Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

TOWARDS PRACTICALIZATION OF BLOCKCHAIN-BASED DECENTRALIZED APPLICATIONS

by
Songlin He

Blockchain can be defined as an immutable ledger for recording transactions, maintained in a distributed network of mutually untrusting peers. Blockchain technology has been widely applied to various fields beyond its initial usage of cryptocurrency. However, blockchain itself is insufficient to meet all the desired security or efficiency requirements for diversified application scenarios. This dissertation focuses on two core functionalities that blockchain provides, i.e., robust storage and reliable computation. Three concrete application scenarios including Internet of Things (IoT), cybersecurity management (CSM), and peer-to-peer (P2P) content delivery network (CDN) are utilized to elaborate the general design principles for these two main functionalities. Among them, the IoT and CSM applications involve the design of blockchain-based robust storage and management while the P2P CDN requires reliable computation. Such general design principles derived from disparate application scenarios have the potential to realize practicalization of many other blockchain-enabled decentralized applications.

In the IoT application, blockchain-based decentralized data management is capable of handling faulty nodes, as designed in the cybersecurity application. But an important issue lies in the interaction between external network and blockchain network, i.e., external clients must rely on a relay node to communicate with the full nodes in the blockchain. Compromization of such relay nodes may result in a security breach and even a blockage of IoT sensors from the network. Therefore, a censorship-resistant blockchain-based decentralized IoT management system is proposed. Experimental results from proof-of-concept implementation and

deployment in a real distributed environment show the feasibility and effectiveness in achieving censorship resistance.

The CSM application incorporates blockchain to provide robust storage of historical cybersecurity data so that with a certain level of cyber intelligence, a defender can determine if a network has been compromised and to what extent. The CSM functions can be categorized into three classes: Network-centric (N-CSM), Tools-centric (T-CSM) and Application-centric (A-CSM). The cyber intelligence identifies new attackers, victims, or defense capabilities. Moreover, a decentralized storage network (DSN) is integrated to reduce on-chain storage costs without undermining its robustness. Experiments with the prototype implementation and real-world cyber datasets show that the blockchain-based CSM solution is effective and efficient.

The P2P CDN application explores and utilizes the functionality of reliable computation that blockchain empowers. Particularly, P2P CDN is promising to provide benefits including cost-saving and scalable peak-demand handling compared with centralized CDNs. However, reliable P2P delivery requires proper enforcement of delivery fairness. Unfortunately, most existing studies on delivery fairness are based on non-cooperative game-theoretic assumptions that are arguably unrealistic in the ad-hoc P2P setting. To address this issue, an expressive security requirement for desired fair P2P content delivery is defined and two efficient approaches based on blockchain for P2P downloading and P2P streaming are proposed. The proposed system guarantees the fairness for each party even when all others collude to arbitrarily misbehave and achieves asymptotically optimal on-chain costs and optimal delivery communication.

TOWARDS PRACTICALIZATION OF BLOCKCHAIN-BASED DECENTRALIZED APPLICATIONS

by
Songlin He

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

May 2022

APPROVAL PAGE

TOWARDS PRACTICALIZATION OF BLOCKCHAIN-BASED DECENTRALIZED APPLICATIONS

Songlin He

Prof. Chase Wu, Dissertation Co-Advisor — Date
Professor of Computer Science, NJIT

Prof. Qiang Tang, Dissertation Co-Advisor — Date
Associate Professor of Computer Science,
The University of Sydney (USYD), Darlington, Australia

Prof. Cristian Borcea, Committee Member — Date
Professor of Computer Science, NJIT

Prof. Ali Mili, Committee Member — Date
Professor of Computer Science, NJIT

Prof. Shouhuai Xu, Committee Member — Date
Gallogly Chaired Professor of Computer Science,
University of Colorado Colorado Springs (UCCS), Colorado, USA

BIOGRAPHICAL SKETCH

Author: Songlin He

Degree: Doctor of Philosophy

Date: May 2022

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, New Jersey, 2022
- Master of Signal and Information Processing,
  Zhejiang Sci-Tech University, Hangzhou, Zhejiang, 2017
- Bachelor of Broadcast and Television Engineering,
  Communication University of Zhejiang, Hangzhou Zhejiang, 2014

Major: Computer Science

Presentations and Publications:

Songlin He, Eric Ficke, Mir Mehedi Pritom, Huashan Chen, Qiang Tang, Qian Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu. "Blockchain-Based Automated and Robust Cyber Security Management." in *Journal of Parallel and Distributed Computing (JPDC'22)*, 2022.

Songlin He, Yuan Lu, Qiang Tang, Guiling Wang, and Chase Wu. "Fair Peer-to-Peer Content Delivery atop Blockchain." in *Proceedings of the 26th European Symposium on Research in Computer Security (ESORICS'21)*, pp. 348-369, Virtual Event, Germany, October 2021. (Paper presentation virtually)

Xuewen Shen, Songlin He, Dingguo Yu, and Zhiyan Tang. "Video Preview Generation Based on Playback Records." in *the 17th IEEE/ACS International Conference on Computer Systems and Applications (AICCSA'20)*, pp. 1-6, Virtual Event, Turkey, November 2020. (Paper presentation virtually)

Songlin He, Qiang Tang, Chase Wu and Xuewen Shen. "Decentralizing IoT Management Systems Using Blockchain for Censorship Resistance." in *IEEE Transactions on Industrial Informatics (TII'19)*, vol. 16, pp. 715-727, 2019.

Haifa AlQuwaiee, Songlin He, Chase Wu, and Xuewen Shen. "On Distributed Information Composition in Big Data Systems." in *Proceedings of the 15th Conference on eScience (eScience'19)*, pp. 168-177, San Diego, USA, September 2019.

Songlin He, Qiang Tang, and Chase Wu. "Censorship Resistant Decentralized IoT Management Systems." in *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (DLoT'18)*, pp. 454-459, NY, USA, November 2018. (Paper presentation in Columbia University, NY, USA)

Songlin He, Tong Sun, Qiang Tang, Chase Wu, Nedim Lipka, Curtis Wigington, and Rajiv Jain. "Secure and Efficient Agreement Signing atop Blockchain and Decentralized Identity." *paper under review*, 2022.

Songlin He, Yuan Lu, Qiang Tang, Guiling Wang, and Chase Wu. "Blockchain-Based P2P Content Delivery with Monetary Incentivization and Fairness Guarantee." *paper under review*, 2022.

Songlin He, Tong Sun, Nedim Lipka, Curtis Wigington, and Rajiv Jain. "Privacy-Preserved Auto-Filling for Agreements with Verifiable Identity Information." *patent under review*, 2022.

Songlin He, Tong Sun, Nedim Lipka, Curtis Wigington, and Rajiv Jain. "Toward Efficient Signing and Verification for Agreements." *patent under review*, 2022.

Songlin He, Eric Ficke, Mir Mehedi Pritom, Huashan Chen, Qiang Tang, Qian Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu. "Method and System for Blockchain-Based Cyber Security Management." *patent under review*, 2022.

Songlin He, Qiang Tang, and Chase Wu. "DIdM-IoT: Legacy Compatible and Sybil Resistant Decentralized Identity Management for Internet of Things." *manuscript ready to submit*, 2022.

To my beloved family

ACKNOWLEDGMENT

My deepest gratitude goes to my advisors Prof. Chase Wu and Prof. Qiang Tang for their valuable guidance, support, and constant motivation during my doctoral journey. They taught me the fundamentals of how to conduct scientific research, what valuable research is and how it can be done. Along the way, their rigorous scientific attitude has deeply influenced me for my future career. I deeply appreciate their time, ideas, discussions, encouragements, and supports in all aspects to guide me gradually becoming a researcher thinking independently.

I would also express my sincere thanks to the committee members including Dr. Cristian Borcea, Dr. Ali Mili, and Dr. Shouhuai Xu for their precious time, great support, and valuable comments.

It is worth noting that without the generous support from multiple sources including the U.S. National Science Foundation (CNS-1801492), Air Force Research Lab (FA8750-19-1-0019), and Computer Science Department of NJIT, it would be impossible to complete my Ph.D. degree.

I would like to thank my labmates for their enriching discussion in research and all valuable help during my Ph.D. life. An incomplete list includes Dr. Yuan Lu, Dr. Long Chen, Mr. Zhenliang Lu, Dr. Huiyan Cao, Dr. Wuji Liu, Mr. Dong Wei, Mrs. Qianwen Ye, Dr. Haifa AlQuwaiee, and more. I will never forget the wonderful time we had together.

Also, I would like to express gratitude to my master advisor (Dr. Wei Shen) and bachelor advisor (Dr. Yiliang Hu) as well as all friends I met in my life. Thanks for encouraging me to pursue my dreams.

No words can express how grateful I am for my lovely family: my parents (Mr. Jiaqiang He and Mrs. Cuihua Liu), my siblings, my close relatives for their endless love and support. Millions of thanks to my grandparents Mr. Yongfa He

and Mrs. Bihua Xian for raising me and my uncle Mr. Shaofa Liu for shaping my characteristics since my childhood. Special thanks to Miss Min Liu for accompanying me during all the tough times in the Ph.D. journey.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

In 2008, Satoshi Nakamoto proposed Bitcoin [110] that aims at a cryptographic currency for trustless online payment [111], whose backbone protocol was later formally proven secure [50]. As the underpinning technology, blockchain (or *distributed ledger*) has been reckoned as the next generation of *value exchange network*, being a complementary component to the existing *information exchange network*, namely, the Internet, and gained wide-spread attention among both industry and academia in recent years. Despite the fact that cryptocurrencies have emerged as the most popular application of blockchain technology, many enthusiasts from different fields have sensed the huge potentials and proposed a range of applications across a multitude of application domains [30]. According to different application scenarios, the development of blockchain can be divided into three stages: In blockchain stage 1.0, it is the era of virtual cryptocurrency represented by Bitcoin [110]. Stage 2.0 is represented by the public blockchain platform, Ethereum [148], which provides Turing-complete programming language to run pre-determined execution logic (dubbed *smart contract*) under certain conditions. This stage is especially for financial applications. Blockchain 3.0 refers to various application scenarios besides the financial field, which can satisfy more complex business demands. Indeed, as individuals embrace Web 3.0 where data with *semantic* meanings are interconnected in a *decentralized* manner, blockchain has been recognized as one of the most fundamental technologies to revolutionize the landscape of the identified application domains [45].

1.1 Blockchain and Related Layers

Blockchain can be defined as an immutable ledger for recording transactions, maintained in a distributed network amongst untrusting peer nodes. Figure 1.1 illustrates the suggested layers relevant to blockchain:

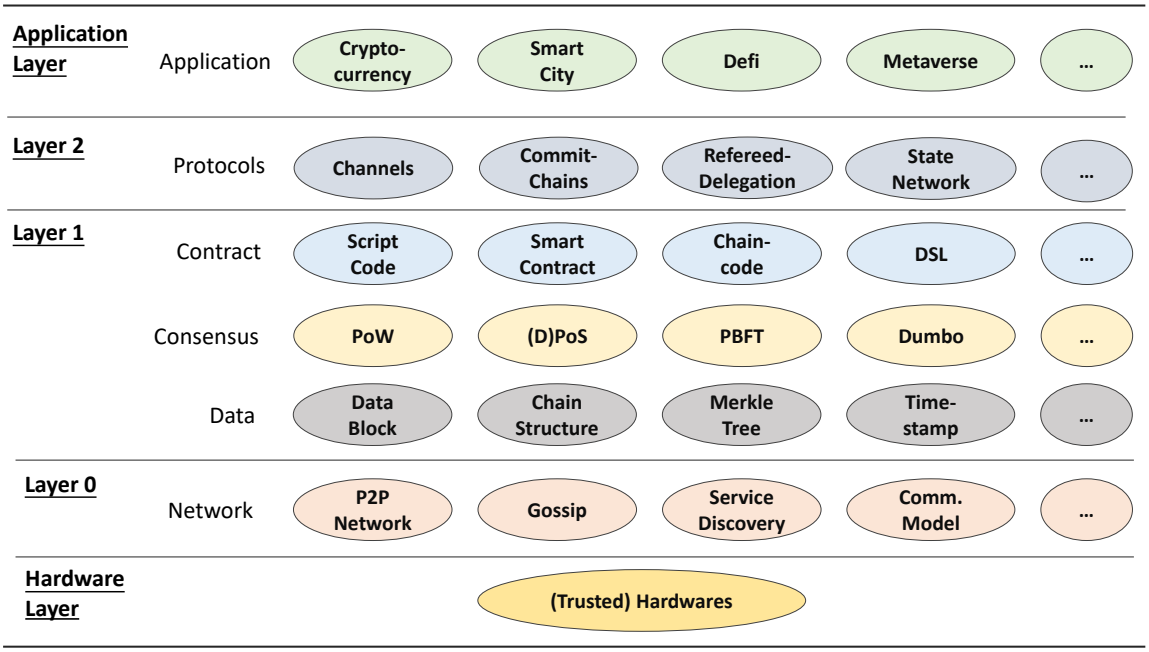


Figure 1.1 The layers related to blockchain.

The *hardware layer* provides the underlying hardwares. Regular hardwares include physical servers, switches, or routers etc. Moreover, the Trusted Execution Environment (TEE) (e.g., Intel SGX [29]) equipped with modern computers can execute senstive or security-critical program code in a specified memory space, called *enclaves*, which is tamper-proof from the operating system or other higher-privileged softwares [57, 77].

The *layer-zero*, or the *network layer*, is typically a peer-to-peer (P2P) network on which blockchain nodes can exchange information asynchronously [112]. Such a network layer not only contains the traditional network architecture, which concentrates on Internet routing, but also ensures reliable communication among participants in a blockchain.

The *layer-one* refers to the core component of blockchain, which maintains an immutable ledger for recording transactions. It further can be divided into three sub-layers: (i) *data layer*, which defines the data structures and required data fields; (ii) *consensus layer* contains various consensus mechanisms, e.g., Proof of Work (PoW) [50], Proof of Stake (PoS) [82], Raft [115], PBFT [23], BFT-SMaRt [16], HoneyBadger [107], Dumbo [58, 97] and others [149]. It is worth pointing out that these consensus mechanisms can further be categorized due to applicable blockchain types (*permissioned* vs. *permissionless*), trust models (*crash fault tolerance* (CFT) vs. *byzantine fault tolerance* (BFT)) or network models (*synchronous*, *partial synchronous* and *asynchronous*). Typically a consensus protocol is characterized by three fundamental properties: *termination*, *agreement* and *safety*; (iii) *contract layer* realizes smart contract, e.g., Bitcoin has limited support for smart contract, called *scripts*. Ethereum [148] defines domain specific language (DSL), viz. *solidity* and the storage or execution costs for all operations (in unit of *gas*) in the smart contract. Hyperledger Fabric (HLF) [9] provides a separated sandbox running environment so that the smart contract, called *chaincode*, can be developed in various advanced programming language such as Golang, NodeJS, Java, deployed and executed in Docker containers or Kubernetes [35].

The *layer-two* treats the layer-one blockchains as oracles, which provide the desired properties of *integrity* (i.e., only valid transactions are appended to the ledger) and *eventual synchronicity with an upper time-bound* (i.e., any valid transaction will eventually be added to the ledger before a critical timeout) [57], with the key purpose of considerably improving the *scalability* of layer-one blockchains. The existing layer-two protocols can further be categorized as three research directions: (i) payment and state channels, e.g., Sprites [105], Perun [38]; (ii) commit-chains, e.g., NOCUST [80] and Plasma [121] and (iii) protocols for refereed delegation, e.g., TrueBit [140] and Arbitrum [74].

The *application layer* envisions diversified application scenarios that can be empoered by blockchain technology. Exemplary application domains include cryptocurrencies [110, 148], (industrial) Internet of Things (IoT) [63], content delivery [61], supply chain [123], healthcare [103], non-fungible tokens (NFTs) [142], decentralized finance [6, 145], decentralized storage [14], decentralized identity [99], metaverse [113] and etc.

In this dissertation, we mainly focus on the blockchain-enabled applications, which closely related to the application layer. One the one hand, the practicalization of blockchain in real-life application scenarios would significantly drive the development of blockchain technology. On the other hand, it is worth stressing that the realization of provably secure protocol in the application layer is essentially highly pertinent to the design of other layers requiring comprehensive considerations.

1.2 Benefits of Blockchain-Based Applications

Blockchain-based application model exhibits promising aspects to solve many problems that confronted by the existing centralized system (e.g., single point of failure) due to its distributed architecture and all advantageous properties brought by the blockchain technology [50]. Specifically, a set of benefits empowered by blockchain-based decentralized applications can be highlighted as follows:

- *Availability.* The blockchain network is in a distributed architecture, which enables the system available even though partial nodes are unreachable.
- *Immutability.* The transactions cannot be reverted once ending up in the blockchain considering the number of simultaneously corrupted nodes are under a threshold.
- *Consistency.* There is a consistent and global ledger state for anyone who views the blockchain as blockchain stores transaction or state after reaching consensus.
- *Accountability.* Since the transactions are immutable on-chain, it is feasible to realize accountability if any party performs malicious activities.

- *Provenance.* Consider that the data submitted to blockchain is valid, then according to the immutability property of transaction histories on-chain, blockchain provides tamper-proof information about the origin of data records.

These properties inherent in the design of blockchain provide many possibilities to enhance various existing application scenarios.

1.3 Potential Problems in General Blockchain-Based Applications

In essence, blockchain provides two core functionalities, i.e., *storage* and *computation*. Unfortunately, blockchain itself is *not* panacea to fit all application settings. With the general architecture of blockchain-enabled decentralized applications, as shown in Figure 1.2, the following potential problems are worth of consideration.

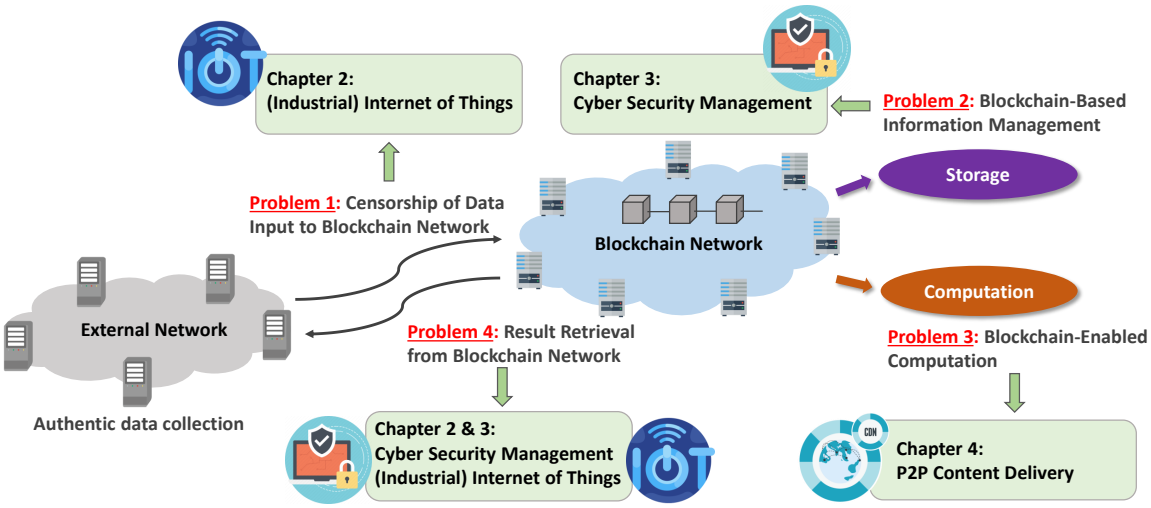


Figure 1.2 The potential problems in a general architecture of blockchain-enabled decentralized applications and our contributions to these problems in the dissertation.

Problem 1. Censorship of data input for blockchain network. Blockchain full nodes need to synchronize all the blocks and participate the consensus process. However, mobile devices (e.g., IoT devices, smart phones, smart wearables) or browser environments are unable to afford the cost of storing a huge volume of blocks or executing computation-intensive consensus. Consequently, those light clients need to hinge on a blockchain full node to relay transactions to all other full nodes [63]. The

challenge arises where the connected full node or the routing to the full node may be corrupted, leading to invalid data input to the blockchain network.

Problem 2. Blockchain-based robust information management. Blockchain, especially permissioned, resembles the functionality of conventional distributed database to provide robust storage [129]. This is because the system would reach consensus before eventually writing the data to ledger, which provide an extra security layer ensuring that even partial of the nodes are corrupted and arbitrarily misbehave, the system still work normally. However, many issues need to be tackled when leveraging blockchain as a platform providing robust storage. For example, efficiently handling a large amount of submitted data.

Problem 3. Blockchain-enabled computation. Two key issues need to be considered when utilizing blockchain as a computing platform: (i) *privacy*. Blockchain inherently exhibits the property of transparency [84]. In a permissionless setting, the entire chain is replicated to all peer nodes in a public network and the whole state is accessible to all for verification. Some data (e.g., random bits) posted on-chain may not impact privacy while some sensitive data may leak important information. On the other hand, a permissioned blockchain provides a promising alternative to mitigate this issue, which however, requires participants to authenticate themselves and may not be suitable for some public uses cases where participants are unknown to each others; (ii) *limited computation power*. The computation power provided by blockchain (essentially the smart contract) is limited. In Proof-of-Work (PoW) or Proof-of-Stake (PoS) based consensus protocols, such a limitation is derived from the known *verifier's dilemma* [98], e.g., there is a global *gas limit* in Ethereum that specifies a maximum amount (i.e., 8,000,000) of gas that can be spent in a single block. While for a permissioned setting, too complex on-chain execution may lead to unacceptable latency and undesirable user experiences.

Problem 4. Valid result retrieval from blockchain network. Smart contract in blockchain can ensure the guaranteed execution of pre-determined logic. However, a typical operation of retrieving data that stored in the ledger or the invocation of smart contract still requires the connection with a full node in the blockchain network, and if it is corrupted, the retrieval results would become invalid. A naive solution connecting with many full nodes would cause considerable communication overhead. Hence, it remains unclear of how to solve this problem.

