F. Hughes, T. Coughlin, and D. M. Commins, "Disposal of disk and tape data by secure sanitization," *IEEE Security Privacy*, vol. 7, no. 4, pp. 29–34, July 2009.

[98] J. Chow, B. Pfaff, T. Garfinkel, and M. Rosenblum, "Shredding your garbage: Reducing data lifetime through secure deallocation," in *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, ser. SSYM'05. USA: USENIX Association, 2005, p. 22.

[99] C. Priebe, D. Muthukumaran, D. O' Keeffe, D. Eyers, B. Shand, R. Kapitza, and P. Pietzuch, "Cloudsafetynet: Detecting data leakage between cloud tenants," in *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, ser. CCSW '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 117?128. [Online]. Available: https://doi.org/10.1145/2664168.2664174

[100] A. Rahumed, H. C. H. Chen, Y. Tang, P. P. C. Lee, and J. C. S. Lui, "A secure cloud backup system with assured deletion and version control," in *2011 40th International Conference on Parallel Processing Workshops*, Sep. 2011, pp. 160–167.

[101] R. Perlman, "File system design with assured delete," in *Third IEEE International Security in Storage Workshop (SISW'05)*, Dec 2005, pp. 6 pp.–88.

[102] J. Reardon, S. Capkun, and D. Basin, "Data node encrypted file system: Efficient secure deletion for flash memory," in *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*. Bellevue, WA: USENIX, 2012, pp. 333–348. [Online]. Available: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/reardon

[103] K. M. Ramokapane, A. Rashid, and J. M. Such, "Assured deletion in the cloud: Requirements, challenges and future directions," in *Proceedings of the*

2016 ACM on Cloud Computing Security Workshop, ser. CCSW '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 97–108. [Online]. Available: https://doi.org/10.1145/2996429.2996434

[104] S. L. Garfinkel and A. Shelat, "Remembrance of data passed: a study of disk sanitization practices," *IEEE Security Privacy*, vol. 1, no. 1, pp. 17–27, Jan 2003.

[105] A. Coglio, "Ethereum's recursive length prefix in acl2," 2019.

[106] D. N. R. (Australia). (2013) Domain name length statistics. [Online]. Available: https://www.domainregistration.com.au/news/2013/1301-domain-length.php

[107] Ropsten testnet pow chain. [Online]. Available: https://github.com/ethereum/ropsten

[108] S. K. Kim, Z. Ma, S. Murali, J. Mason, A. Miller, and M. Bailey, "Measuring ethereum network peers," in *Proceedings of the Internet Measurement Conference 2018*, ser. IMC '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 91–104. [Online]. Available: https://doi.org/10.1145/3278532.3278542

[109] Ropsten testnet exporer. [Online]. Available: https://ropsten.etherscan.io/