Authenticity of submitted data to blockchain network. As an orthogonal and hot research direction, ensuring the authenticity of submitted data to blockchain network is of great importance. Specifically, the on-chain storage and execution is trustable in the sense that: (i) the immutability property of blockchain ensures the integrity of stored data on-chain; (ii) the smart contract can use the on-chain stored data to execute pre-determined programs without manipulation. However, the authenticity of submitted data is not guaranteed by blockchain itself. Such a problem requires extra system design, i.e., *oracle protocols* [151, 152].

These above problems appeared in blockchain-based decentralized applications are general and solving these issues would not only improve the system security, but also, from a broader perspective, accelerate the practicalization of applying blockchain technology to real-life scenarios.

1.4 Main Contributions and Dissertation Structure

It this dissertation, we investigate the main usage of blockchain regarding *storage* and *computation* in several application scenarios including cyber security, industrial Internet of Things and peer-to-peer content delivery. Through the design of protocols in these specific application settings, we essentially provide the general solutions to solve all the problems mentioned earlier. Overall, the remaining of the dissertation is organized as follows.

Chapter 2 focuses on the protocol design of interactions between external network and blockchain-based information management system (i.e., the solution to the Problems 1 and 4 in Figure 1.2) by considering the (industrial) Internet of Things (IoT) application scenario. The key observation is that the blockchain network itself is robust to provide data management, as our later proposed design for the cyber security management. However, when interacting with external networks, e.g., the sensor network in the IoT setting, the data sending from sensor network to blockchain network (inbound flow) or the data (e.g., a command) retrieved from the blockchain network for the sensor network, e.g., actuators, (outbound flow) may be censored in the sense of being maliciously mixed, tampered or dropped. To this end, to handle the inbound flow, we proposed a gossip-based diffusion mechanism and augmented consensus designs to realize to censorship resistance between IoT network and the blockchain-based decentralized data management systems, while for the outbound flow, we proposed a multi-party invocation mechanism to enable reliability and a signature aggregation mechanism to provide efficiency for retrieving results.

Chapter 3 discusses the designs of blockchain-enabled information management system (i.e., the solution mainly to Problems 2 and 4 in Figure 1.2) in the specific cyber security setting, where blockchain is incorporated to provide robust and automated cyber security management so that a defender can detect the potential attacks based on robustly stored historical cyber data and newly given cyber intelligence, and be aware of the degree that the managed network has been damaged. The role of blockchain mainly lies in two aspects: the first is for robust storage of collected cyber data, which is analogue to conventional distributed database while being more secure and robust; the second is for automated and guaranteed execution of cyber functionalities hinging on pre-determined smart contract and therefore less manual inference is needed.

Chapter 4 presents the designs regarding blockchain-empowered computation (viz. the solution to the Problem 3 in Figure 1.2) in the specific P2P content delivery setting. In particular, the key challenge of designing a P2P content delivery protocol is to rigorously guarantee fairness. However, there exist several challenges, e.g., conventional fairness definition is insufficient to the specific P2P content delivery setting and it is well-known fairness cannot be completely guaranteed without a trusted third party (TTP). We therefore defined more fine-grained fairness, leveraged blockchain to play the role of a TTP and proposed both downloading-setting and streaming-setting protocol designs. In addition, we elaborate many design considerations to protect privacy against corrupted system participants and optimize the on-chain storage and computational costs.

Chapter 5 concludes the dissertation, provides reflections, and points out several future research directions.

CHAPTER 2

CENSORSHIP RESISTANCE BETWEEN BLOCKCHAIN AND IIOT

This chapter presents the design of interactions between the external network and the blockchain-based information management system in the specific industrial Internet of Things setting [62, 63].

2.1 Introduction

Spawned from machine-to-machine (M2M) technology, Internet-of-Things (IoT) is becoming a dynamic global network infrastructure with self-configuring capabilities where physical and virtual "things" with identities, physical attributes, and virtual personalities are seamlessly integrated into the information network [85]. According to Statista, the number of IoT devices worldwide will be over 75 billion by 2025 [136]. IoT is recognized as one of the most important areas of future technology and is gaining vast attention from a wide range of industries [91].

As a subset and natural evolution of IoT, Industrial Internet of Things (IIoT) shares common technologies (sensors, cloud platforms) with IoT but has higher requirements on security, scalability and reliability. One example of the IIoT vision is the Industrial Internet of Things Services and People (IoTSP) platform [93]. The rapid development of IIoT is facilitated by the capability of data generation, collection, aggregation, and analysis over the Internet to maximize the efficiency of machines and the throughput of operations. This brings about significant challenges since data may flow across various boundaries at the risk of attacks or failures.

Specifically, existing IoT systems (including IIoT) mainly rely on centralized service, where sensors collect and send data directly to a central server on the cloud for analysis, as shown in Figure 2.1. This model has several drawbacks. For example, the cloud server may present a single point of failure; clouds are typically vendor

specific and may not be compatible with each other, thus adversely affecting data sharing between them. Also, existing centralized IoT solutions are expensive due to a high cost in infrastructure and maintenance. Among these shortcomings, security is of primary concern. By 2022, half of the security budgets for IoT will be allocated to fault remediation, recalls, and safety failures rather than protection [117]. Therefore, a distributed trust technology ensuring security is regarded as a cornerstone for the continual growth of such IoT solutions. The blockchain technology is under rapid development and has proved to be an effective solution to realizing such goals due to its intrinsic security [117].

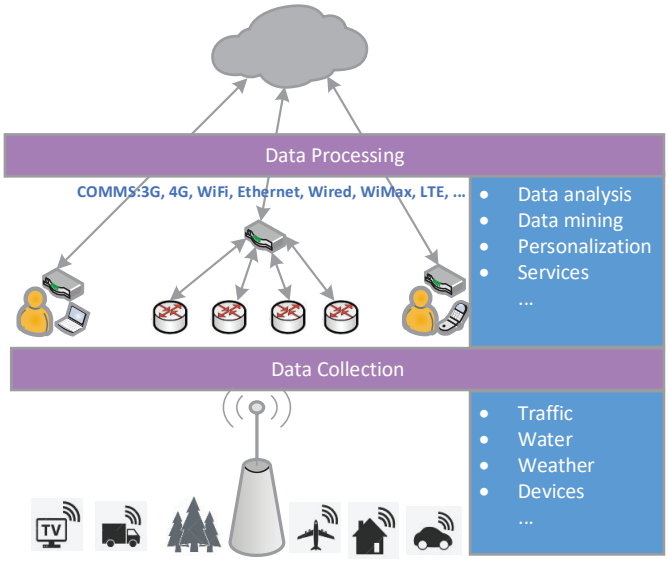


Figure 2.1 A centralized IoT system architecture.

Blockchain is typically viewed as an immutable *ledger* for recording *transactions*, maintained in a distributed network of mutually untrusting *peers* [9]. Any participating peer can submit data (sometimes also referred to as a transaction), which is eventually broadcasted and replicated at all participating peers executing some consensus protocols. As an abstract layer, blockchain technology provides a reliable delivery of messages to ensure that all participating peers have a consistent copy of the ledger. This is referred to as "transparency", a property frequently mentioned

about blockchain; on the other hand, once a message is written to the ledger and replicated at all peers, each peer can only modify its local ledger and the data would remain intact in other peers' ledgers. This is referred to as "immutability", another important property of blockchain. A more unique function of blockchain is to support "smart contract"[1], which is a piece of program code that implements a pre-defined application logic and deterministically runs on all participating peers.

The above properties of blockchain technology have facilitated its widespread applications to the IoT domain. For example, the "immutability" property of blockchain brings resistance to unauthorized modification. Since the entire history of device configuration is stored in the blockchain, recovery from incidents is straightforward. Depending on whether or not peers need to be authorized, blockchain technology is divided into two main categories: permissioned and permissionless. In this study, we focus on *permissioned blockchain* where participating nodes are all certified and known to others. In a more visionary level, IBM laid out a blueprint for "device democracy" [19], which employs blockchain to distributively manage transaction processing and coordination among hundreds of billions of interacting devices. Such an ambitious goal might take time to come to life, but on the other hand, decentralizing local management systems via permissioned blockchain to improve robustness and availability is much more viable [76].

Problems. Although blockchain technology offers a promising way to decentralize IoT management systems, such decentralization cannot be realized completely based on *existing* blockchain platforms such as the popular permissioned blockchain, Hyperledger fabric (Fabric for short) [9]. Note that blockchain technology (in particular, the consensus protocol) itself only concerns how to replicate data across peers consistently. Many practical issues such as data input from external sources and data output from the ledger are not considered by the consensus protocol. These

[1]The "scripts" in Bitcoin is a predecessor of smart contract, while in ethereum [148], it is a collection of code (functions) and data (state) that reside at a specific address.

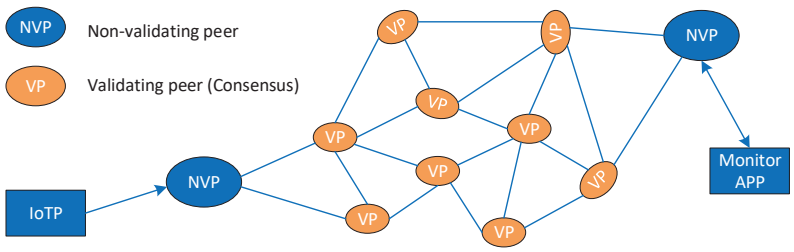


Figure 2.2 Data flow in a hyperledger fabric-based IoT management system.

problems are currently subject to ad-hoc designs and could potentially become a bottleneck in revealing the full power of a decentralized system.

Normally, multiple sensors are connected to one server (referred to as *gateway node*, which is one of the non-validating nodes and whose goal is to settle with the heterogeneity between different sensor networks and the cloud and effectively retrieve data from sensor networks [153]), and the server is responsible for forwarding on behalf of the sensors and participating in the consensus protocol to post the collected data to the distributed ledger. Obviously, if this gateway node is corrupted, sensor messages cannot be even transmitted to any of the blockchain's full nodes, thus the sensor simply loses the ability of "writing" to the ledger. In fact, such kind of architecture is common in existing systems, for example, Figure 2.2 shows the data flow when building IoT application on top of Hyperledger Fabric [67].

Problem 1. The gateway node, i.e., the non-validating peer (NVP) node in Fabric, could be censored. Consider a (potentially decentralized) IoT management system for environmental monitoring, where interested departments control the gateway node. The notorious *Flint water crisis* is a practical example and lesson. Flint authorities insisted for months that the city water was safe to drink, but finally it was reported that the Michigan Department of Environment Quality and the city of Flint discarded two of the collected samples containing a dangerous level of lead to avoid high cost and lawsuit [146].

Problem 2. The query result from the blockchain network could be censored. As an IoT management system, besides writing data into the ledger, sometimes actuators/devices may also need to read or receive instructions from the ledger. Similarly, at present, such message passing out of the blockchain is still carried out via an external non-validating node, which connects to one or several full nodes[2] in the blockchain network. If this external node or its connected full nodes are hacked/censored to be malicious, e.g., critical control commands are dropped, serious consequences may occur.

Consider the application of decentralized energy IoT management, the sensors continuously send real-time environmental measurements to the ledger, and the management servers analyze these measurements and send instructions back to the actuators. For example, if the temperature or pressure reaches a threshold, the servers need to instruct the actuators to shut the valve or reduce the amount of oxygen pumped into the combustion facility. If the forwarding node is compromised, such instructions may be dropped or modified on purpose to create a disaster.

These problems motivate us to consider how to build a *censorship resistant* decentralized IoT management system.

Contributions. We design a protocol that decentralizes the message passing module for sending and receiving data from a distributed ledger, thus avoid the single point of failure at the gateway node; moreover, this is done in a way that is compatible with existing consensus protocols so that our method can be plugged into existing platforms, as detailed below.

- First, we propose to replace the traditional gateway node in IoT scenarios with several *seed nodes*, which perform the same function as gateway node but also participate in blockchain network as *full nodes*. Then we introduce a message "diffusion" mechanism to realize censorship resistance considering the single entry point problem and propose an augmented consensus protocol to achieve reliable data delivery.

[2]Full nodes execute, validate and commit transactions to the ledger in a blockchain network. Each of them maintains a copy of the ledger state.

- Furthermore, we propose the protocol to deal with the single exit point problem and the case that data on at most $\frac{1}{3}$ of all full nodes are maliciously modified.
- Last, we propose to leverage the cryptographic tool of public key aggregation to reduce communication overhead and complexity of verification.

2.2 Related Work

Integration of IoT and blockchain. Billions of connected devices in future IoT networks face significant technical challenges in security, privacy, and interoperability, which are not taken into consideration during the design phase of IoT products [117]. The blockchain technology under rapid development emerges as a viable solution to addressing these challenges in decentralizing IoT systems.

Many challenges confronted by current IoT architectures may be addressed by blockchain. In [86], Kshetri presented a positive attitude towards strengthening IoT with blockchain and provided insights into how blockchain enhances IoT security, such as leveraging blockchain-based identity and access management systems or improving the overall security in supply chain networks. Cha et al. investigated data confidentiality and authentication based on blockchain [24]. Novo proposed to utilize blockchain as the access control layer for better security and privacy [114]. Alfonso et al. conducted a survey of the integration of blockchain and IoT, where different application domains are categorized, including *smart home*, *smart city* and *smart energy* [117].

Gossip protocol. A *gossip protocol* [33] is a procedure where a data item is routed to all members in a distributed network similar to epidemics spreading. Gossiping has been traditionally used for reliable information dissemination, but its applicability goes far beyond in distributed systems. Uber implemented a gossip protocol variation called *SWIM* [95] to allow independent workers to discover each other. Cassandra [90] used a gossip protocol for peer discovery and metadata propagation. *Docker*'s multi-host networking [34] employed a gossip protocol to exchange overlay network

information. Hyperledger fabric [9] implemented a gossip data dissemination protocol to ensure data integrity and consistency among different roles of nodes.

Kermarrec et al. [79] provided the general organization of a gossip protocol and discussed one of its most successful applications for *dissemination*, which is achieved by letting peers forward messages to each other. Eugster et al. [42] elaborated the gossiping dissemination process with three parameters: i) the number of messages stored in a node's local cache, ii) the number of selected peers for message forwarding, and iii) the upper bound of times that a message is forwarded. The Shuffle protocol in [52] is designed to disseminate information among a collection of wireless devices in a mesh network, but it only considers a synchronous model where the transmission duration among peers is constant. Andrew et al. [8] improved this model by taking into account the dynamics of a real network and employed exponential distribution and hyperexponential to simulate various transmission durations among peer nodes. In this study, we use gossip to realize robust message dissemination from sensor networks to blockchain networks and conduct experiments in real distributed environments.

Censorship resistance. Censorship resistance in IoT data communication is made possible by the decentralization and immutability nature of blockchain network. The study in [117] pointed out that the decentralization of IoT on top of blockchain is censorship resistant because *inside* the blockchain network, there is no controller and entities only trust the quality of the cryptographic algorithms that govern the operations. Obviously, the censorship problem still exists in the components of the blockchain network that communicate with *external* devices. Hence, we provide a formal definition of "censorship resistance" in blockchain-based IoT and propose an effective solution.

Table 2.1 Key Notations Related to the Censorship Resistance Protocol

Notation	Represent for
L_{alive}	the list maintaining live nodes in blockchain network
$sensor_{ID}$	the unique ID of a sensor
δt_{sensor}	the time period when a sensor sends data
δt_{seed}	the time period when seed nodes process data
δt_{dif}	the time period when data diffusion is completed in a synchronous network
$\mathcal{Z}$	a physical zone including sensors and gateway nodes
$\mathcal{Z}_{\mathcal{ID}}$	the unique ID of a zone $\mathcal{Z}$
d	the data collected by a sensor
l	the number of sensors in a certain zone $\mathcal{Z}$
k	the number of selected neighbor nodes in gossip protocol
n	the number of data items collected during δt_{sensor}
n'	the number of data items during B_{seed}
s	the number of seed nodes
c	command/instruction sent to actuators from blockchain network
σ	signature from the message sender
γ	the local cache size for gossip protocol
$\mathcal{N}$	the number of full nodes in blockchain network
C	the local cache on a peer node for gossiping
BUF	the buffer that a full node uses to receive data from sensors
ts	time stamp
$SEED$	message from sensors to seed nodes
B_{seed}	a constructed batch of $SEEDs$ maintained on seed nodes
$node_{ID}$	the unique ID of a full node in blockchain network
GM	gossip message from seed nodes to all full nodes

2.3 Problem Formulation

In this section, we formulate the problem and describe security requirements. The notations are provided in Table 2.1 for the convenience of reference.

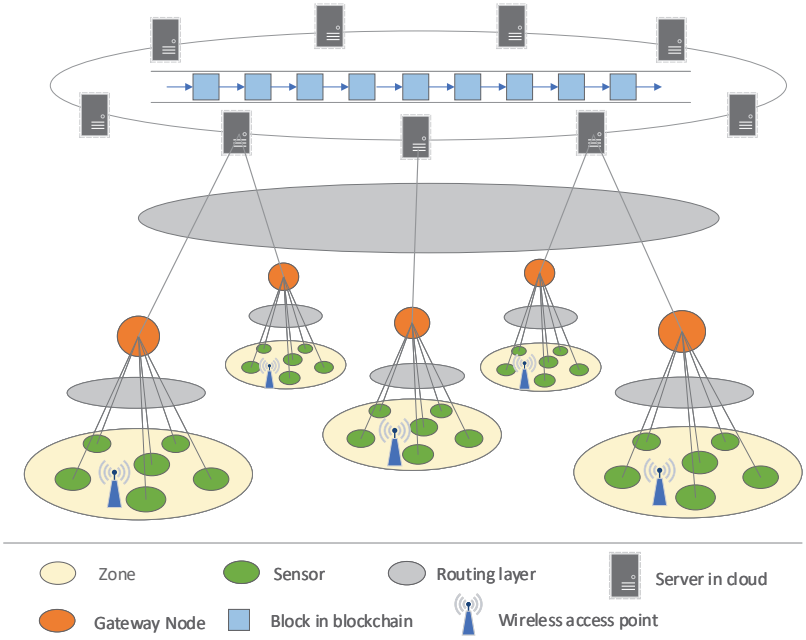


Figure 2.3 Blockchain-based IoT management system model.

Figure 2.3 illustrates the current blockchain-based IoT management model. Typically, in a blockchain-based IoT management system, multiple sensors are deployed in a certain area (e.g., a power plant) for data collection (e.g., temperature measurements). The collected data are sent to the nearest gateway node and forwarded by routers through a wireless network to a server in the blockchain network, which starts to execute the consensus protocol and replicate the data across all participating servers (also known as "peers" or "replicas"). Such consistent data items stored in blocks are appended to the blockchain as "transactions".

To investigate the security issues in blockchain-based IoT, we first provide the following definitions.

Definition 1 (Consensus)**.** *A consensus protocol has the following properties:*

- *Termination: Each participating peer outputs something locally within a limited amount of time.*
- *Agreement: All honest peers in the network agree on the same value.*
- *Validity: If all honest peers receive the same value v, then the agreed result should be equal to v.*

From this definition, we know that consensus only considers ledger replication, while disregarding how inputs are received from and outcomes are delivered to external clients. External clients are not full nodes of the distributed ledger, and thus have to rely on some servers to relay. As such, existing architectures assume "trusted" relay, which is vulnerable in practice as the relay server could either be hacked or simply be malicious. To realize the properties of "reliable message delivery" and "transparency" of a distributed ledger, we provide the following definition:

Definition 2 (Censorship Resistance)**.** *Consider a sequence of data items* $(d_1, d_2, ..., d_n)$ *sent from an IoT network to a blockchain network. The system is censorship resistant if it meets the following two conditions:*

- *The ledger records a permutation of the vector without any data loss.*
- *The corresponding actuator in the IoT network is guaranteed to eventually receive the value of* $y = \mathcal{F}(d_1, \ldots, d_n)$*, which is also stored in the ledger, where* $\mathcal{F}$ *is a pre-defined processing function.*

We now introduce the security issues in the current blockchain-based IoT model.

Security against entry point censorship. A malicious or hacked node[3] relaying messages from the sensor network to the blockchain network may act arbitrarily, e.g., drop messages, infinitely delay messages, or modify message contents. Meanwhile, even correct data is disseminated to the blockchain network, it may get lost during the process of reaching a consensus among all peers. We define the security in these two cases as follows:

- An adversary $\mathcal{A}$ corrupts the gateway node g in a zone $\mathcal{Z}$ including $sensor_1, ...,$ $sensor_l$. The message sent from the sensor network to g is denoted as $m = (d_1, ..., d_n)$. We consider a bad event B_1 as follows: (1) the number of data that forwarding from g to blockchain network is less than n; (2) there is no data forwarding from g to blockchain network since g blocks all the messages; (3) some data items in message m are modified before sending to blockchain network.

[3]Generally, a cluster with a master-slave architecture is constructed for the gateway node to tolerate crash fault, but it still acts as a single node since only the master node is responsible for providing services. Our proposed solution tolerates both crash fault and byzantine fault.

- An adversary $\mathcal{A}$ corrupts f nodes in the blockchain network to execute a consensus protocol. The message sent to the blockchain network from the sensor network is denoted as $m = (d_1, ..., d_n)$. We consider a bad event B_2 as follows: (1) not all nodes in the blockchain network update m to their ledgers; (2) all nodes update m to their ledgers but on some nodes, the number of data items in m is less than n, i.e., $|m| < n$; (3) all nodes update m to their ledgers and on all nodes $|m| = n$, but the data items in m are out of order.

The security of our proposed protocol requires that for every polynomial time adversary $\mathcal{A}$, the probability $Pr[B_1]$ and $Pr[B_2]$ is negligibly small.

Security against exit point censorship. When querying from the blockchain network, a corrupted node may perform malicious actions to actuators to cause a disaster. We define the security in this case as follows: An adversary $\mathcal{A}$ corrupts f nodes in the blockchain network. We consider a bad event B_3 as follows: when querying from the ledger, the instruction y is modified to y' and sent to actuators. Again, the security of our proposed protocol requires that for every polynomial time adversary $\mathcal{A}$, the probability $Pr[B_3]$ is negligibly small.

2.4 System Overview

To defend against potential threats of censorship, we augment the current blockchain network architecture and the consensus protocol. Figure 2.4 illustrates the data flow in the improved architecture.

2.4.1 Censorship Resistant Inbound Delivery

Our protocol carries out messages delivery as follows:

- The sensors disseminate data to $f + 1$ gateway nodes (referred to as "seed nodes"), which are full nodes, not just forwarding messages from the sensor network to the blockchain network.
- The seed nodes disseminate data to all other peer nodes in blockchain network through gossip-based diffusion mechanism.
- A leader node starts the byzantine consensus protocol to replicate the data;

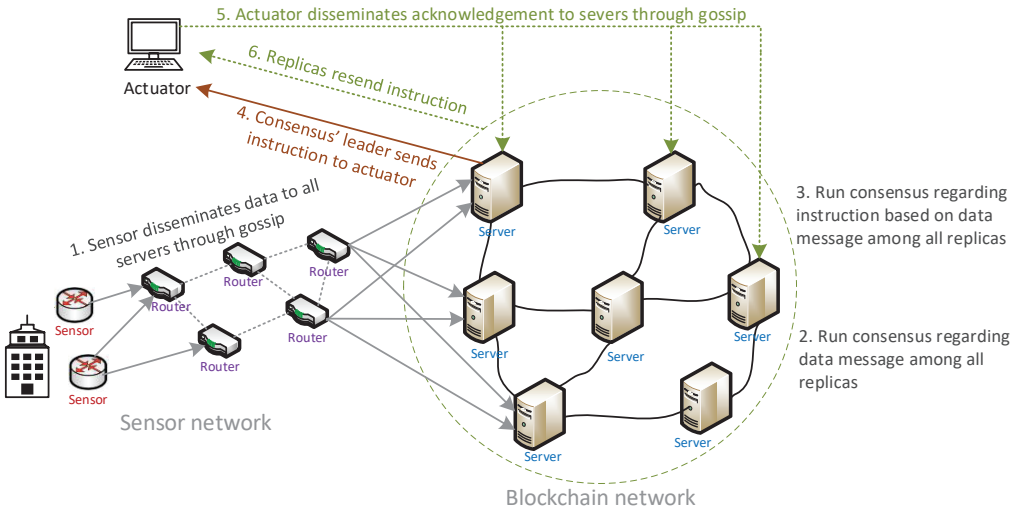


Figure 2.4 Data flow in the improved architecture to resist censorship.

- Each replica performs filtering validation to check if there is any data loss after consensus.

Specifically, to defend against censorship at the data entry point, we need to make sure that each sensor is connected to multiple servers instead of just one single gateway node. In the proposed scheme, the conventional single gateway node is replaced with multiple full nodes in the blockchain network (i.e., *seed nodes*), which perform not only the same function as the original gateway node but also a set of blockchain operations such as reaching consensus and updating ledger, hence eliminating the crash fault of the original gateway node. Furthermore, the number of seed nodes is at least $f + 1$ to tolerate the byzantine fault as discussed later. We propose to use a gossip-based protocol to achieve message diffusion among all peers for better robustness. Moreover, we enhance the underlying consensus protocol (e.g., BFT-SMaRt [16]) such that each honest participating peer further checks whether the block being replicated has dropped some data before updating the local ledger. If a sufficient number of peers observe data missing, the consensus process is restarted (e.g., a view change type of sub-protocol is triggered). We would like to point out that this enhancement is generic and could be applied to any BFT protocols.

After this round, the data is appended to the local ledger of each full node. To further enhance the protocol to support basic data analysis and instruction delivery, we propose the following:

2.4.2 Censorship Resistant Outbound Delivery

Our protocol carries out data processing and instruction delivery as follows:

- For a pre-defined processing function $\mathcal{F}$, another round of consensus is initiated using the outcome of $\mathcal{F}(\cdot)$ as the data to be replicated. Such agreement is the same as the third step in the aforementioned message delivery round, the consensus content is instruction instead of message.
- Once the value of $\mathcal{F}(\cdot)$ is written in each local ledger, the leader forwards the value of $\mathcal{F}(\cdot)$ with the peer nodes' signatures to the corresponding actuators/devices.
- Actuators receive an instruction and send feedback containing an acknowledgement to all full nodes.
- All replicas maintain a timer and wait for the acknowledgement for each sent instruction; if the acknowledgement is not received within a pre-defined time period, they all resend the instruction to actuators, Details are elaborated in Section 2.5.

After the data is written to the ledger, the nodes run the analysis program $\mathcal{F}$ that is pre-defined and deployed in a smart contract (an example about $\mathcal{F}$ is provided in Section 2.5), and use the output y of $\mathcal{F}$ as input to run another consensus protocol. At the end of this consensus protocol, there are a sufficient number of signatures on the same y, and an honest leader node forwards the output y together with the signatures to the actuator. The actuator simply broadcasts an acknowledgement to all servers if it receives instructions from the leader server and successfully verifies their signatures. The peer nodes wait for a pre-defined period of time, and if there is no feedback from the corresponding actuator, they all send value y to the actuator. This feedback mechanism achieves an opportunistic efficiency: when the leader is honest, only one single message is sent to the actuator and this single message contains the signatures

of most peers (specifically $2f + 1$, where f is the largest number of malicious nodes) in the blockchain network; only when the leader node drops the outgoing instruction, the other peers jointly inform the actuator. Although we assume that the diffusion from the sensor network to the blockchain network (and vice versa) be completed in a fixed amount of time (i.e., as a synchronous network), the network that connects the peers (i.e., the blockchain network itself) could be partially asynchronous and we are still able to deploy such consensus protocols as PBFT [23].

2.5 Protocol Design

In this section, we provide a detailed description of protocol design to realize censorship resistance on top of blockchain-based IoT. The main idea is to further decentralized data entry and exit point. For simplicity, we denote the number of all replicas (full nodes) as $\mathcal{N}$, $|\mathcal{N}| = 3f + 1$[4], where f is the maximum number of faulty nodes. In addition, BFT-SMaRt, which implements a modular state machine replication protocol atop a Byzantine consensus algorithm [134], is used as an example for the underlying consensus protocol to explain our protocol process.

2.5.1 Handling Censorship of Single Entry

The data is sent from the sensor network to the blockchain network, which is referred to as "inbound flow". Figure 2.5 illustrates the normal operation on inbound flow in our protocol, as detailed in the following three phases:

Phase I for message diffusion. The goal in this phase is to ensure that data from each sensor be quickly diffused to every full node in a more robust way instead of relying on the single gateway node, so that when the consensus protocol is invoked, all full nodes have a copy of the sensor data in place.

[4]M. Castro, B. Liskov et al. [23] proved that a minimum of $3f + 1$ replicas/peers are needed to tolerate at most f faulty/byzantine replicas/peers.

Blockchain network is essentially a decentralized point-to-point network. To broadcast data from each sensor to the blockchain network in an efficient and robust way, we propose a gossip-based diffusion protocol as detailed below:

a) Initialization. Initially, all peers in the blockchain network execute the discovery service and message exchange to maintain a dynamic list $L_{alive}^{node_{ID}}$ of live peers they can connect to. Such a list contains IP address, port, and public key of peer nodes.

b) Data multicasts to seed nodes. We call those nodes that participate in blockchain network and also initially being connected by sensors to receive data as *seed nodes*. Sensors periodically send collected data $d_1^{sensor_j}, \ldots, d_n^{sensor_j}, j = 1, \ldots, l$ to s seed nodes, the message is in the form of $SEED = \langle Z_{ID}, sensor_{ID}, d, ts \rangle_{\sigma_{sensor_{ID}}}$.

c) Processing on seed nodes. Each seed node maintains a local buffer BUF for received data from sensors and always check the signature validity before caching sensor data to BUF. Every δt_{seed}, each seed node accumulates $SEEDs$ in BUF as a batch B_{seed}, and counts the number of data items in B_{seed} as $\tilde{n}$, which is used to check *data loss* later. $SEED_i$ in this batch is sorted sequentially according to ts. It is practical to ensure that $\delta t_{seed} < \delta t_{sensor}$. Therefore, the unpredictable network impact is eliminated and all these s seed nodes have the same state of B_{seed} ready.

d) Gossip diffusion. We consider a synchronous network where message diffusion can be completed in δt_{dif}. In order to reduce the complexity incurred by message mixing and ensure the same number of data items on each seed node, we set $\delta t_{dif} < \delta t_{seed}$. The seed nodes then disseminate the gossip message $GM = \langle node_{ID}, B_{seed}, \tilde{n}, ts \rangle_{\sigma_{node_{ID}}}$ to all peers in the blockchain network through *gossip-based diffusion algorithm*, as shown in Algorithm 1. Note that as the system tolerates up to f malicious nodes, the number of seed nodes $k \geq f + 1$ ensures that malicious actions be detected and hence not updated to the ledger.

The gossip-based diffusion algorithm in Algorithm 1 is divided into three steps:

a) Topology construction. Practically, each peer node in the blockchain network maintains a list of its direct and indirect *neighbor nodes* whose information is stored in $L_{alive}^{node_{ID}}$. An *overlay network* is formed with virtual links from each peer node to its neighbor nodes. It is worth mentioning that the gossip-based algorithm can also be utilized to maintain the gossip network itself: peer nodes periodically exchange and update $L_{alive}^{node_{ID}}$ with each other so the network topology can be dynamically maintained when some nodes leave or join.

Algorithm 1 Gossip-based Message Diffusion

Input: $GM = \langle node_{ID}, B_{seed}, \tilde{n}, ts \rangle_{\sigma_{node_{ID}}}$

Output: $true$ or $false$

1: **Initialization**: the number s of seed nodes; the number k of selected neighbor nodes; the maximum number t of times a message can be forwarded; the information about neighbor nodes stored in $L_{alive}^{current\_node_{ID}}$; the local cache $C^{node_{ID}}[\gamma]$ of size γ as a buffer for received messages

2: $m, \sigma_{node_{ID}} \leftarrow parse(GM)$, where $m = \langle node_{ID}, B_{seed}, \tilde{n}, ts \rangle$;

3: $r \leftarrow verify(\sigma_{node_{ID}}, node_{pk} \leftarrow L_{alive}^{current\_node_{ID}}, m)$;

4: **if** $r == true$ **then**

5: **if** $C^{node_{ID}}$ *does not contain* m **then**

6: forward m with ts and $\sigma_{current_{node}}$ to k neighbor nodes selected from $L_{alive}^{current_{node_{ID}}}$;

7: add quadruple ($key = m$, $counter = 1$, $flag = false$, $integrity = false$) to $C^{node_{ID}}$;

8: **while** $C^{node_{ID}}.size \geq \gamma$ **do**

9: remove those items whose $counter \geq t$ and $integrity$ is $true$;

10: **else**

11: **if** ($counter$ for m) $< t$ **then**

12: forward m with ts and $\sigma_{current_{node}}$ to k neighbor nodes selected from $L_{alive}^{current_{node_{ID}}}$;

13: $counter++$ for the item whose $key == m$;

14: **else**

15: **if** $flag$ for $m == false$ **then**

16: update $flag$ for m as $true$; **return** true;

17: **else**

18: **return** $false$;

b) Peer selection. Each peer node in the blockchain network is initialized with a number of gossip parameters during the topology construction step, including: a local cache $C^{node_{ID}}$ with size γ; the maximum number t of times a message can be forwarded; and the number k of neighbor nodes a peer node selects to forward messages each time. Among these parameters, k plays a critical role in diffusion efficiency since the value of k and the selected nodes affect the dissemination speed. Previous study shows that constructing a gossip-based topology on top of a *peer sampling service* [71] can ensure a uniformly random selection of peers.

c) Data dissemination. Those k uniformly and randomly selected nodes are called *passive nodes* and the node starting to send messages is an *active node*. Their interaction is described as follows:

   i) Each seed node acts as an active node, and uniformly and randomly selects k nodes as passive nodes from its local cache $C^{node_{ID}}$. A gossip message GM is retrieved from BUF.

   ii) Each active node sends message GM to all of its corresponding passive nodes.

   iii) All passive nodes act as active nodes to repeat this process by randomly and uniformly choosing k nodes from their local cache and forwarding message GM.

   iv) Each node (including seed nodes) maintains a set of quadruples $(key, counter, flag, integrity)$ in the local cache $C^{node_{ID}}$, where the key is the gossip message GM, and *counter* is to count how many times GM is forwarded by the node. For efficiency, the hash value of GM is computed to quickly determine if the current node has already forwarded such a message. If the received message is already in its cache, we increase the counter; otherwise, the new item is added to the cache. If $counter \geq t$, we stop forwarding this message and consider it as *stable* by setting $flag$ to be $true$. In a synchronous network, all nodes are able to reach a *stable* status within a reasonable time period δt_{dif}. The *integrity* is set to be $false$ by default, indicating whether or not this message is checked in the later *data loss* phase. If the total number of messages exceeds the cache size γ on the node, we remove those items whose $counter \geq t$ and *integrity* is $true$.

In a gossip network with $\mathcal{N}$ nodes, a message sent from a seed node is relayed by a set of randomly selected k nodes in every round and is expected to reach all other nodes after θ rounds, i.e., $\sum_{i=0}^{\theta} k^i = \mathcal{N}$, and hence $\theta = \lceil log_k(1 - \mathcal{N}(1-k)) - 1 \rceil$. Especially when $k = 2$, the process turns to be a binary tree and the complexity of rounds becomes $O(log\mathcal{N})$.

Phase II for byzantine consensus. We augment the consensus protocol to support censorship checking during the consensus process. Note that all full nodes have the input data in place after the *message diffusion* phase.

The consensus' leader node firstly sends a *PROPOSE* message containing $\mathcal{B}_{seed}$ to other replicas. All other replicas receive the *PROPOSE* message and then check the validity of the proposed batch and the sender's leadership: if both are true, then register $\mathcal{B}_{seed}$ and send a *WRITE* message containing a cryptographic hash of the batch, denoted as $H(\mathcal{B}_{seed})$, to all other replicas. If a replica receives $\lceil \frac{|R|+f+1}{2} \rceil$

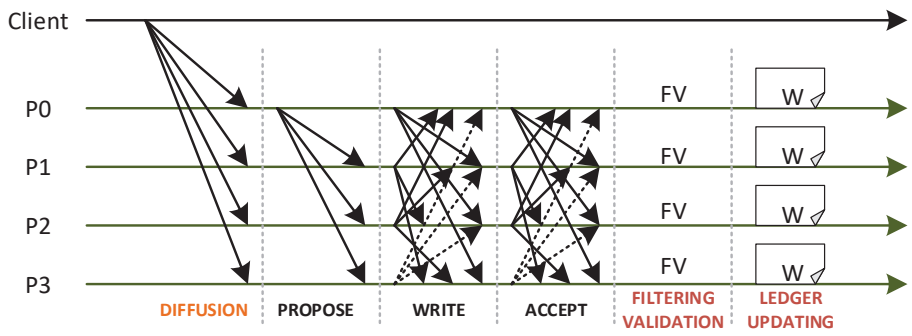


Figure 2.5 Message pattern for dealing with the single entry.

WRITE messages with an identical hash, it sends an *ACCEPT* message containing this hash to all other replicas.

Phase III for data loss check. If a replica receives $\lceil \frac{|R|+f+1}{2} \rceil$ *ACCEPT* messages for the same hash, it performs *FILTERING VALIDATION (FV)* to detect data loss by comparing the number $\tilde{n}$ of messages in *PROPOSE* with the number of messages received from the sensors, i.e., n. If filtering validation passes, it appends the new data $\mathcal{B}_{seed}$ to the ledger; otherwise, a *view change*[5] may take place to elect a new leader and all replicas are required to converge to the same consensus execution. More details can be found in [134].

We would like to point out that it might be more efficient to perform *FV* right after *PROPOSE* to avoid *WRITE* and *ACCEPT* if filtering was noticed. However, this would require modifying the original consensus, e.g., the BFT-SMaRt protocol. Our design only involves adding a few phases after the protocol finishes and hence facilitates quick implementation and convenient deployment.

[5]If all nodes executing the consensus protocol have the same leader, they are in the same view. Views are numbered consecutively, and the leader of a view is a replica p such that $p = v \ mod \ \mathcal{N}$, where v is the view number. Hence, when the leader is considered to fail, a view change is carried out by setting the new leader to be $p = (v + 1) \ mod \ \mathcal{N}$ to continue consensus execution.

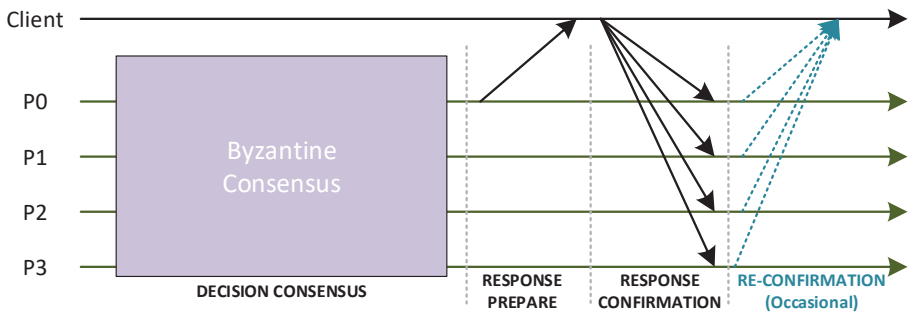


Figure 2.6 Message pattern for dealing with the single exit.

2.5.2 Handling Censorship of Single Exit

In many blockchain-based IoT scenarios, sensors collect and send data to the ledger and meanwhile actuators receive instructions for further actions. These instructions could be the outcomes of some data analysis procedures applied to the collected data. Thus, we also need to ensure that i) the instruction from the blockchain network to the sensor network (referred to as "outbound flow") is "legitimate", i.e., the instruction is the consensus of the participants rather than a single node, and ii) the instruction is successfully delivered to the intended actuator. Figure 2.6 shows the normal operation on the outbound flow, as detailed in the following two phases:

Phase I for decision consensus. After the data batch is written to the ledger, each honest node executes a data analysis program in the form of $y = \mathcal{F}(GM)$. Program $\mathcal{F}$ is typically known *a priori* as it is application-specific and may vary in different scenarios. Algorithm 2 gives a simple example of how function $\mathcal{F}$ works: the input is the data batch that updated to ledger which contains sensor data from different zones, by calculating the average value of sensors from the same zone and comparing with the threshold $\mathcal{T}$, corresponding instruction from a pre-defined set $\mathcal{Y}$ is returned, otherwise no action is needed by returning $\perp$. The decision consensus phase executes the same steps as the aforementioned Byzantine consensus phase. The only difference is the content to be agreed on, which is the output y instead of data batch $\mathcal{B}_{seed}$. We split it into two rounds for different consensus contents, because in some cases such

Algorithm 2 An Example of Analysis Program $\mathcal{F}$

Input: $\mathcal{B}_{seed}$

Output: $y/\perp$

Initialization: the instruction set $\mathcal{Y}$; a temperature threshold: $\mathcal{T} \in \mathbb{Z}$; the sum of temperatures collected by all sensors in a specific zone: $\mathcal{S}_{Z_{ID}} \leftarrow 0$; the counter that keeps track of the number of times the sensor data is counted: $t_{Z_{ID}} \leftarrow 0$;
$SEED_i, i \in [n] \leftarrow parse(\mathcal{B}_{seed})$;
for $SEED_i$ **do**
    $\langle Z_{ID}, d \rangle \leftarrow parse(SEED_i)$;
    if Z_{ID} exists **then**
        $\mathcal{S}_{Z_{ID}} \leftarrow \mathcal{S}_{Z_{ID}} + d$;
        $t_{Z_{ID}} + +$;
    else
        $new(\langle Z_{ID}, \mathcal{S}_{Z_{ID}}, t_{Z_{ID}} \rangle)$;
        $\mathcal{S}_{Z_{ID}} \leftarrow d$;
        $t_{Z_{ID}} \leftarrow 1$;
for all Z_{ID} **do**
    $\langle Z_{ID}, AVG_{Z_{ID}} \rangle \leftarrow \langle Z_{ID}, \frac{\mathcal{S}_{Z_{ID}}}{t_{Z_{ID}}} \rangle$;
    if $AVG_{Z_{ID}} > \mathcal{T}$ **then return** $y \leftarrow \mathcal{Y}$;
return $\perp$;

as the data collection system, only the data needs to be recorded in the ledger, while in other cases, it may need both. At the end of the decision consensus phase, the output based on the sensor data is updated to the ledger as well. Then, the execution of *RESPONSE* phase is triggered.

Phase II for response. The response phase includes prepare, confirmation, and occasional re-confirmation. In *RESPONSE PREPARE*, the honest consensus' leader forwards command/instruction y to the actuator if it is the agreed outcome, which means that it has collected sufficient[6] signatures from peer nodes on the same y. Once the actuator receives an instruction and verifies all signatures, it enters into the *RESPONSE CONFIRMATION* phase, in which the actuator simply broadcasts

[6]For crash fault tolerance, sufficient refers to at least $f + 1$ peer nodes, while for byzantine fault tolerance, sufficient refers to at least $2f + 1$ peer nodes, proof can be found in [23].

the signed acknowledgement ack_σ to all servers: this is done the same way as in the DIFFUSION phase via gossip. Other non-leader replicas wait for ack_σ after updating y to the ledger. If they do not receive an acknowledgement within a predefined time period, they all resend y to the actuator by themselves, also via gossip.

2.5.3 Security Analysis

Security against entry point censorship. We first analyze the security against entry point censorship, i.e., data flow from the sensor network to the blockchain network. Existing blockchain-based IoT management systems rely on a single gateway node to relay messages. We propose to replace the single gateway server with $f + 1$ full nodes (i.e., "seed nodes") in the blockchain network, which not only participate in the blockchain operations but also act as conventional gateway nodes for message forwarding. As defined in our security model, the security against entry point censorship requires that the probability of bad events B_1 and B_2 is negligibly small. Specifically, B_1 includes three cases where the malicious gateway node may i) drop some messages, ii) infinitely delay messages without relaying to the blockchain network, and iii) modify message contents and send modified messages to a subset of peer nodes. B_2 also includes three cases: i) some peer nodes cannot receive messages from the sensor network, therefore failing to update the ledger; ii) all nodes update messages to the ledger successfully, but the number of messages on some nodes is less than what have been sent from the sensor network; iii) all nodes successfully update a correct number of messages in their ledger, but in a different order.

We now discuss how our protocol prevents the above cases in a synchronous network between sensors and full nodes. For consensus, we can still handle a partially synchronous network among the full nodes. To tolerate byzantine fault, the total number of peer nodes in the blockchain network is expected to be at least $3f + 1$, where f is the maximum number of faulty nodes. For Case i) in B_1, having $f + 1$

seed nodes instead of one single gateway node ensures that at least one honest node be selected. With a more robust gossip-based diffusion mechanism, the honest node relays a correct number of messages to all peer nodes in the blockchain network. For Case ii) in B_1, since at most f nodes can infinitely delay the messages, having $f + 1$ seed nodes ensures that at least one honest node be selected and then gossip messages to all other peer nodes. For Case iii) in B_1, even though malicious nodes may modify messages, the signature verification ensures that the modified data be rejected and never updated to the ledger. For Cases i) and ii) in B_2, we propose an augmented consensus protocol, where a filtering validation phase, which is added to the regular BFT consensus protocol, checks data loss before updating the ledger by comparing the number of data messages after consensus with the number of data items received in the gossip-based diffusion process. If equal, they are updated to the ledger; otherwise, a view change is triggered to reach consensus again. After at most f rounds, data is correctly updated to the ledger since an honest node is selected to be the leader. Case iii) in B_2 is addressed by the byzantine consensus with proved security. In sum, our proposed protocol satisfies the security requirements on *entry point censorship resistance*.

Security against exit point censorship. We now analyze the security against exit point censorship, i.e., data flow from the blockchain network to the sensor network. Unlike permissionless blockchain such as Ethereum, where all participants maintain one public chain, in permissioned blockchain such as hyperledger fabric, each peer node maintains a copy of the ledger, which can be modified locally. The blockchain-based IoT management system not only stores data in the ledger, but also is expected to make proper decisions and communicate with corresponding actuators. Generally, only one "leader" node is responsible for sending instructions to actuators for efficiency since actuators are typically equipped with limited computing power. If

the "leader" node is compromised, it may: i) send modified instructions to actuators; and ii) infinitely delay the instructions.

Our protocol solves the first ping the aggregated signatures from most (specifically, at least $2f + 1$) of the peer nodes: if failed, no action is taken by the actuator; otherwise, a correct instruction is executed. In the second problem, the actuator does not receive any instruction from the malicious leader. In both of these cases, if the actuator takes no action, no acknowledgement is sent back to the peer nodes, and then an occasional *re-confirmation* is triggered, where all peer nodes jointly inform the actuator.

An alternative solution is to initiate a view change to select a new leader. Since there are at most f malicious nodes, eventually an honest node is selected to send instructions to the actuator. However, this process may be repeated for f times in the worst case and is not suitable for time-critical IoT scenarios. In our protocol, the leader is honest in most cases, so *re-confirmation* is rare. Even if it happens, all nodes just need to send *once* and the actuator is guaranteed to receive the instruction even though it may need to communicate with more peer nodes instead of only the "leader" as in the alternative solution. Hence, our protocol satisfies the security requirement on exit point censorship resistance.

2.5.4 Reducing Verification Complexity

In blockchain-based IoT systems, some processes have similar properties, e.g., sensors send collected data to the blockchain network, and peer nodes send instruction back to actuators. More specifically, in the former, $sensor_1, ..., sensor_k$ possessing their public keys $pk_{s_1}, pk_{s_2}, \ldots, pk_{s_k}$ send messages $m_1, m_2, \ldots, m_k$ and corresponding signatures $\sigma_{s_1}, \sigma_{s_2}, \ldots, \sigma_{s_k}$ to nodes in the blockchain network and get verified. Similarly, in the latter, all non-leader replicas $r_1, r_2, \ldots, r_n$ who owns public keys $pk_{r_1}, pk_{r_2}, \ldots, pk_{r_n}$ send instructions/commands $c_1, c_2, \ldots, c_n$ together with signatures $\sigma_{r_1}, \sigma_{r_2}, \ldots, \sigma_{r_n}$ to

the leader for forwarding. In these cases, it is important to exploit a more efficient and secure way to verify signatures. By leveraging the work in [18], we propose to leverage the modified BLS multi-signature aggregation scheme (referred to as *public key aggregation*) to reduce the communication and verification complexity, based on the following considerations:

- Multiple sensors of the same type are typically deployed in a region for improved fault tolerance and sensing accuracy, and some of them may very likely collect identical measurements. Using a signature aggregation mechanism is efficient but may suffer from *rouge public-key attack* [18].
- Prepending the sensor's public key to the collected data before signing defends against the above attack, but would not be able to make full use of the advantages brought by aggregating identical messages.

The adoption of *public-key aggregation* defends against *rouge public-key attack* while achieving efficiency. We take the second scenario as an example to explain the application of this scheme, which contains the following components:

- A bilinear pairing $e : G_0 \times G_1 \rightarrow G_T$. The pairing is efficiently computable, and non-degenerated. All three groups have prime order q. Let g_0 and g_1 be the generator of G_0 and G_1, respectively.
- Two hash functions $H_0 : \mathcal{M} \rightarrow G_0; H_1 : G_1^n \rightarrow R^n$ where $R := 1, 2, ..., 2^{128}$ and $1 \leq n \leq \tilde{N}$, These two hash functions are treated as random oracle in the security analysis.

With these components, the scheme works as follows:

- $KenGen()$: choose a random $\alpha \xleftarrow{R} Z_q$ and set $h \leftarrow g_1^{\alpha} \in G_1$, output $pk := (h)$ and $sk := (\alpha)$.
- $Sign(sk, c_i)$: sign command c_i and output $\sigma_i \leftarrow H_0(c_i)^{\alpha} \in G_0$, where $i = \{1, 2, ..., n\}$ denotes different replicas in the blockchain network.
- $Aggregate((pk_{r_1}, \sigma_1), ..., (pk_{r_n}, \sigma_n))$:
  - $compute : (t_1, ..., t_n) \leftarrow H_1(pk_{r_1}, ..., pk_{r_n}) \in R^n$.
  - $output : \sigma \leftarrow \sigma_1^{t_1} \cdots \sigma_n^{t_n} \in G_0$.
- $Verify(pk_{r_1}, ..., pk_{r_n}, c_i, \sigma_i)$: to verify the multi-signature σ_i on c_i, we do

– *compute* : $(t_1, ..., t_n) \leftarrow H_1(pk_{r_1}, ..., pk_{r_n}) \in R^n$.
– *compute* : $apk \leftarrow pk_1^{t_1} \cdots pk_n^{t_n} \in G_1$.
– if $e(g_1, \sigma_i) = e(apk, H_0(c_i))$, output "accept"; otherwise, output "reject".

The above scheme is for verifying multiple signatures on one message. If messages keep flowing, it is more efficient to verify as a batch. Specifically, consider a triple (m_i, σ_i, apk_i) for $i = 1, 2, ..., b$, where b is the number of messages in one batch. If m_i are all distinct, then:

- *compute* : $\tilde{\sigma} \leftarrow \sigma_1 \cdots \sigma_b \in G_1$.
- Accept all b tuples as valid iff $e(g_1, \tilde{\sigma}) = e(apk_1, H_0(m_1)) \cdots e(apk_b, H_0(m_b))$.

If there are identical messages in m_i, then:

- *obtain* : $\rho_1, ..., \rho_b \xleftarrow{R} 1, 2, ..., 2^{64}$.
- *compute* : $\tilde{\sigma} \leftarrow \sigma_1^{\rho_1} \cdots \sigma_b^{\rho_b} \in G_1$.
- Accept all b tuples as valid iff $e(g_1, \tilde{\sigma}) = e(apk_1^{\rho_1}, H_0(m_1)) \cdots e(apk_b^{\rho_b}, H_0(m_b))$.

Thus, verifying b messages requires only $b + 1$ instead of $2b$ pairings if verifying one at a time. Hence, such a batch-based mechanism can further improve verification efficiency.

2.6 Implementation and Evaluation

To shed some light on the behavior of how the gossip-based diffusion mechanism resist the censorship of the single entry point (for the single exit problem, it also relies on gossip-based message dissemination to send instructions to actuators/devices, we do not repeat the redundant evaluation of such process), we implement the gossip-based message dissemination process[7] with following settings:

[7]The implementation source code of the gossip-based message dissemination can be found at *https://github.com/Blockchain-World/gossip-based-diffusion.git*

- We create a gossip network testbed using 21 Google Cloud virtual machine (VM) instances[8], each of which is equipped with 3.75 GB memory and 1 vCPU and has JRE 1.8.0_181 installed on Ubuntu 16.04.
- The bandwidth between nodes in the same zone is 1.96 Gbps, while across different zones, it is at least 700 Mbps. Since the message size is relatively small, these bandwidths make the data transfer time negligible compared with the protocol execution time.
- We gossip 10 data items from one client to the other 20 peer nodes. Each data item is a short string of about six bytes, and each peer node has a local cache large enough to buffer 10 data items.

Figure 2.7(a)-2.7(f) plots the number of received data items on each peer node during a certain period of time, where a red line represents the maximum number of tolerated malicious nodes while a blue line represents the behavior of honest nodes. These results illustrate censorship resistance where the gossip-based diffusion mechanism guarantees that all data items sent from the client be delivered to all honest peer nodes in the blockchain network.

We calculate the average time cost and standard deviation for gossiping one data item from the "sensor network" (client node) to all other peer nodes in the blockchain network, as shown in Figure 2.8 and Table 2.2. We have the following observations and explanations:

- The variation in the average time cost with different selected neighbor nodes in each gossip round is caused by the non-uniformity when randomly selecting neighbor nodes. In our implementation, we use Java *Math.random()* with pseudo-randomness to generate k random numbers as the selected indices of neighbor nodes. In some rounds, it is possible that the same target nodes are selected by different peer nodes and some nodes are not covered until after a few rounds and eventually receive messages. This process could be optimized using *peer sampling service* as mentioned in Section 2.5, but we do not focus on gossip optimization in this study.
- The standard deviation is relatively large. The measured time cost includes message transmission, neighbor selection, and consecutive writing I/O operations

[8]In our gossip testbed, one VM acts as a client, while the other 20 VMs act as peer nodes. These VMs are located in different zones: $node_1$ to $node_8$ reside in the same zone (*us-east1-b*), $node_9$ to $node_{15}$ reside in *u-east4-c*, and $node_{16}$ to $node_{20}$ reside in *us-central1-c*.

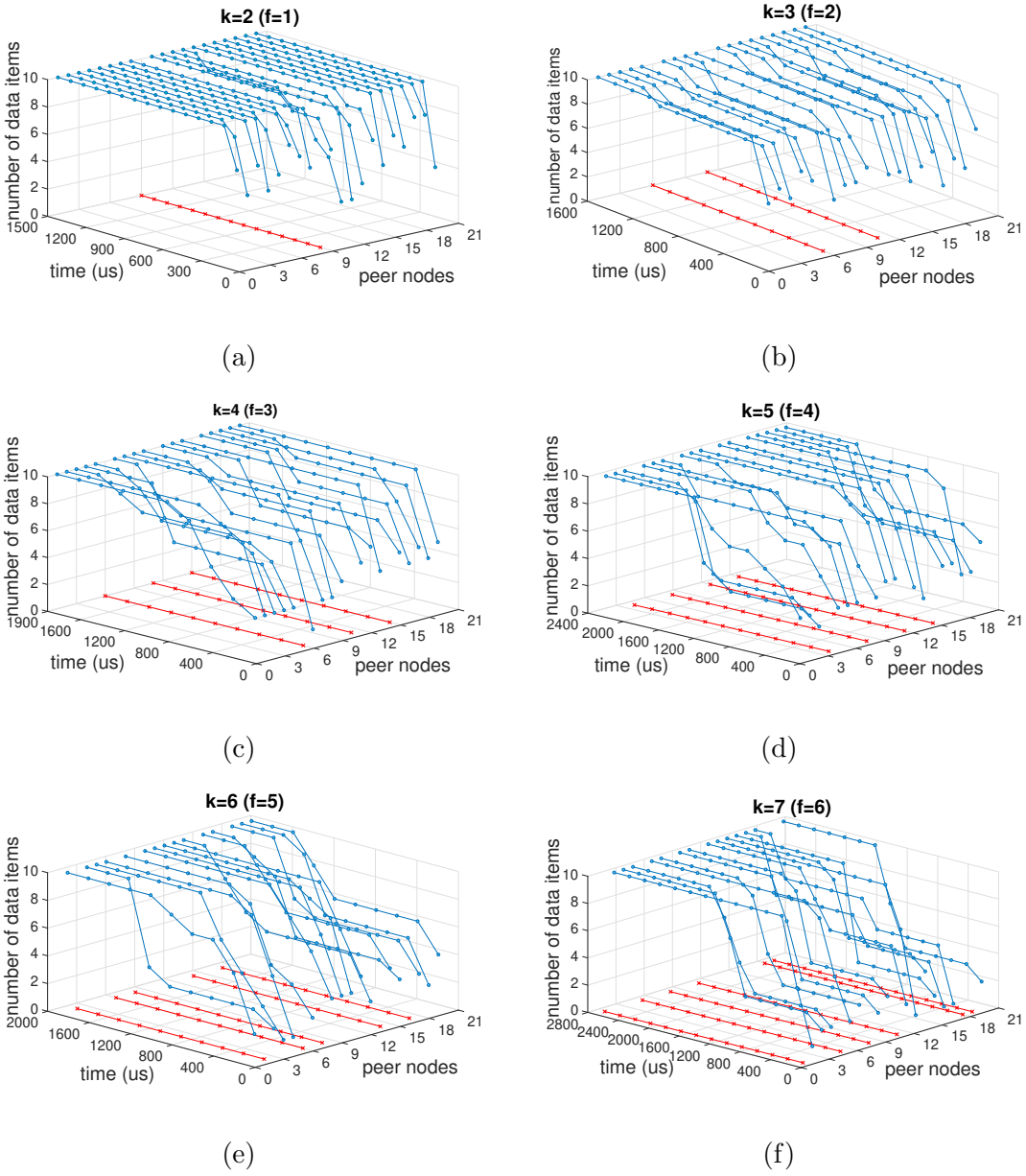


Figure 2.7 The gossiping diffusion time for various selected neighbor nodes.

for recording timestamp. The network condition and the randomness in peer selection affect the time cost of message delivery, i.e., some nodes may receive messages sooner than others.

- There exists a slowly increasing trend in the average time cost, as more selected neighbor nodes result in more traffic in the network. In some rounds, the randomly selected neighbor nodes may have already been selected in previous rounds, also contributing to the increase in the average time cost.

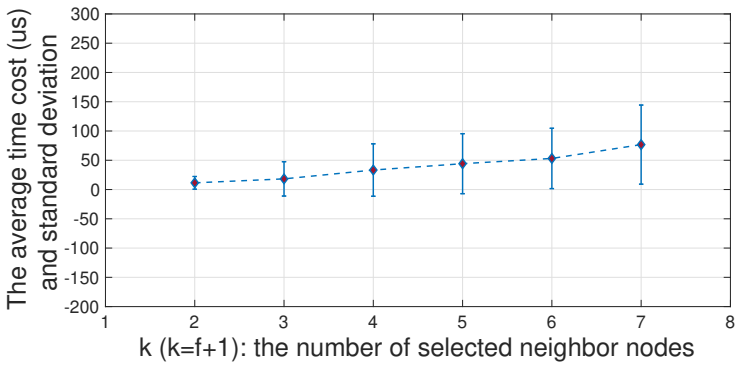


Figure 2.8 The average time cost and standard deviation for gossiping protocol.

Table 2.2 The Average Time Costs for the Gossiping Protocol

k (k=f+1)	The Average Time Cost (ms)	Standard Deviation
k=2 (f=1)	11.43	10.86
k=3 (f=2)	18.21	29.33
k=4 (f=3)	33.32	44.79
k=5 (f=4)	44.16	51.24
k=6 (f=5)	53.13	51.54
k=7 (f=6)	76.73	67.58

2.7 Summary

In this chapter, we present the design to defend against the censorship problem between blockchain-based decentralized information management system and external network, i.e., IoT sensor network as a concrete application scenario. For data flows from a sensor network to a blockchain network, we overcome potential censorship on a gateway node by employing gossip-based diffusion protocol to achieve guaranteed message delivery. Moreover, we improve the consensus protocol by checking data loss before writing to the ledger and replicating process outcome to facilitate opportunistic outcome delivery. Finally, we leverage the cryptographic tool of public

key aggregation to reduce communication and verification complexity, and analyze the security of our protocol. We implement the proposed gossip-based diffusion algorithm and illustrate message delivery with censorship resistance in the presence of faulty nodes. The results show that the protocol is efficient. It is worth pointing out that the proposed design can be used in any general application (besides IoT setting) scenario when blockchain-based management systems interact with external networks.

CHAPTER 3

BLOCKCHAIN-BASED CYBER SECURITY MANAGEMENT

This chapter presents the design of blockchain-based information management system in the specific cyber security management setting [60].

3.1 Introduction

The importance of enterprise-level cyber security management cannot be overstated. For example, when Bob who defends an enterprise network becomes aware of a new Advanced Persistent Threat (APT) attack that has been active in the wild for a while, he needs to investigate whether or not his network has been a victim of the APT and if so, what the damages are. Indeed, the standard defined in ISO/IEC 27035 includes a five-phase incident management process: prepare, identify, assess, respond and learn [69]. In order to be effective, such standardization must be supported by tools [7]. However, existing tools mainly focus on vulnerability management, incident management, or security information and event management [56,73]. Despite these tools, many routine cyber defense activities are still a *manual* process [7], meaning that defenders cannot respond to cyber events rapidly. Moreover, the manual process is often conducted in isolation because enterprises rarely share cyber intelligence with each other. This is true despite the extensive body of work highlighting the importance of sharing cyber threat intelligence [2, 7, 20, 32, 126, 127]. For example, learning the attacks that have successfully penetrated into enterprise A would undoubtedly help enterprise B defend its network against the same and similar attacks. This observation demonstrates that the problem of effective cyber security management (CSM) remains largely open.

One main challenge encountered when designing an effective CSM system is how to ensure its *robustness*. Specifically, a centralized CSM is vulnerable to single-point-

of-failure, which often outweighs its advantage in terms of performance especially because CSM itself is clearly an important target of the attackers. In addition to the vulnerability to attacks, a centralized CSM is not available when there is system crash. This robustness against cyber attacks and crashes naturally motivates the use of distributed or decentralized CSM. However, leveraging the classical distributed database that tolerates crashes for CSM purposes is still problematic because this technique requires to trusting all participants (i.e., *crash fault tolerant* (CFT) [1,129]) but not resilient under attacks (i.e., Byzantine faults). This highlights the importance of incorporating Byzantine Fault-Tolerance (BFT) into a decentralized CSM. Another important matter is to automate CSM itself because traditionally CSM has been done in a manual fashion [7], which incurs delays and is tedious and error-prone. In addition, a CSM system should offer other properties such as *accountability*, meaning that both providers and consumers of cyber intelligence should be held accountable for their activities. Overall, we make a significant step towards formalizing CSM by defining three kinds of CSM functions in relation to cyber intelligence sharing, whereby the participating defenders share and leverage cyber intelligence for their CSM purposes.

3.2 Related Work

Our CSM functions take certain threat intelligence as input. Prior studies in threat intelligence sharing can be divided in to four categories: (i) characterizing the opportunities and challenges [64,126]; (ii) understanding the legal and regulatory matters [4, 133]; (iii) exploring standardization and principles [32, 49]; and (iv) developing tools [20, 32, 124]. Our study is closely related to the preceding (iv), but ours is unique since we formulate the problem of *robust* and *automated* CSM and present a *blockchain-based* design and implementation. It is worth mentioning that CSM is different from cyber forensics [100]. This is because forensics is oriented toward

certain details, such as attack attribution and criminal investigation, which are not critical in all cases of cyber attacks, and can inhibit the efficient response to active attacks. In contrast, CSM prioritizes efficiency, which is closer to the aim of incident response.

3.3 CSM Model, Data Structures and Functions

3.3.1 Terminology

A cyber defender, Bob, manages a set of entities, which are broadly defined to accommodate computers and other objects of cybersecurity significance. As shown in Figure 3.1, we make the distinction between *external* entities (i.e., those not managed by Bob but which may be managed by another defender, Cindy) and *internal* entities (i.e., those managed by Bob); this external vs. internal distinction is from a specific defender's point of view: in this case, Bob's. An entity can be in one of three states: *normal*, *victim* or *attacker*. A *victim* entity is one that has been compromised by an external or internal *attacker* entity; an *attacker* entity is one that exhibits malicious behavior; and a *normal* entity is one that is neither a victim nor an attacker entity. A normal entity can become a victim entity when it is attacked by an external or internal attacker entity, and a victim entity can elevate to an attacker entity.

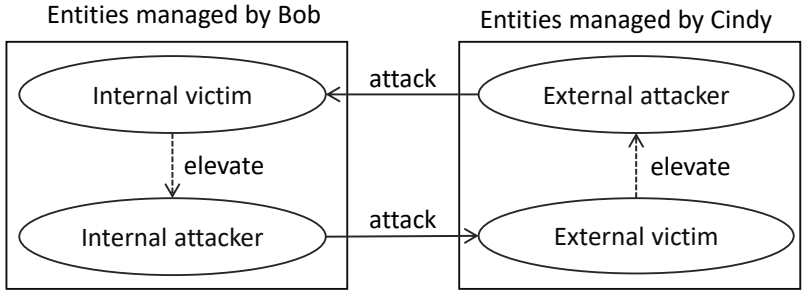


Figure 3.1 External vs. internal attacker and external vs. internal victim.

3.3.2 CSM Model

In the CSM model, a defender Bob, or more precisely his CSM App (CSMA), leverages some input cyber intelligence to identify victim and attacker entities, where the input intelligence may be (i) shared by other defenders or (ii) discovered by some cyber defense tools used by the defender Bob. In what follows, we describe five kinds of cyber intelligence, three classes of CSM functions, and a general data structure designed to facilitate those CSM functions.

Input cyber intelligence. As illustrated in Figure 3.2, we consider five kinds of input cyber intelligence, which are prefixed by 'I-'.

(I-1A) Intelligence that points to some *external attackers*, possibly accompanied by the time window during which an external attacker is active.

(I-1B) Intelligence that points to some *internal attackers*, which may have attacked some external victims and been detected by another defender, or some internal victims and been detected by some cyber defense tools used by Bob.

(I-2A) Intelligence that points to some *external victims*, which have been attacked by some internal or external attackers.

(I-2B) Intelligence that points to some *internal victims*, which have been attacked by some internal or external attackers. The intelligence may be collected, for example, by the leakage of data specific to the victim (e.g., social security numbers or passwords) or by a cyber defense tool (e.g., intrusion detection system or anti-malware tool).

(I-3) Intelligence that points to some *new* defense capabilities, such as methods for detecting previously undetected attacks (e.g., 0-day attacks).

An overview of three classes of CSM functions. As depicted in Figure 3.2, Bob's CSMA takes as input some cyber intelligence and the relevant cyber data, uses the CSM functions (specified below) to identify other internal or external attackers/victims, and outputs the resulting intelligence. Bob may choose to share this output with another defender, say Cindy, about his internal or external attackers/victims (i.e., input cyber intelligence I-1A, I-1B, I-2A, I-2B from Cindy's point of view). To be specific, we define three classes of CSM functions, as shown

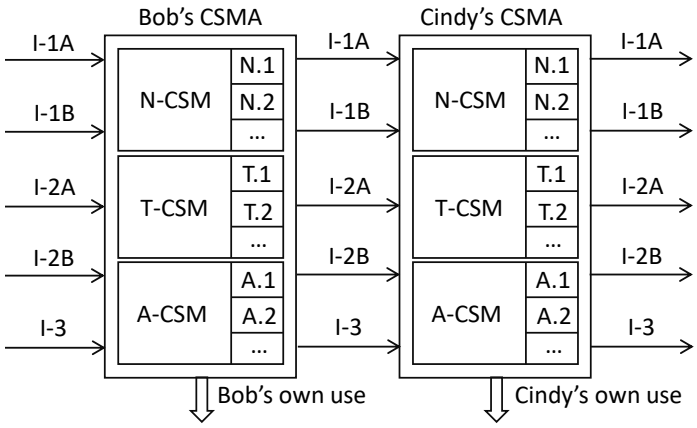


Figure 3.2 CSM model with input cyber intelligence and CSM functions.

in Figure 3.2: (i) Network-centric CSM (N-CSM), which leverages network-related data and cyber intelligence for CSM purposes; (ii) Tools-centric CSM (T-CSM), which leverages data collected from cyber defense tools and cyber intelligence for CSM purposes; and (iii) Application-centric CSM (A-CSM), which leverages application-specific data and cyber intelligence for CSM purposes. Each class contains multiple CSM functions, and the core ideas of these functions are described below.

N-CSM functions are centered at examining the input cyber intelligence against network traffic data, which may be collected at a gateway between the external network (e.g., the Internet) and the internal network (e.g., an enterprise network). Network traffic data can be represented by IP packets and TCP/UDP flows, which incur different costs on storage. We define the following three N-CSM functions.

(N.1) This function is designed to identify internal victims of some external attackers, which are given as the input cyber intelligence (i.e., input I-1A). Specifically, at time t', Bob is given cyber intelligence that an *external attacker*, identifiable by its IP address, attacker_IP, was active at some point in time interval $[t_1, t_2]$ where $t' \geq t_2$. Bob needs to identify his internal systems that may have been compromised by the external attacker in time interval $[t_1, t_2]$.

(N.2) This function aims to identify external attackers that may have caused the compromise of some internal victims (i.e., input I-2B). At time t', Bob is given cyber intelligence that an *internal victim*, identifiable by victim_IP, was attacked at some point in time interval $[t_1, t_2]$ where $t' \geq t_2$. Bob needs to identify the external IP addresses that contacted victim_IP in time interval $[t_1, t_2]$.

(N.3) This function is designed to identify potential secondary victims that may have been attacked before, during or after the known compromise of some other *internal victim* (i.e., input I-2B and/or I-1B). Specifically, at time t', Bob is given cyber intelligence that an internal victim IP address, identifiable by its IP address, victim_IP, was attacked at some point in time interval $[t_1, t_2]$ where $t' \geq t_2$. Then, Bob needs to identify the other victims that were contacted by the potential attackers that may have compromised the given victim_IP during time interval $[t_1, t_2]$.

T-CSM functions are centered at cyber defense tools, such as Network-based Intrusion Detection Systems (NIDSs), Host-based Intrusion Detection Systems (HIDSs), and anti-malware systems. These tools often output alerts as indicators of malicious or suspicious activities. We define the following three T-CSM functions.

(T.1) This function identifies the attack path(s) via which a known *internal victim* was compromised (i.e., input I-2B). Specifically, at time t', Bob is given cyber intelligence that an internal victim, say victim_IP, was compromised at some point during the time interval $[t_1, t_2]$ where $t' \geq t_2$. Then, Bob needs to identify the attack path(s) that may have been leveraged to compromise victim_IP.

(T.2) This function identifies victims of zero-day attacks by leveraging a new defense capability (i.e., input I-3). Specifically, at time t', Bob is given cyber intelligence on a new detection method (e.g., signature) for detecting a previously unknown zero-day attack. Then, Bob needs to identify the internal victims that were attacked according to the new detection method during a past time interval $[t_1, t_2]$, $t' \geq t_2$.

(T.3) This function is designed to identify the cascading damage caused by a given attacker (i.e., input I-1A or I-1B). Specifically, at time t', Bob is given cyber intelligence that a malicious external or internal entity was active at some point in time interval $[t_1, t_2]$ where $t' \geq t_2$. Then, Bob needs to identify the entities that were directly or recursively accessed by the malicious entity during time interval $[t_1, t_2]$.

A-CSM functions are centered at specific applications that are often exploited to wage attacks, such as drive-by downloads via web browsers and spear-phishing via email. As examples, we consider the following three A-CSM functions.

(A.1) This function identifies secondary *internal victims* (e.g., browsers or email clients) that have been targeted by the same attack that succeeded against a known compromised entity (i.e., input I-2B). Specifically, at time t', Bob is

given cyber intelligence that an internal entity (i.e., browser or email user) was compromised at some point in time interval $[t_1, t_2]$ where $t' \geq t_2$. Then, Bob needs to identify the other internal victim entities (i.e., browsers or email users) that communicated with any of the external attacker (i.e., URLs or email users) that compromised the internal victim during time interval $[t_1, t_2]$.

(A.2) This function identifies *internal victims* (e.g., browsers or email users) of an external attacker (namely input I-1A). Specifically, at time t', Bob is given an external attacker (i.e., URL or email address) that was active at some point in time interval $[t_1, t_2]$ where $t' \geq t_2$. Then, Bob needs to identify the other internal victims (i.e., browsers or email users) that may be compromised because they communicated with the external attacker during time interval $[t_1, t_2]$.

(A.3) This function identifies *internal victims* that may be impacted by known attacks against an external victim (e.g., spoofed URL or email address, namely input I-2A). At time t', Bob is given cyber intelligence that an external victim (i.e., URL or email address) was spoofed to wage attacks at some point in time interval $[t_1, t_2]$ where $t' \geq t_2$. Then, Bob needs to identify the external attackers (i.e., URLs or email addresses) that spoofed the given external victim during time interval $[t_1, t_2]$ and the internal victims (i.e., browsers or email addresses) that communicated with the external attacker during time interval $[t_1, t_2]$.

A general CSM data structure. In order to realize the CSM functions, appropriate data representations are needed. We propose a general data structure, known as an *Annotated Graph Time Series Representation* (AGTSR), by dividing the time horizon into $T + 1$ *time windows* at some resolution (e.g., hour or day). In order to reduce the number of notations, we make the following convention: the default use of t, t_1, t_2 refers to specific points in time; we also use the term *time window* t, t_1, t_2 to refer to the t-th, t_1-th, and t_2-th time window, where $0 \leq t, t_1, t_2 \leq T$.

For time window t, we use $G(t) = (V(t), E(t), A(t))$ to represent the relevant cyber activities for CSM purposes, where $V(t)$ is the vertex set with each vertex representing an entity (e.g., IP address, computer or device), $E(t)$ is the arc set with each arc representing some communication activity, and $A(t)$ is the annotation set such that $A(t) = \{A_{uv}(t) : (u, v) \in V(t) \times V(t)\}$ with $A_{uv}(t)$ being a set of annotations associated to $(u, v) \in V(t) \times V(t)$ and $A_{uv}(t).\mathsf{count}$ denotes the number of IP packets or TCP/UDP flows along an arc (u, v) in time window t. That is,

$A_{uv}(t).\mathsf{count} = 0$ means $(u, v) \notin E(t)$ and $A_{uv}(t).\mathsf{count} > 0$ means $(u, v) \in E(t)$, and count is the number of IP packets or TCP/UDP flows from entity (e.g., IP address) u to entity v in time window t. The meanings of annotations in $A_{uv}(t)$ are specific to the class of CSM functions, and will be elaborated below. In principle, $G(t)$ may be stored as an adjacency matrix or list; for simplicity, we will focus on the adjacency matrix representation and $A_{uv}(t)$ can be seen as an extension of the standard adjacency matrix. Our model can support division of a network into subnets with both intra- and inter-subnet communications. We can achieve this by extending $G(t) = (V(t), E(t), A(t))$ of time window t to $G^m(t) = (V^m(t), E^m(t), A^m(t))$, where $V^m(t) \subseteq V(t)$ are the nodes belong to a subnet and formulate a partition of $V(t)$, $(u, v) \in E^m(t)$ means $u, v \in V^m(t)$, and A^m_{uv} means $u, v \in V^m(t)$. There are also arcs $E^{m,m'}(t) = \{(u, v) : u \in V^m(t), v \in V^{m'}(t)\}$. The cybersecurity meanings of these notations are specific to the CSM functions in question and thus elaborated later.

Besides, we use $\max_{t\in[t_1,t_2]} |V(t)|$ to denote the maximum number of entities (e.g., computers, IP addresses, or browsers) during a time window in between time window t_1 and time window t_2, namely $\max_{t\in[t_1,t_2]} |V(t)| = \max\{|V(t_1)|, |V(t_1 + 1)|, \ldots, |V(t_2)|\}$ with $0 \le t_1 \le t_2 \le T$. Similarly, we define that $\max_{t\in[t_1,t_2]} |V^m(t)| = \max\{|V^m(t_1)|, |V^m(t_1 + 1)|, \ldots, |V^m(t_2)|\}$.

3.3.3 CSM Data Structures

For N-CSM, AGTSR can accommodate network communications such that a node $u \in V(t)$ represents a computer, and an arc $(u, v) \in E(t)$ represents the communications between nodes u and v initiated by u. In N-CSM, we are often concerned with border communications, meaning the communications between the internal entities and the external entities. In this case, $V(t)$ is partitioned into V^{external} and V^{internal}, where V^{external} is the set of external entities (e.g., IP addresses) and V^{internal} is the set of internal entities. For time window t, there is a $G(t) = (V(t), E(t), A(t))$ as

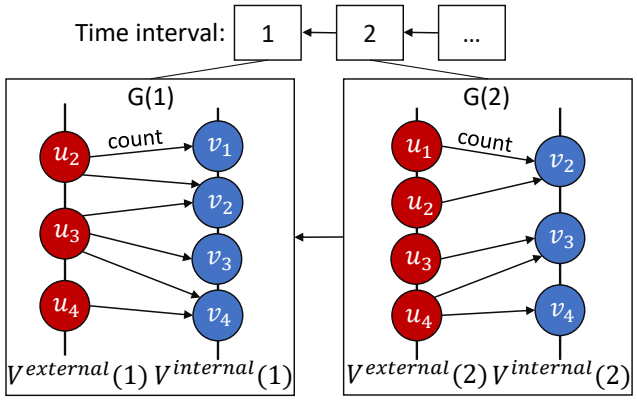


Figure 3.3 Data structure for N-CSM.

defined above. Figure 3.3 illustrates $G(1), G(2), \ldots$; for example, we have $u_2, u_3, u_4 \in V^{\text{external}}(1)$ and $v_1, v_2, v_3, v_4 \in V^{\text{internal}}(1)$ where count is only illustrated for $(u_2, v_1) \in E(1)$ for a better visual effect.

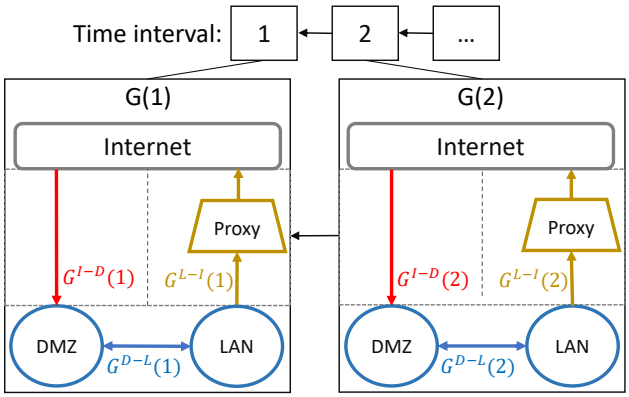


Figure 3.4 Data structure for T-CSM.

For T-CSM, Figure 3.4 shows an example network to clearly convey the ideas. The network has three disjoint subnets: the Internet (i.e., the external subnet), the demilitarized zone for external-facing servers (DMZ), and the local area network (LAN). This suggests that Bob can use (i) an AGTSR to represent the interactions between the Internet and the DMZ, or $G^{\mathtt{I-D}}(t)$ for short; (ii) an AGTSR to represent the interactions between the LAN and the Internet, or $G^{\mathtt{L-I}}(t)$ for short; and (iii) an AGTSR to represent the interactions within the DMZ itself, within the LAN itself

and between the border of the DMZ and the LAN, or $G^{\mathtt{D-L}}(t)$ for short. Note that $V(t) = V^{\mathtt{I-D}}(t) \cup V^{\mathtt{L-I}}(t) \cup V^{\mathtt{D-L}}(t)$. In T-CSM, the annotation of an arc is a list of alerts (i.e., $A_{uv}(t) = \{\mathsf{alerts}\}$). These alerts are triggered by the traffic across each arc, which often corresponds to a routing path rather than a physical link.

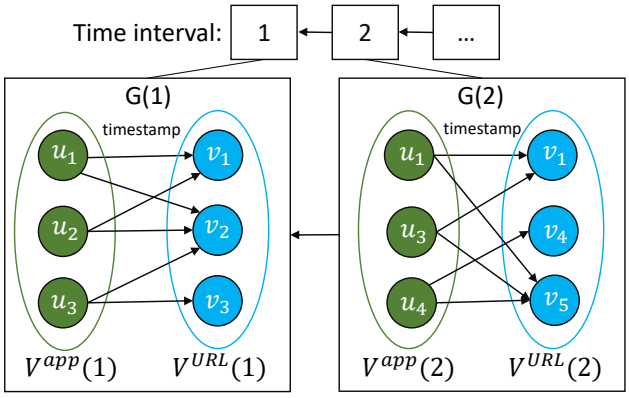


Figure 3.5 Data structure for A-CSM.

For A-CSM, we use the example of web applications, but the discussion can be adapted to accommodate other applications, e.g., email systems. In this example, browsers (or their IP addresses) are internal entities and URLs are external entities. As illustrated in Figure 3.5, we have $G(t) = (V(t), E(t), A(t))$, where $V(t) = V^{\mathtt{app}}(t) \cup V^{\mathtt{URL}}(t)$, $E(t)$ is the arc set such that arc $(u, v) \in E(t)$ means browser $u \in V^{\mathtt{app}}(t)$ visited URL $v \in V^{\mathtt{URL}}(t)$ in time window t, each arc $(u, v) \in E(t)$ is annotated with a $\mathsf{timestamp} \in A_{uv}(t)$, where a value of -1 means $(u, v) \notin E(t)$.

3.3.4 CSM Functions

Here we introduce the CSM functions. Note that all these functions are implemented and deployed in the smart contract (i.e., chaincode) to facilitate CSM.

Algorithm 3 realizes N-CSM function N.1 by identifying victims given an attacker. The algorithm considers each time window within a given time interval $[t_1, t_2]$, checking each arc originating from the attacker to identify the entities that were accessed by the attacker. The query returns a list of all such entities. The algorithm

Algorithm 3 N-CSM Function N.1 (Identifying Victims)

Input: attacker, T, $G(t) = (V(t) = V^{\text{internal}}(t) \cup V^{\text{external}}(t), E(t), A(t))$ for $t \in [t_1, t_2]$ with $0 \leq t_1 \leq t_2 \leq T$

Output: $\langle t, \text{victims}(t) \rangle$ for $t \in [t_1, t_2]$

```
1: for t ∈ [t_1, t_2] do
2:     if attacker ∈ V^external(t) then
3:         victims(t) ← ∅;
4:         for v ∈ V^internal(t) do                    ▷ Check victims
5:             if A_{attacker,v}(t).count > 0 then
6:                 victims(t) ← victims(t) ∪ {v};
7: return victims(t) for t ∈ [t_1, t_2];
```

Algorithm 4 N-CSM Function N.2 (Identifying Potential Attackers)

Input: victim_IP, T, $G(t) = (V(t) = V^{\text{internal}}(t) \cup V^{\text{external}}(t), E(t), A(t))$ for $t \in [t_1, t_2]$ with $0 \leq t_1 \leq t_2 \leq T$

Output: $\langle t, \text{attackers}(t) \rangle$ for $t \in [t_1, t_2]$

```
1: for t ∈ [t_1, t_2] do
2:     if victim_IP ∈ V^internal(t) then
3:         attackers(t) ← ∅;
4:         for a ∈ V^external(t) do                    ▷ Check attackers
5:             if (a, victim_IP) ∈ E(t) then
6:                 attackers(t) ← attackers(t) ∪ {a};
7: return attackers(t) for t ∈ [t_1, t_2];
```

has a time complexity $\mathcal{O}((t_2 - t_1 + 1) \cdot \max_{t \in [t_1, t_2]} |V^{\text{internal}}(t)|)$, where $(t_2 - t_1 + 1)$ indicates the number of time windows that are considered.

Algorithm 4 realizes N-CSM function N.2 by identifying potential attackers based on their communications to a given victim. The algorithm considers each time window within the time interval $[t_1, t_2]$, checking which attacker entities tried to access the given victim entity. The algorithm has a time complexity $\mathcal{O}((t_2 - t_1 + 1) \cdot \max_{t \in [t_1, t_2]} |V^{\text{external}}(t)|)$.

Algorithm 5 realizes N-CSM function N.3 by identifying secondary victims of the attacker that compromised the input victim. The algorithm uses Algorithm 4

Algorithm 5 N-CSM Function N.3 (Identifying Extended Victims)

Input: victim_IP, T, $G(t) = (V(t) = V^{\text{internal}}(t) \cup V^{\text{external}}(t), E(t), A(t))$ for $t \in [t_1, t_2]$ with $0 \leq t_1 \leq t_2 \leq T$

Output: $\langle t, \text{potential\_victims}(t) \rangle$ for $t \in [t_1, t_2]$

1: **for** $t \in [t_1, t_2]$ **do**
2: potential_victims$(t) \leftarrow \emptyset$;
3: **if** victim_IP $\in V^{\text{internal}}(t)$ **then**
4: tmp_attackers $\leftarrow \emptyset$;
5: **for** $u \in V^{\text{external}}(t)$ **do**
6: **if** $A_{u,\text{victim\_IP}}(t)$.count > 0 **then**
7: tmp_attackers$(t) \leftarrow$ tmp_attackers$(t) \cup \{u\}$; ▷ u accessed victim_IP
8: **for** $u \in$ tmp_attackers(t) **do**
9: **for** $v \in V^{\text{internal}}(t)$ **do**
10: **if** $A_{u,v}(t)$.count > 0 **then**
11: potential_victims$(t) \leftarrow$ potential_victims$(t) \cup \{v\}$; ▷ u accessed v and may have compromised it
12: **return** potential_victims(t) for $t \in [t_1, t_2]$;

to compute the potential external attackers, which are then used to identify the other internal entities that may have been compromised by the potential attackers. The algorithm has a time complexity $\mathcal{O}((t_2 - t_1 + 1) \cdot \max_{t \in [t_1, t_2]} |V^{\text{internal}}(t)| \cdot \max_{t \in [t_1, t_2]} |V^{\text{external}}(t)|)$.

Algorithm 6 realizes T-CSM function T.1 by inferring the attack paths to the compromised internal entity (e.g., computer or IP address, namely input I-2B) in time interval $[t_1, t_2]$. The algorithm creates a tree of potential attackers from the given compromised internal entity. The tree grows according to the relevant network activities, and add new nodes when new attackers are identified. The resulting tree structure contains the target as the root, compromised internal entities as internal nodes, and all possible attackers as the leaves. Since the given compromised entity belongs to the internal LAN, the algorithm's search space originates in $G^{\mathtt{D}-\mathtt{L}}(t')$ and branches out within the network until all entities have been considered. Once

the relevant $G^{\mathtt{D-L}}(\cdot)$'s have been exhausted, the algorithm checks both $G^{\mathtt{I-D}}(\cdot)$ and $G^{\mathtt{L-I}}(\cdot)$ to identify potential external attackers. The algorithm has a time complexity $\mathcal{O}((t_2-t_1+1)\cdot((\max_{t\in[t_1,t_2]}|V^{\mathtt{D-L}}(t)|)^2+\max_{t\in[t_1,t_2]}|V^{\mathtt{I-D}}(t)|+\max_{t\in[t_1,t_2]}|V^{\mathtt{L-I}}(t)|)\cdot \max_{t\in[t_1,t_2]}|V(t)|)$.

Algorithm 7 realizes T-CSM function T.2 by retrospectively detecting victims of a zero-day attack during the past time windows prior to discovery of the zero-day attack (i.e. input I-3). The cyber intelligence may come in the form of an alert sequence from either an IDS' output or a previously unexplained anomaly. In either case, the defender needs to look at all previous IDS alerts to find matches. For this purpose, the algorithm traces back over the past time windows in between t_1 and t_2, by looking at each IDS alert in the set of arc annotations. The algorithm has a time complexity $\mathcal{O}((t_2-t_1+1)\cdot((\max_{t\in[t_1,t_2]}|V^{\mathtt{D-L}}(t)|)^2+\max_{t\in[t_1,t_2]}|V^{\mathtt{I-D}}(t)|+\max_{t\in[t_1,t_2]}|V^{\mathtt{L-I}}(t)|)\cdot\max_{t\in[t_1,t_2]}|V|(t))$.

Algorithm 8 realizes T-CSM function T.3 by identifying the cascading damage of a given attacker (i.e., input I-1A or I-1B). The algorithm determines which entities were targeted by the given attacker, either directly or recursively. The algorithm has a time complexity $\mathcal{O}(t_2-t_1+1\cdot\max_{t\in[t_1,t_2]}|V(t)|^2)$.

Algorithm 9 realizes A-CSM function A.1 by identifying suspicious internal applications (i.e., potentially compromised browsers). The input to the algorithm is a browser as an internal victim (i.e., input I-2B). The output is a set of compromised browsers (internal victims) that have accessed any URLs visited by the given compromised browser during time interval $[t_1,t_2]$. The time complexity $\mathcal{O}((t_2-t_1+1)\cdot\max_t|V^{\mathtt{app}}(t)|\cdot\max_t|V^{\mathtt{URL}}(t)|)$.

Algorithm 10 realizes A-CSM function A.2 by identifying victim browsers. The input to the algorithm is a known malicious URL (i.e., input I-1A). The output is the set of browsers (internal victims) that accessed the malicious URL during time interval $[t_1,t_2]$. The algorithm has a time complexity $\mathcal{O}((t_2-t_1+1)\cdot\max_t|V^{\mathtt{app}}(t)|)$,

Algorithm 6 T-CSM Function T.1 (Inferring Attack Paths)

Input: Victim_IP, T, $G(t) = (G^{\mathtt{I-D}}(t), G^{\mathtt{D-L}}(t), G^{\mathtt{L-I}}(t))$ for $t \in [t_1, t_2]$ with $0 \leq t_1 \leq t_2 \leq T$

Output: Attack_Paths $= (V^{AP}, E^{AP}, A^{AP})$

$V^{AP} \leftarrow \{\text{Victim\_IP}\}$; $E^{AP} \leftarrow \emptyset$; $A^{AP} \leftarrow \emptyset$;
for $t = t_2$ downto t_1 **do**
    Node_Queue $\leftarrow$ New FIFO;
    while Node_Queue is not empty **do** ▷ Conduct BFT
        for Vertex $v \in V^{AP}$ **do**
            Node_Queue.ENQUEUE(v);
        Searched_Nodes $\leftarrow \emptyset$; Current_Node $\leftarrow$ Node_Queue.DEQUEUE();
        if Current_Node $\in V^{\mathtt{D-L}}(t)$ **then**
            for Vertex $v \in V^{\mathtt{D-L}}(t)$ **do**
                if $A^{\mathtt{D-L}}_{v,\text{Current\_Node}}(t)$.alerts $\neq \emptyset$ **then**
                    if $v \notin V^{AP}$ **then**
                        $V^{AP} \leftarrow V^{AP} \cup \{v\}$;
                        for Vertex $v' \in V^{AP}$ **do**
                            $A^{AP}_{v,v'}$.alerts $\leftarrow \emptyset$; $A^{AP}_{v',v}$.alerts $\leftarrow \emptyset$;
                    $A^{AP}_{v,\text{Current\_Node}}$.alerts $\leftarrow A^{AP}_{v,\text{Current\_Node}}$.alerts $\cup A^{\mathtt{D-L}}_{v,\text{Current\_Node}}(t)$.alerts;
                    if $v \notin$ Searched_Nodes $\cup$ Node_Queue **then**
                        Node_Queue.ENQUEUE(v);
        Searched_Nodes $\leftarrow$ Searched_Nodes $\cup$ Current_Node;
    for sub $\in \{\mathtt{I-D}, \mathtt{L-I}\}$ **do**
        for Vertex $v \in V^{AP}$ **do**
            if $v \in V^{\mathtt{sub}}(t)$ **then**
                for Vertex $v' \in V^{\mathtt{sub}}(t)$ **do**
                    if $A^{\mathtt{sub}}_{v',v}(t)$.alerts $\neq \emptyset$ **then**
                        if $v' \notin V^{AP}$ **then**
                            $V^{AP} \leftarrow V^{AP} \cup \{v'\}$;
                            for Vertex $\hat{v} \in V^{AP}$ **do**
                                $A^{AP}_{\hat{v},v'}$.alerts $\leftarrow \emptyset$; $A^{AP}_{v',\hat{v}}$.alerts $\leftarrow \emptyset$;
                        $A^{AP}_{v',v}$.alerts $\leftarrow A^{AP}_{v',v}$.alerts $\cup A^{\mathtt{sub}}_{v',v}(t)$.alerts;
return Attack_Paths $= (V^{AP}, E^{AP}, A^{AP})$;

where $\max_t |V^{\mathtt{app}}(t)|$ is the maximum number of browsers that accessed some URLs during a time window.

Algorithm 7 T-CSM Function T.2 (Identifying Victims of Zero-Day Attacks)

Input: Attack_Signature, T, $G(t) = \{G^{\mathtt{I-D}}(t), G^{\mathtt{D-L}}(t), G^{L-I}(t)\}$ for $t \in [t_1, t_2]$ with $0 \leq t_1 \leq t_2 \leq T$

Output: $\langle t, \mathsf{Matches(t)} \rangle$ where $t \in [t_1, t_2]$

1: Matches $\leftarrow$ New linked list of empty lists
2: **for** $t \in [t_1, t_2]$ **do**
3: **for** $\mathtt{sub} \in \{\mathtt{I-D}(t), \mathtt{D-L}(t), L-I(t)\}$ **do**
4: **for** Vertex $v \in V^{\mathtt{sub}}(t)$ **do**
5: **for** Vertex $v' \in V^{\mathtt{sub}}(t)$ **do**
6: **if** Attack_Signature $\subseteq A^{\mathtt{sub}}_{v,v'}(t)$.alerts **then**
7: Matches$(t) \leftarrow$ Matches$(t) \cup \{(v, v')\}$;
8: **return** Matches(t) for $t \in [t_1, t_2]$;

Algorithm 3.3.4 realizes A-CSM function A.3 by identifying victim browsers of spoofed (e.g., typo-squatted) URLs. The input to the algorithm is an abused URL url_id (i.e., input I-2A), The output includes the set of possibly spoofed URLs, denoted by spoofed_urls(t), and the set of potential victim browsers, denoted by victim_apps(t), for $t \in [t_1, t_2]$. Lines 3-7 of Algorithm 3.3.4 find each of the spoofed URLs $v \in V^{\mathtt{URL}}(t)$ that has an *edit distance* smaller than a given threshold $\tau\_distance$, where edit distance is computed using Algorithm 12 (which is a variant of the Levenshtein distance algorithm). Lines 1-2 of Algorithm 12 extracts the domain names from url_1 and url_2. Lines 3-4 create the array of components (i.e., the components separated by the '.' character) for each of domain names. Lines 5-6 determine the maximum and minimum lengths of the component arrays respectively. Lines 8-16 compute the edit distance for each components of $\mathsf{component_1}$ and $\mathsf{component_2}$ starting from the last component (usually top-level domain names such as '.com' or '.net' are the last components) and sum the edit distances of individual components to get the total_distance between $\mathsf{domain_2}$ and $\mathsf{domain_2}$. For example, consider $\mathsf{url_1}$ = "`mail.google.com/contact.php`" and $\mathsf{url_2}$ = "`mali.g00gle.com/home.php`." We define their edit distance as the edit distance between $\mathsf{domain_1}$ = `mail.google.com`

Algorithm 8 T-CSM Function T.3 (Identifying Cascading Damage)

Input: Attacker_IP, T, $G(t) = (G^{\mathtt{I-D}}(t), G^{\mathtt{D-L}}(t), G^{L-I}(t))$ for $t \in [t_1, t_2]$ with $0 \leq t_1 \leq t_2 \leq T$

Output: Damage_Graph $= (V^{DG}, E^{DG})$

$V^{DG} \leftarrow \{\text{Attacker\_IP}\}$; $E^{DG} \leftarrow \emptyset$; $A^{DG} \leftarrow \emptyset$;
for $t \in [t_1, t_2]$ **do**
  for sub $\in \{\mathtt{I-D}, \mathtt{L-I}\}$ **do** ▷ Check arcs which come from the Internet
    if Attacker_IP $\in V^{\mathtt{sub}}(t)$ **then**
      for Vertex $v \in V^{\mathtt{sub}}(t)$ **do**
        if $A^{\mathtt{sub}}_{\text{Attacker\_IP},v}(t)$.alerts $\neq \emptyset$ **then**
          if $v \notin$ V$^{\mathsf{DG}}$ **then**
            $V^{DG} \leftarrow V^{DG} \cup \{v\}$;
            for Vertex $v' \in V^{DG}$ **do** ▷ Initialize empty arcs to existing nodes
              $A^{DG}_{v,v'}$.alerts $\leftarrow \emptyset$; $A^{DG}_{v',v}$.alerts $\leftarrow \emptyset$;
          $A^{DG}_{\text{Attacker\_IP},v}$.alerts $\leftarrow A^{DG}_{\text{Attacker\_IP},v}$.alerts $\cup A^{\mathtt{sub}}_{\text{Attacker\_IP},v}(t)$.alerts;
  for Vertex $v \in V^{DG}$ **do**
    Node_Queue.ENQUEUE(v);
  Searched_Nodes $\leftarrow \emptyset$;
  while Node_Queue is not empty **do** ▷ Conduct BFT
    Current_Node $\leftarrow$ Node_Queue.DEQUEUE();
    if Current_Node $\in V^{\mathtt{D-L}}(t)$ **then**
      for Vertex $v \in V^{\mathtt{D-L}}(t)$ **do**
        if $A^{\mathtt{D-L}}_{\text{Current\_Node},v}(t)$.alerts $\neq \emptyset$ **then**
          if $v \notin$ V$^{\mathsf{DG}}$ **then**
            $V^{DG} \leftarrow V^{DG} \cup \{v\}$
            for Vertex $v' \in V^{DG}$ **do**
              $A^{DG}_{v,v'}$.alerts $\leftarrow \emptyset$;
              $A^{DG}_{v',v}$.alerts $\leftarrow \emptyset$;
          $A^{DG}_{\text{Current\_Node},v}$.alerts $\leftarrow A^{DG}_{\text{Current\_Node},v}$.alerts $\cup A^{\mathtt{D-L}}_{\text{Current\_Node},v}(t)$.alerts;
          if $v \notin$ Searched_Nodes $\cup$ Node_Queue **then**
            Node_Queue.ENQUEUE(v);
    Searched_Nodes $\leftarrow$ Searched_Nodes $\cup$ Current_Node;
return Damage_Graph $= (V^{DG}, E^{DG}, A^{DG})$;

Algorithm 9 A-CSM Function A.1 (Identifying Compromised Browsers)

Input: app_id, T, $G(t)$ for $t \in [t_1, t_2]$ and $0 \leq t_1 \leq t_2 \leq T$

Output: $\langle t, \mathsf{suspicious\_app}(t) \rangle$ for $t \in [t_1, t_2]$

1: **for** $t \in [t_1, t_2]$ **do**
2: $\quad \mathsf{suspicious\_app}(t) \leftarrow \emptyset$;
3: $\quad \mathsf{temp\_URL\_set} \leftarrow \emptyset$;
4: $\quad$ **for** $v \in V^{\mathtt{URL}}(t)$ **do**
5: $\quad\quad$ **if** $(\mathsf{app\_id}, v) \in E(t)$ **then**
6: $\quad\quad\quad \mathsf{temp\_URL\_set}(t) \leftarrow \mathsf{temp\_URL\_set}(t) \cup \{v\}$; ▷ v was accessed by app_id
7: $\quad$ **for** $v \in \mathsf{temp\_URL\_set}(t)$ **do**
8: $\quad\quad$ **for** $u \in V^{\mathtt{app}}(t)$ **do**
9: $\quad\quad\quad$ **if** $(u, v) \in E(t)$ **then**
10: $\quad\quad\quad\quad \mathsf{suspicious\_app}(t) \leftarrow \mathsf{suspicious\_app}(t) \cup \{v\}$; ▷ app u accessed URL v and is therefore suspicious
11: **return** $\langle t, \mathsf{suspicious\_app}(t) \rangle$ for $t \in [t_1, t_2]$;

Algorithm 10 A-CSM Function A.2 (Identifying Victims of A Malicious URL)

Input: url_id, T, $G(t)$ for $t \in [t_1, t_2]$ and $0 \leq t_1 \leq t_2 \leq T$

Output: $\langle t, \mathsf{victim\_apps}(t) \rangle$ for $t \in [t_1, t_2]$

1: **for** $t \in [t_1, t_2]$ **do**
2: $\quad \mathsf{victim\_apps(t)} \leftarrow \emptyset$;
3: $\quad$ **for** $u \in V^{\mathtt{app}}(t)$ **do**
4: $\quad\quad$ **if** $E(t)[u, \mathsf{url\_id}] \neq -1$ **then**
5: $\quad\quad\quad \mathsf{victim\_apps}(t) \leftarrow \mathsf{victim\_apps}(t) \cup \{u\}$; ▷ Application u accessed url_id
6: **return** $\langle t, \mathsf{victim\_apps(t)} \rangle$ for $t \in [t_1, t_2]$;

and $\mathsf{domain}_2 = \texttt{mali.g00gle.com}$. More specifically, it is the sum of the edit distance between components `mail` and `mali`, the edit distance between components `google` and `g00gle`, and the edit distance between components `com` and `com` respectively. Lines 9-15 of Algorithm 3.3.4 identify all the victim browsers that visited any of the spoofed URLs in set $spoofed\_urls(t)$. Algorithm 3.3.4 has a time complexity $\mathcal{O}((t_2 - t_1 + 1) \cdot \max_t |V^{\mathtt{app}}(t)| \cdot \max_t |V^{\mathtt{URL}}(t)|)$.

Algorithm 11 A-CSM Function A.3 (Identifying Victim URLs and Victim Applications of Spoofed URLs)

Input: $\mathsf{url\_id}$, T, $\tau_{distance}$, $G(t)$ for $t \in [t_1, t_2]$ with $0 \leq t_1 \leq t_2 \leq T$

Output: $\langle t, \mathsf{spoofed\_urls}(t), \mathsf{victim\_apps}(t) \rangle$ for $t \in [t_1, t_2]$

for $t \in [t_1, t_2]$ **do**
  $\mathsf{spoofed\_urls}(t) \leftarrow \emptyset$; $\mathsf{victim\_apps}(t) \leftarrow \emptyset$;
  for $v \in V^{\mathtt{URL}}(t)$ **do**
    if $0 <$EDIT_DISTANCE$(v, \mathsf{url_id}) \leq \tau_{distance}$ **then**
      $\mathsf{spoofed\_urls}(t) \leftarrow \mathsf{spoofed\_urls}(t) \cup \{v\}$;
  for $v \in \mathsf{spoofed\_urls}(t)$ **do**
    for $u \in V^{\mathtt{app}}(t)$ **do**
      if $(u, v) \in E(t)$ **then**
        $\mathsf{victim\_apps}(t) \leftarrow \mathsf{victim\_apps}(t) \cup \{u\}$;
return $\langle t, \mathsf{spoofed\_urls}(t), \mathsf{victim\_apps}(t) \rangle$, $t \in [t_1, t_2]$;

Algorithm 12 Edit Distance $(\mathsf{url}_1, \mathsf{url}_2)$ [108, 147]

Input: $\mathsf{url}_1, \mathsf{url}_2$

Output: $\mathsf{total\_distance}$ (edit distance between url_1 and url_2)

$\mathsf{domain}_1 \leftarrow$ EXTRACTING_DOMAIN(url_1);
$\mathsf{domain}_2 \leftarrow$ EXTRACTING_DOMAIN(url_2); ▷ extracting components; e.g., 'google' and 'com' for google.com
$\mathsf{compo}_1 \leftarrow$ EXTRACTING_COMPO(domain_1);
$\mathsf{compo}_2 \leftarrow$ EXTRACTING_COMPO(domain_2);
$\mathsf{MAX} \leftarrow \max(|\mathsf{compo}_1|, |\mathsf{compo}_2|)$;
$\mathsf{MIN} \leftarrow \min(|\mathsf{compo}_1|, |\mathsf{compo}_2|)$;
$\mathsf{total\_distance} \leftarrow 0$;
for $i \leftarrow 0$ to $\mathsf{MIN} - 1$ **do**
  if $|\mathsf{compo}_1| > |\mathsf{compo}_2|$ **then**
    $\mathsf{distance} \leftarrow$ LEVENSHTEIN$(\mathsf{compo}_1[\mathsf{MAX} - i], \mathsf{compo}_2[\mathsf{MIN} - i])$;
  else
    $\mathsf{distance} \leftarrow$ LEVENSHTEIN$(\mathsf{compo}_2[\mathsf{MAX} - i], \mathsf{compo}_1[\mathsf{MIN} - i])$;
  $\mathsf{total\_distance} \leftarrow \mathsf{total\_distance} + \mathsf{distance}$;
return $\mathsf{total\_distance}$;

3.4 B2CSM System and Evaluation

A straightforward realization of the CSM model depicted in Figure 3.2 would let each defender build a centralized cyber security management system (mainly for the

sake of efficiency) to maintain their own cyber data and perform CSM invocation due to some received threat intelligence. However, such a design is insufficient due to the considerations that: (i) centralized architecture typically poses the risk of single-point-of-failure, and therefore a decentralized system would be preferable to tolerate crash faults; (ii) when considering a decentralized system, even in the network of the same enterprise that the defender manages, it is still possible that part of the servers get corrupted, hence not only crash fault tolerance (CFT) but also byzantine fault tolerance (BFT) are needed to construct a robust CSM system; (iii) besides the robust storage of cyber data, the invocation records (e.g., which party invoked a CSM function in a certain time point) are also potentially valuable to realize accountability, and these records should be tamper-proof against malicious actions; (vi) The decentralized system is expected to correctly execute some pre-defined operations, i.e., the CSM functions, in an automated manner instead of the involvement of manual procedure. To overcome these challenges, we propose leveraging blockchain to build a decentralized, automated and robust blockchain-based CSM system, leading to B2CSM. In what follows, we present the key designs of B2CSM; instantiate it atop the blockchain platform; analyze its security properties and evaluate its performance based on real cyber data.

3.4.1 B2CSM Model and Architecture

Figure 3.6 highlights the B2CSM model, which extends the CSM model by storing defenders' cyber data, $G(t)$'s, in the B2CSM blockchain network and incorporating B2CSM Apps and B2CSM Agents. B2CSM Apps are the interface for defenders to run CSM functions, by (i) taking as input some cyber intelligence and identifiers (e.g., a time frame) of the relevant cyber data and (ii) presenting the output of the CSM functions to the defender. B2CSM Agents collect cyber data and write the data to the B2CSM blockchain network.

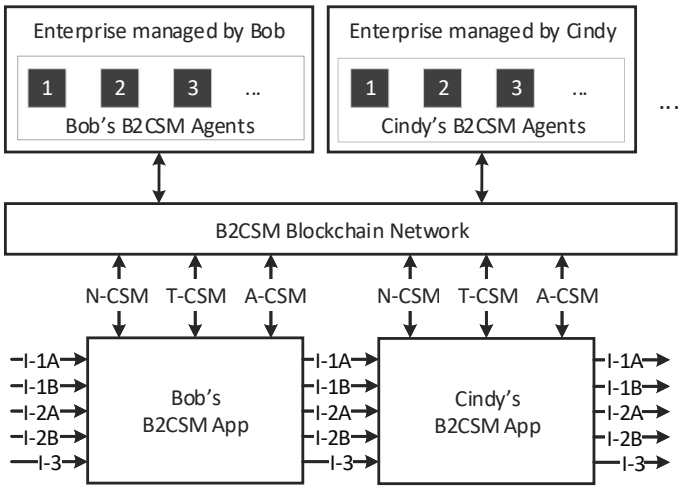


Figure 3.6 The B2CSM model extending from the CSM model.

The B2CSM model described in Figure 3.6 lends itself to the B2CSM architecture depicted in Figure 3.7, which is presented from the defenders' perspective. In this architecture, a defender uses a set of B2CSM agents to collect cyber data from the enterprise network. These agents write the collected data into the defender's B2CSM blockchain network. The defender interacts with their B2CSM App to execute CSM functions with some input cyber intelligence. The CSM functions run in the form of smart contract at full nodes in the B2CSM blockchain network. The B2CSM middleware acts as an intermediary between the B2CSM App and the blockchain network. To provide permissioned access control, the defenders are identified via a Certificate Authority (CA). These components interact with each other to form the B2CSM system.

3.4.2 B2CSM System Design and Security Analysis

Instantiating architecture into system. The B2CSM architecture in Figure 3.7 can be instantiated into B2CSM systems in different ways. We now propose a concrete instance by providing needed design choices:

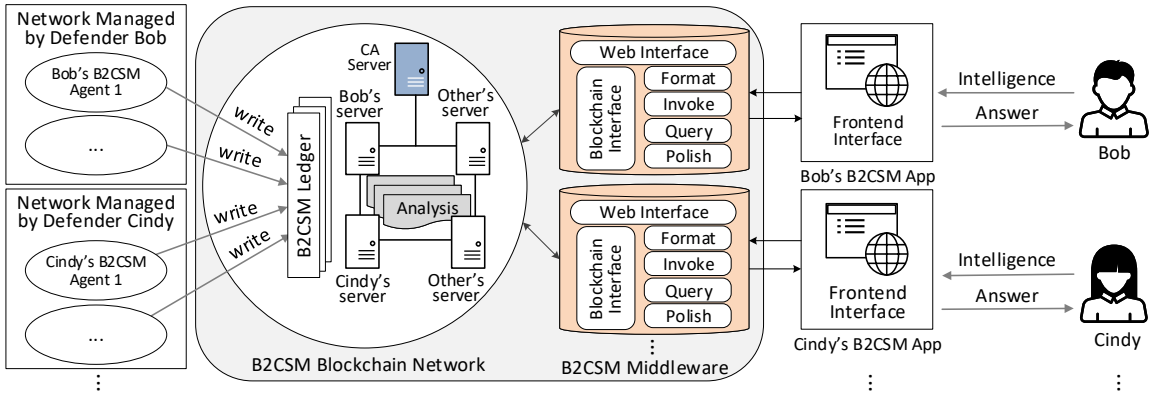


Figure 3.7 Illustration of B2CSM architecture.

Decision on blockchain type. We propose using the *permissioned* blockchain [9] to realize B2CSM due to the following reasons: (i) CSM needs to authenticate its participants and users because cyber security management copes with sensitive data; (ii) only parties who are interested in CSM (e.g., honest defenders share a common goal of protecting their systems from malicious attacks) need to participate in, and therefore it is unnecessary to be public to all; (iii) the participating entities may not fully trust each other, highlighting the importance of achieving accountability. Note that permissioned blockchains can be further divided into *private* blockchains, where the full nodes in the blockchain network belong to one enterprise, and *consortium* blockchains, where the full nodes are managed by multiple enterprises.

Decision on the number of chains and their types. Since there are three classes of CSM functions, we propose using one chain per class. The Fabric *channel* mechanism offers this service and creates a separated "subnet" containing its joined members, its ordering service nodes, a shared ledger, and the application chaincodes. One-chain-per-class provides a modular structure for the B2CSM network and allows for flexible extension of more potential CSM functions. In B2CSM, we propose two kinds of chains or channels:

- **Private chain/channel for storage**: A defender of each enterprise can create a *private* channel to store its own cyber data and perform different CSM functions, leading to a (permissioned) *private* B2CSM blockchain.
- **Consortium chain/channel for sharing**: Defenders can jointly create a channel to store cyber data (secret shares [128] or encrypted) and share their cyber intelligence, leading to a (permissioned) *consortium* B2CSM blockchain.

In both cases, each channel maintains a unique *ledger*, which consists of a *blockchain* for *on-chain* data storage (as transactions) and *state database* for *off-chain* data storage (as key-value pairs), and can serve specific CSM class, namely N-CSM, T-CSM or A-CSM.

Figure 3.8 depicts the channel architecture of B2CSM. The defender of an enterprise can create a private channel by only allowing the server nodes managed by the defender to join. Intuitively, the cyber data of an enterprise is maintained and accessed on by its own servers, which jointly maintain a distributed ledger. Moreover, defenders of different enterprises can create a consortium channel for sharing their cyber intelligence data such as the outputs of CSM function invocations. Following a general BFT consensus security model [23], the number of full nodes N in any channel satisfies $N \geq 3f + 1$, where f is the number of faulty nodes that can be tolerated.

Decision on the consensus protocol. As our threat model considers compromised blockchain network nodes, we need to make B2CSM achieve Byzantine Fault Tolerance (BFT). Note that the Ordering Service Nodes (OSNs) in Fabric are external nodes (i.e., rather than the blockchain's full nodes) and that the ordering service only supports Crash Fault-Tolerance (CFT) consensus mechanisms such as Zookeeper with Kafka or Raft [65]. To achieve BFT, we propose integrating the work in [135], known as BFT-SMaRt. Moreover, we propose running the ordering service at the full nodes of the B2CSM blockchain, instead of delegating this service to extra nodes.

Decision on the state database to use. Fabric supports *leveldb* and *couchdb* as state databases. Although both support key-value storage, *couchdb* offers rich queries (e.g., the value can be JSON format whereas *leveldb* only supports string-based queries).

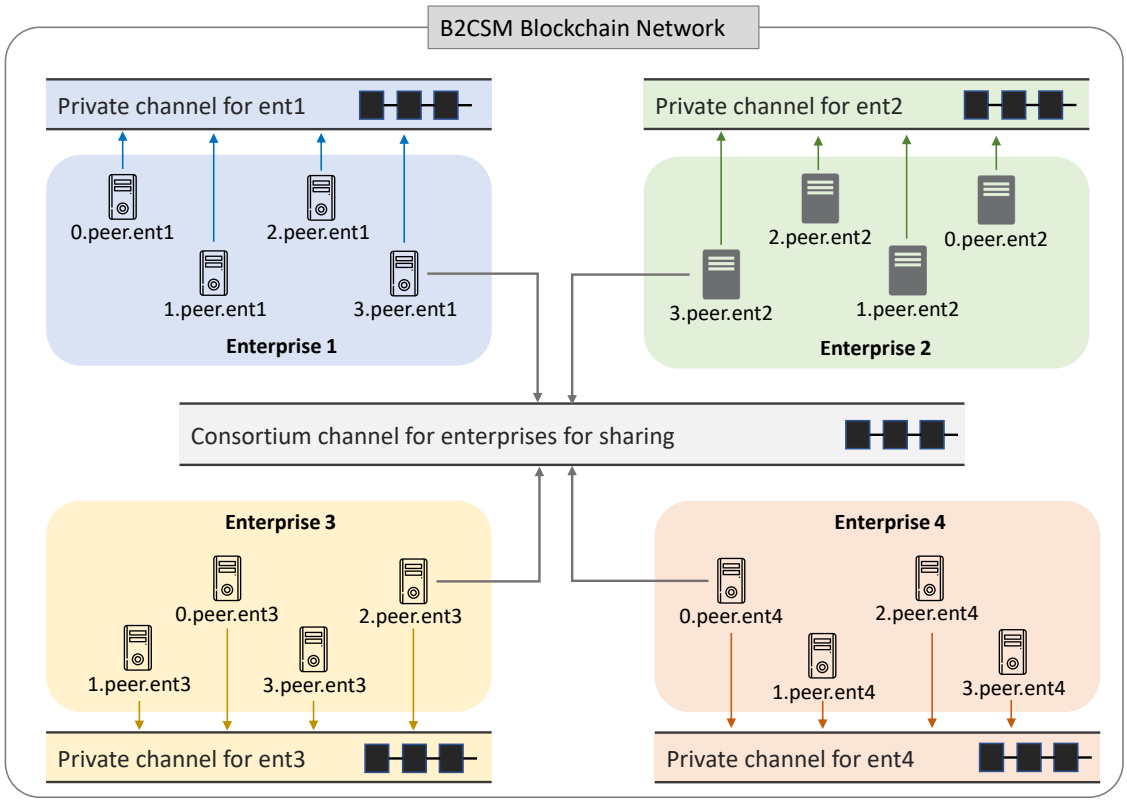


Figure 3.8 The channel architecture in B2CSM blockchain network.

In light of this, we adopt couchdb as the B2CSM state database and the concrete data format is elaborated in Section 3.4.2.

Decision on the locality of the B2CSM middleware. We propose running the middleware at every B2CSM blockchain full node. The middleware has multiple sub-functions, such as formatting a defender's invocation of CSM functions, interacting with the B2CSM blockchain network and decentralized storage network, and polishing the output of CSM functions before returning it to the B2CSM App. These services are important because (i) different kinds of CSM functions may require different kinds of data pre-processing, and (ii) the middleware serves as an intermediate level of abstraction to support extensive functionalities that may emerge in the future.

B2CSM system design. In light of the design choices above, we now present the B2CSM system design, which contains two main phases: *cyber data replication* and *CSM function invocation*.

Phase I: cyber data replication. This phase allows a defender, e.g., Bob, to robustly store the cyber data via private channels in the B2CSM blockchain network. Upon system is setup, defenders can continuously write collected cyber data to the channel via B2CSM agents, i.e., the local servers managed by Bob. The key challenge is *how to deal with a large volume of cyber data* in terms of efficient writing and reading, and robust storage. We propose two methods to solve this challenge and analyze their advantages and disadvantages. Specifically,

Method 1 $\mathcal{M}_1$: splitting into chunks with fine-grained ledger structure. In order to handle a large volume of cyber data $G(t)$, we need to attain efficient *uploading* and *retrieval*. In order to replicate a large volume of cyber data to the blockchain network, we propose dividing $G(t)$ into small data units and uploading them in parallel. On the other hand, the efficiency of data retrieval largely depends on how the data units are stored in the B2CSM network. While it is tempting to store all of the cyber data on the blockchain in the form of transactions and use smart contracts to point to which blocks contain the relevant data units for what time window, this design will incur large latency when multiple blocks need to be traversed. Besides, a block may contain data units belonging to different time windows as the block size is fixed when initializing a channel, leading to possible retrieval of irrelevant cyber data. This prompts us to propose a proper ledger structure by extending the Fabric state database: the blockchain full nodes not only reach consensus on data units and package them into consecutive blocks, but also proactively update the state database for later efficient retrieval purposes.

Figure 3.9 depicts the structure of the B2CSM ledger, where the blockchain stores two kinds of transactions: (i) the transactions containing the history of cyber

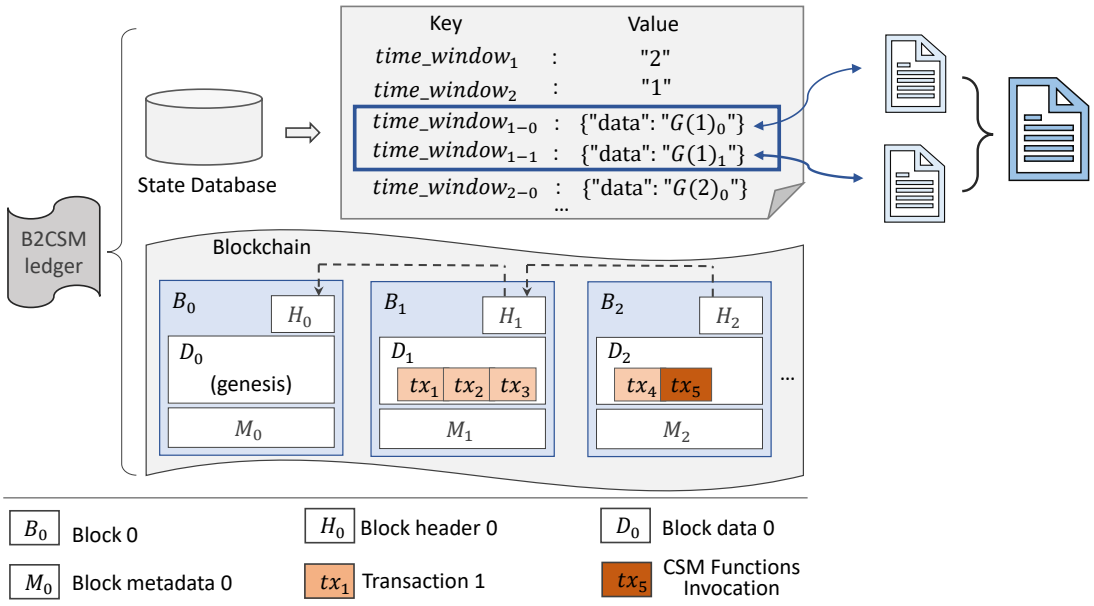


Figure 3.9 Illustration of the B2CSM ledger structure with a blockchain and a state database. The state database stores the cyber data units.

data replication (i.e., who submits which cyber data to the B2CSM blockchain network); (ii) the transactions containing the history of CSM function invocations for auditing purposes. The *state database* stores real cyber data $G(t)$ as shown in Figure 3.9, where a large $G(t)$ is divided into multiple data units. For example, $time\_window_1$ consists of two data units that are respectively keyed by $time\_window_{1\text{-}0}$ and $time\_window_{1\text{-}1}$. When defenders make CSM queries, the B2CSM middleware can invoke the CSM functionalities in the smart contracts, which take as input the relevant cyber data that is retrieved from the state database. This fine-grained ledger structure leverages the advantages of both blockchain and database structures [129] to facilitate blockchain-based applications as they process large volumes of data.

Method 2 $\mathcal{M}_2$: integrating with decentralized storage network. An alternative way of handling a large volume of cyber data is to incorporate with decentralized storage network, instantiated by IPFS. The key idea lies in storing the real cyber data in the IPFS while recording a reference, i.e., the returned content

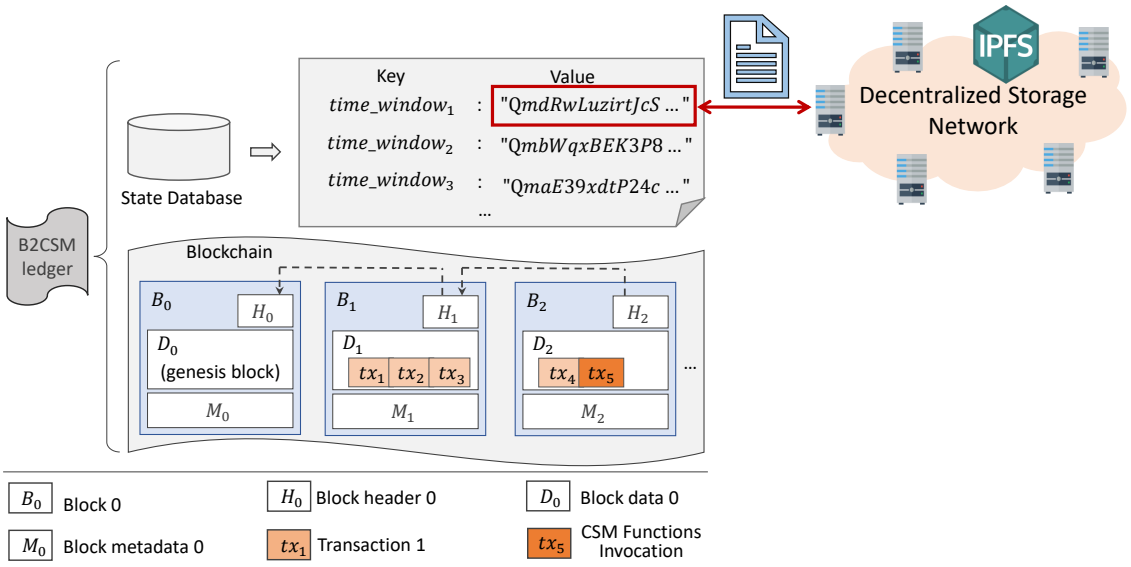


Figure 3.10 Illustration of the B2CSM ledger structure with a blockchain and a state database. The state database stores content id returned by IPFS.

id *cid* of the cyber data on blockchain. We leverage the (private) IPFS cluster [89] that can be deployed by the defender on its own servers instead of the public version to store the cyber data since in that case, the data uploaded to IPFS typically needs to be encrypted, leading to extra cost for data decryption during retrieval.

One potential issue that may appear in the Fabric-IPFS enabled hybrid architecture is that a corrupted full node in the B2CSM blockchain network may maliciously modify or drop the *cid*, leading to subsequent inaccessibility. To tackle this issue, we leverage a gossip-based[1] diffusion method and propose an augmented consensus mechanism. At a high-level, the replication procedure $\mathcal{M}_2$ works as: (i) a B2CSM agent signs the collected cyber data $G(t)$ along with the time window t and submitted to $f+1$ full nodes in the defender created private B2CSM blockchain channel; (ii) these $f+1$ full nodes execute the gossip procedure so that all full nodes can cache the cyber data temporarily; (iii) the *leader* node in the consensus

[1]A gossip protocol is a procedure where the cyber data $G(t)$ can be routed to all full nodes by letting each peer node randomly and uniformly select θ neighbor nodes and forward the data.

mechanism submits the cyber data to IPFS and receives the *cid*, then starts the BFT consensus, e.g., via BFT-SMaRt [16] consensus mechanism, with other full nodes, eventually all full nodes receive the *cid*; (iv) each full node retrieves the data in IPFS via the *cid* and verifies the attached signature generated by defender, if valid, updates the state database with the key of time window t and the value of *cid*, as depicted in Figure 3.10, and then clears the locally cached cyber data. Otherwise, if the signature is invalid, a *view change* (VC) is triggered to elect a new leader node and restart from the prior step (iii).

Comparison of the two methods $\mathcal{M}_1$ and $\mathcal{M}_2$. The method $\mathcal{M}_1$ stores the cyber data in the state database, which brings the advantage that the chaincode/smart contract of CSM functions can conveniently and efficiently retrieve cyber data for function execution. But for this method, it essentially stores the copy of the cyber data on each full node, leading to relatively higher storage cost (compared with $\mathcal{M}_2$). For the method $\mathcal{M}_2$, the advantage lies in less on-chain storage cost, e.g., consider there are N full nodes in the channel and the cyber data size is 1Gb, then the on-chain storage cost for $\mathcal{M}_2$ is $N \times 46$ bytes (i.e., the length of a *cid*) along with the 1Gb cyber data that stored in IPFS, while $N \times$ 1Gb for method $\mathcal{M}_1$. However, the drawback of $\mathcal{M}_2$ becomes clear during data retrieval. Specifically, the CSM function needs to execute by taking as input the cyber data and threat intelligence, in that case, it is not common to let smart contract directly retrieve from IPFS (since it is external source, which may cause non-determinism in Fabric chaincode). Hence, either new retrieval mechanism needs to be developed or query/invocation latency would become considerably large (for large cyber data) if the defender downloads from IPFS and feeds to the chaincode. How to get the best of both of these two methods will be one very interesting question for future work.

Phase II: CSM functionality invocation. After the cyber data $G(t)$ is replicated to the Fabric channel (i.e., via method $\mathcal{M}_1$) or the Fabric-IPFS enabled architecture

(i.e., via method $\mathcal{M}_2$), the defender Bob can invoke CSM functions to identify potential risks with respect to a given piece of threat intelligence. We describe the high-level idea here: (i) the defender submits the time window t, the CSM type such as N-CSM, and the intelligence via B2CSM App, which forwards to multiple full nodes in the private channel; (ii) these full nodes (i.e., the B2CSM middleware on them) execute the CSM functions and sign the result; (iii) the B2CSM App aggregates the results from these full nodes and present to the defender.

Cyber threat intelligence sharing. Besides the two main phases above in B2CSM system, a potential phase is cyber threat intelligence sharing. Practically sharing cyber intelligence is at the defender's discretion. In B2CSM, if the defender Bob would like to share cyber intelligence with other defenders, the following operations can be conducted: (i) the shared cyber intelligence can be encrypted using public key encryption (PKE) so that the sensitive information contained in the intelligence data can be protected against others; (ii) additionally, the cyber intelligence data can be shared in a consortium channel, where only the defenders who would like to share with each other are involved and can access. Such a consortium channel-based sharing mechanism brings an added advantage of accountability due to the immutability property of blockchain.

Also, in both cases, the defender who shares the (encrypted) cyber intelligence can sign the shared intelligence, and the resulting signature acts as the proof of authenticity of the intelligence data. As mentioned earlier, guaranteeing the authenticity of the threat intelligence *per se* is an orthogonal research problem and the extensive study of cyber intelligence sharing [2, 7, 20, 32, 126, 127] in B2CSM system naturally forms one future work.

A specific CSM functionality invocation demonstrating data flow. Now we utilize a specific CSM function N.1 and the method $\mathcal{M}_1$ to demonstrate a concrete data flow. As a pre-execute phase, the cyber data for N.1 is collected and stored as

an adjacency matrix where each row represents an external IP address, each column is an internal IP address, and the value (either 1 or 0) in the i-th row and the j-th column indicates whether such an external IP has visited the internal IP during a time period or not. The adjacency matrix is parsed as JSON format and then submitted to the private channel that created by the defender in B2CSM blcockchain network following the steps in phase I. Listing 1 illustrates an example of stored cyber data (units) in state database.

Listing 1 N-CSM Cyber Data $G(t)$ Example (in State Database)

```
{
  "timeWindow": "20211101", // It means "20211101-20211102" if replication frequency is one day
  "allDataUnits": [{
      "dataUnit": "20211101-0",
      "externalIPs": ["192.168.10.74", "192.168.10.75", "192.168.10.81"],
      "internalIPs": ["192.168.1.115", "192.168.1.116", "192.168.1.67"],
      "visitRecords": [["0","1","1"],["1", "1","0"],["0", "0","0"]]
    },
    {
      "dataUnit": "20211101-1",
      "externalIPs": ["192.168.10.74", "192.168.10.75", "192.168.10.81"],
      "internalIPs": ["192.168.1.121", "192.168.1.124", "192.168.1.7"],
      "visitRecords": [["0","0","1"],["0", "1","1"],["1", "0","0"]]
    },
    ...
    ],
    ...
}
```

Consider the example of function N.1, which identifies potential victims of an attacker with attackerIP during time interval $[t_1, t_2]$. In this case, the following steps occur: (i) the defender invokes the B2CSM App with the threat intelligence shown in Listing 2 and specifies that the channel is N-CSM; (ii) the App sends a request to multiple full nodes, and on each node the B2CSM middleware invokes the N.1 function that deployed as chaincode in the N-CSM channel; (iii) the chaincode retrieves the cyber data from state database and executes the pre-defined processing functions and outputs the potential victim IP addresses that have been attacked by attckerIP during time interval $[2021/11/01, 2021/11/02]$; (iv) the B2CSM middleware signs the output on behalf of the full node and returns this to the App; (v) the server running the

Listing 2 N-CSM Cyber Intelligence Example

```
{
"attackerIP": "192.168.10.74", // An external IP address
"timeInterval": "20211101-20211102"
}
```

Listing 3 N-CSM Cyber Data $G(t)$ Example (in IPFS)

```
{
  // It means "20211101-20211102" if replication frequency is one day
  "timeWindow": "20211101",
  "cyberData": {
    "externalIPs": ["192.168.10.74", "192.168.10.75", "192.168.10.81",...],
    "internalIPs": ["192.168.1.115", "192.168.1.116", "192.168.1.67",...],
    "visitRecords": [["0","1","1",...],["1", "1","0",...],["0", "0","0",...],
    ...]
    },
  "oracle_proof": {
    "value": "..."
    },
    "defender_signature": {
    "value": "..."
    },
    "meta_data": {
    "timestamp": "...",
    ...
    }
}
```

B2CSM App verifies the signatures for the results received from the full nodes and shows the defender a set of victims' IP addresses. Listing 3 further shows an example of the cyber data that stored in IPFS. Note that such a potentially large JSON file in Listing 3 is split into chunks of 256 kb and stored on different IPFS peer nodes.

If some full nodes in the N-CSM channel crash, a defender is notified that those servers are unreachable. Furthermore, if the signature verifications of some full nodes fail, the defender will also be notified that those servers are suspicious of being attacked. Consequently, corresponding actions, e.g., replace the suspicious full node and join new servers to the N-CSM channel, can be taken by the defender. Note that the preceding discussion similarly applies to other CSM functions.

Security objectives. We define five security objectives for B2CSM.

Correctness. The correctness of the outputs of the CSM functions is assured, with respect to the input cyber intelligence and the cyber data $G(t)$.

Integrity. The integrity of data, namely the cyber data written by the B2CSM agents to the B2CSM blockchain network (and DSN in method $\mathcal{M}_2$), and the invocation history of the CSM functions stored in blockchain, is assured. This means the data cannot be manipulated without detection, as long as the fraction of compromised nodes in the underlying blockchain is bounded by a certain upper threshold.

Availability. The availability of the data stored in B2CSM is assured. Specifically, the cyber data written by the B2CSM agents to the B2CSM blockchain network (and DSN in method $\mathcal{M}_2$), and the invocation history of the CSM functions stored in the blockchain must remain available as long as the fraction of compromised nodes in the underlying blockchain network is bounded from above by a certain upper threshold.

Consistency. The consistency of the data, namely cyber data written by the B2CSM agents to the B2CSM blockchain network (and DSN in method $\mathcal{M}_2$), and the invocation history of the CSM functions stored in blockchain, is assured. This means all of the honest nodes in a B2CSM channel have the same global view about the data's state, as long as the fraction of compromised nodes in the underlying blockchain platform is bounded by a certain upper threshold.

Accountability. The B2CSM agents cannot write data into the blockchain network without record of the writing. Similarly, the B2CSM Apps cannot run CSM functions without record of the activity.

Threat model. We consider an attacker with the following capabilities: (i) The attacker can compromise B2CSM blockchain full nodes, by penetrating into some bounded fraction of them. The attacker has total control over these compromised nodes and can coordinate their activities in an arbitrary (i.e., Byzantine) fashion. (ii) The attacker can interfere with message deliveries. The attacker can control the order of message deliveries in the blockchain network. The attacker can arbitrarily delay message deliveries to each computer (but not forever, see Assumption 2 below)

by waging Denial-of-Service (DoS) or other similar attacks. We consider the attacker with following standard abilities.

Assumption 1. Cryptographic assurance. We make standard assumptions to assure the security of cryptographic schemes (e.g., digital signatures). Informally speaking, these assumptions say that as long as cryptographic keys (if applicable) are not compromised, cryptographic schemes are secure. That is, in order for the attacker to compromise a cryptographic assurance, the attacker has to penetrate into a system in question to compromise the cryptographic keys or cryptographic service (for attaining "oracle" access to a cryptographic function) [150].

Assumption 2. Communication model. For the B2CSM blockchain network, we assume the communications between the full nodes are *partially synchronous*, meaning that each message is delivered to the honest nodes within some unknown delay [36]. While in other steps in the B2CSM system, the communication is considered *synchronous* in the sense that the message can only be delayed up to *a-priori* known time period Δ.

Assumption 3. Corruption threshold. For the full nodes in any channel (since each (private or consortium) channel represents a separated ledger) of the B2CSM blockchain network, we assume that no more than one-third of them are compromised simultaneously, which is inherent to the adopted Byzantine Fault-Tolerance (BFT) protocol [135].

Security analysis. Consider the attacker cannot compromise a defender Bob or the computers running the B2CSM App; otherwise, the attacker can manipulate the output arbitrarily. The security analysis of B2CSM systems instantiated from the B2CSM architecture is analyzed as follows, which is based on method $\mathcal{M}_1$.

- *Correctness.* The correctness states that the outputs of the CSM functions are reliable. To generate the authentic outputs, we can analyze each-step execution during the whole data flow: (i) the authenticity of the input cyber intelligence is considered correct by validating the digital signature attached with the intelligence data, which is generated by the sharer; (ii) the integrity of $G(t)$

stored in B2CSM blockchain network can be ensured due to the immutability property of blockchain; (iii) with the authentic input cyber intelligence and integrated cyber data, the CSM functions can be correctly executed unless the attacker can manipulate the execution of smart contract in blockchain, which is of negligible probability; and (iv) no more than one-third of the full nodes can be compromised simultaneously, namely assumption 3, which ensures the defender to receive the correct outputs by picking majority (i.e., $f+1$ same results or the majority of $2f+1$ returned results, where f is the malicious nodes that can be tolerated in the blockchain network with N full nodes [23, 63]) of the invocation results from the full nodes.

- The *integrity*, *availability* and *consistency* objectives are assured by the inherent properties of blockchain [9], including: (i) security of cryptographic primitives such as hash functions and asymmetric signatures, namely assumption 1; (ii) the distributed architecture of the blockchain system; (iii) the execution of the consensus mechanism in the partial synchronous network model, i.e., assumption 2. Meanwhile, the *accountability* objective is ensured because (i) the data including B2CSM agents' public key and timestamp is stored as transactions when writing cyber data to the blockchain network; (ii) when a defender invokes CSM functions, the smart contract is automatically triggered to record such an activity. With the assurance of aforementioned integrity of blockchain data, all the activities cal be tracked, leading to accountability.
- *Accountability.* The accountability objective is ensured because (i) the data including the defender's public key and timestamp is stored as transactions when writing cyber data to the blockchain network; (ii) when a defender invokes CSM functions, the smart contract is automatically triggered to record such an activity. With the assurance of aforementioned integrity of blockchain data, all the activities cal be tracked, leading to accountability.

Overall, the CSM invocation can be automatically (due to the reliable execution of CSM functions in the pre-determined and deployed smart contract) performed without any manual inference given the authentic cyber intelligence, and the returned result is guaranteed to be correct in the sense of all the aforementioned properties.

3.4.3 B2CSM System Performance and Analysis

Performance metrics. We propose two CSM-specific performance metrics: (i) *Data Replication Throughput* (DRT). The DRT metric measures the performance in writing data to the B2CSM blockchain. Since $G(t)$ is often large in volume and would be splitted into multiple chunks as in method $\mathcal{M}_1$, each with m rows and n columns,

e.g., $m = 3$ and $n = 3$ in Listing 1. We call each chunk a *data unit*, whose size is limited by the transaction size in blockchain network. Let $|\Phi|$ be the size of $G(t)$ and $\mathcal{T}_{replication}$ be the the total time cost for replicating $G(t)$ to the blockchain. Then we define $\mathsf{DRT} = |\Phi|/\mathcal{T}_{replication}$; (ii) *Application Query Latency* (AQL). The AQL metric measures the time interval between when a defender invokes a CSM function and when the defender receives the response, namely $\mathcal{T}_{invocation} = \mathcal{T}_{reqf} + \mathcal{T}_{cp} + \mathcal{T}_{resf}$, where $\mathcal{T}_{reqf}$ is the request formatting time (i.e., the time interval between the B2CSM middleware receiving a request from a B2CSM App and the B2CSM middleware submitting the transaction to the blockchain network), $\mathcal{T}_{cp}$ is the chaincode processing time (i.e., the time interval between the channel starting to execute the CSM function and the middleware receiving the query result from the blockchain network), and $\mathcal{T}_{resf}$ is the response formatting time (i.e., the time interval between the middleware receiving the result from the blockchain network and the middleware sending the result to the B2CSM App).

The above performance metrics are affected by the following *block-cutting* parameters that are involved when encapsulating transactions into blocks: *batch size* (by default, 10 transactions per block); *batch timeout* (by default, 2 seconds); and *block size* (by default, 512 KBytes). When the *batch size* or *block size* are met, or the *batch timeout* is reached, the OSNs encapsulate transactions into a new block. This means that one $G(t)$ might be stored into multiple blocks. Inspired by [141], we use the following block-cutting parameters in our experiments (unless explicitly specified otherwise): block timeout = 2 seconds; block size = 512KB; batch size = 30 transactions per block.

A B2CSM prototype system. We implement a prototype system of B2CSM to analyze the performance. The preceding design choices influence the prototype system, and a four-node architecture is depicted in Figure 3.11. The B2CSM prototype system is built on top of a browser-server architecture. The B2CSM App

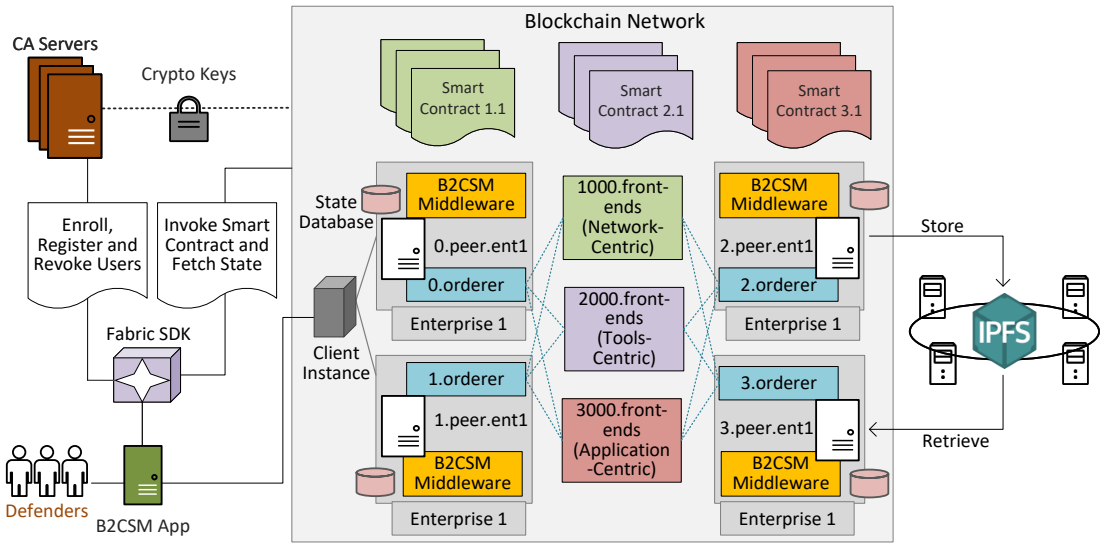


Figure 3.11 Illustration of the B2CSM prototype system with 4 blockchain peer nodes, on which a private channel is created for one enterprise.

has two modules: one displays blockchain-related information, including a dashboard with various kinds of information (e.g., B2CSM blockchain's peer nodes' IP addresses, the numbers of blocks and transactions for each channel). This presents a defender with the B2CSM blockchain's status in real-time. The other module offers a defender with a web-based interface to run the desired CSM functions with input cyber intelligence and receive the response from the CSM functions.

The Fabric software development kit provides the interfaces for interacting with the blockchain network (e.g., register users, install chaincode, instantiate chaincode, invoke transactions, and query ledgers). A Fabric client is instantiated when the defender initiates communication with the B2CSM blockchain network. This client only needs to be instantiated *once*, and subsequent sessions with the blockchain network can reuse it.

Experiments design and performance evaluation. We conduct experiments with the prototype system involving (as an example) one defender or enterprise, denoted by *ent1*. The defender has a range of CSMAs responsible for writing cyber data to the B2CSM blockchain network. The blockchain consists of four peer nodes,

denoted by *0.peer.ent1*, *1.peer.ent1*, and so on. These peer nodes are the full nodes for the B2CSM blockchain. There are four couchdb databases: *Couchdb_Peer0_Ent1*, *Couchdb_Peer1_Ent1*, etc. Each couchdb state database is connected with one peer node for recording its current world state.

There are four ordering nodes: *0.orderer*, *1.orderer*, *2.orderer*, and *3.orderer*. These four nodes act as the replicas for BFT SMaRt-based ordering service, which assures that as long as the fraction of malicious nodes does not exceed 1/3 (i.e., 1 when there are 4 full nodes), the ordering service is secure. We also conduct the experiment on 7 (tolerating 2 faulty nodes) and 10 (tolerating 3 faulty nodes) ordering nodes that reside on peer nodes.

There are three frontends, named *1000.frontends* (for N-CSM), *2000.frontends* (for T-CSM), and *3000.frontends* (for A-CSM). These frontend nodes are responsible for (i) relaying the transactions that are issued by the B2CSM clients to the consensus protocol and (ii) forwarding the blocks that are generated by the ordering nodes to peer nodes.

It is worth pointing out that the above architecture can be readily tuned to build a consortium blockchain network by re-running the network setup with changed configuration file such that, e.g., the peer nodes' names would change from *0.peer.ent1*, *1.peer.ent1*, *2.peer.ent1*, *3.peer.ent1* to *0.peer.ent1*, *0.peer.ent2*, *0.peer.ent3*, *0.peer.ent4* respectively and so do other service components such as ordering nodes, and then letting these components join in the same channel that a defender creates.

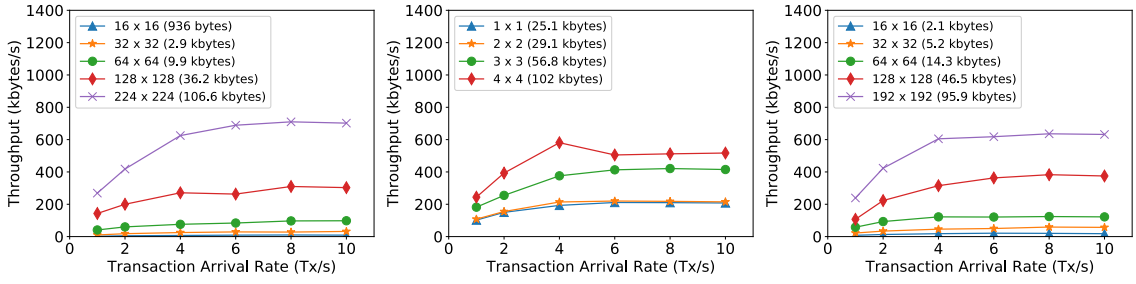
The hardware for conducting our experiments is a small-scale cluster of four Virtual Machines (VMs) residing on two heterogeneous servers, representing four nodes to formulate a private B2CSM blockchain. One server is a Dell PowerEdge R740, which is equipped with 2 Intel(R) Xeon(R) CPU Silver 4114 processors (with 13.75 MB L3 cache and 20 cores of 2.2 GHz for each processor), 256 GB (16 slots $\times$

16 GB/slot) 2400MHz DDR4 RDIMM memory, and an 8 TB (8 slots × 1TB/slot) 2.5 inch SATA hard drive. The other server is a Dell Precision Rack 7910, which is equipped with 2 Intel(R) Xeon(R) CPU E5-2630 v3 processors (with 15MB cache and 6 cores of 2.4GHz for each processor), 16GB 2133MHz DDR4 RDIMM ECC memory, and a 256GB 2.5 inch SATA solid state drive. The four VMs have the same configuration of 8 vCPUs, 24 GB memory and 800 GB hard drive and are connected via a Local Area Network (LAN). The operating system in each VM is Ubuntu 16.04 (64-bit) with kernel version 4.15. The Fabric version is 1.2, the Java version is 1.8.0_211, and the golang version is 1.11.10.

B2CSM performance based on experiments with real-world datasets. We now evaluate CSM-specific performance in DRT and AQL using real data. In N-CSM experiments, we utilize a dataset collected from a honeypot during 7 days, and the time resolution is days (i.e., each day is a time interval). In T-CSM experiments, we use a dataset collected by the USMA team from the 2017 CDX Competition [118], as if it were collected at a production enterprise network, which indeed instantiates the model highlighted in Figure 3.4. As this dataset does not have ground truth tags, for our experimental purposes, we replay the traffic using a popular open-sourced intrusion detection system, Suricata [138], with a popular, free ruleset referred to as Emerging Threats [68]. We store Suricata's alerts in an AGTSR $G(t)$ for time window t, where nodes represent the source and destination IP addresses of each attack. In A-CSM experiments, we consider the example of a defender recording how an enterprise's browsers have accessed the external URLs. In the simplest case, the cyber data is stored in the form ($browser$, URL, $timestamp$), meaning that $browser$ accessed the URL at the time given by $timestamp$. Our experiments employ the Georgia Tech data received from [139] over the period of 2/1/2019-2/6/2019. The data contains mappings between malware instances, which are treated as browser

applications for our purpose, and the external URLs. The data is pre-processed into a bipartite AGTSR over the time horizon of $T = 6$ days.

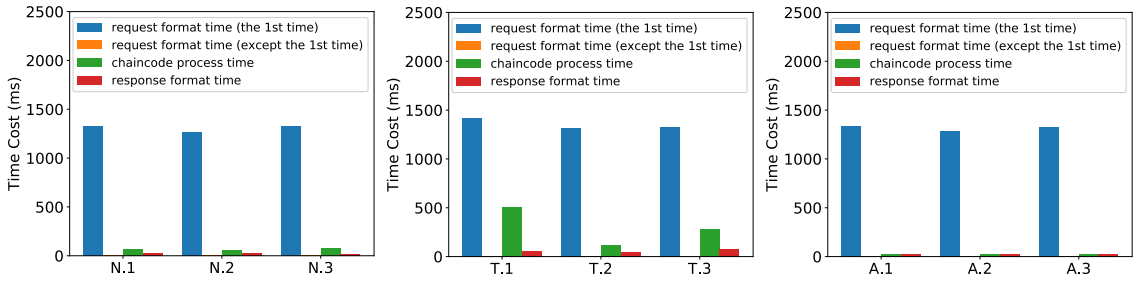
Figure 3.12(a), 3.12(b), and 3.12(c) plots B2CSM's cyber data replication throughput (denoted by $\mathsf{DRT}_{\mathsf{CSM}}$) using the real-world datasets mentioned above. We observe that the throughput varies with CSM scenarios. The throughput of T-CSM is significantly different from those of N-CSM and A-CSM. This is caused by the fact that the T-CSM data is quite different from the N-CSM and A-CSM data as follows. The T-CSM data volume is large and the volumes of data units vary substantially because some data units contain more empty elements than others (recalling that T-CSM data is generated from network traffic); in contrast, N-CSM data and A-CSM data are uniformly distributed (i.e., data units are about the same size). This explains why T-CSM has a lower throughput. From the throughput, we observe that after the transaction arrival rate exceeds 4, the throughput stays stable, especially for N-CSM and A-CSM; this may be caused by the limited computing resources on the full nodes in our experiments. In T-CSM, we observe an "abnormal" throughput at transaction arrival rate 4 and data unit of 4×4 (i.e., 102 KBytes per unit); this may be caused by the limited computing resources at the full nodes and the cumulative effect of non-uniform distribution in the units' data volumes.



(a) N-CSM $\mathsf{DRT}_{\mathsf{CSM}}$ (b) T-CSM $\mathsf{DRT}_{\mathsf{CSM}}$ (c) A-CSM $\mathsf{DRT}_{\mathsf{CSM}}$

Figure 3.12 B2CSM's $\mathsf{DRT}_{\mathsf{CSM}}$ in different CSM experiments.

Figure 3.13(a), 3.13(b), and 3.13(c) plots B2CSM's AQL using the real-world datasets mentioned above. We observe the following: (i) For the request formatting time, it takes about 1.4 seconds for the *first* invocation of a CSM function, but much smaller time for subsequent invocations. This is because the former requires us to initialize a (one-time) Hyperledger Fabric client object on behalf of the B2CSM App before connecting to the blockchain network; whereas, the latter can simply reuse the object created by the former. (ii) For the chaincode processing time, the time cost varies for different invocations of CSM functions. (iii) The response time is relatively stable (i.e., varies only slightly).



(a) N-CSM AQL (b) T-CSM AQL (c) A-CSM AQL

Figure 3.13 B2CSM's AQL in different CSM cases.

Table 3.1 further presents the break-down of the latency time, where $\mathcal{T}^1_{reqf}$ is the request formatting time when a CSM function is invoked *for the first time* by a B2CSM App and $\mathcal{T}^2_{reqf}$ is the request formatting time after the initial invocation of a CSM function. We highlight that the former time costs $\mathcal{T}^1_{reqf}$ is only one-time though it is relatively longer. Besides, the chaincode processing time depends on the smart contract complexity (i.e., the complexity of a CSM function). Finally, the response formatting time $\mathcal{T}_{resf}$ is bigger than the request formatting time $\mathcal{T}^2_{reqf}$ when disregarding object-creating time during the first invocation of a CSM function; this is because each full node needs to sign the query results before sending back to the B2CSM App. In summary, we have the following conclusion: *The response delay is*

Table 3.1 B2CSM's Application Query Latency (unit: ms)

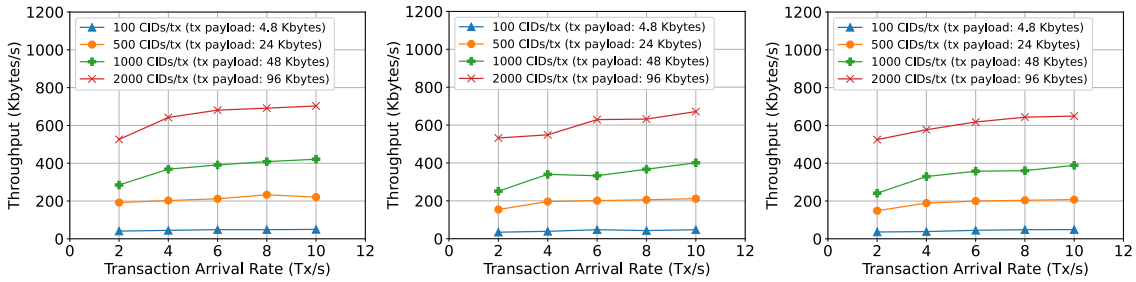
CSM Classes	CSM functions	$\mathcal{T}^1_{reqf}$	$\mathcal{T}^2_{reqf}$	$\mathcal{T}_{cp}$	$\mathcal{T}_{resf}$
N-CSM	N.1	1321	0.17	69.18	23.47
	N.2	1265	0.18	57.6	23.49
	N.3	1329	0.17	75.86	18.37
T-CSM	T.1	1420	0.19	504.27	52.81
	T.2	1317	0.16	120.13	46.92
	T.3	1327	0.17	279.63	72.66
A-CSM	A.1	1336	0.21	28.92	28.14
	A.2	1287	0.19	27.51	24.23
	A.3	1324	0.17	30.33	30.62

mainly due to: (i) the creation of a Hyperledger Fabric client object corresponding to a CSM function invoked from a B2CSM App for the first time; and (ii) the specific chaincode execution of CSM functions. Reducing these time costs can relatively improve the response time.

Scalability with varied number of nodes. Figure 3.14(a), 3.14(b), and 3.14(c) plots the throughput (denoted by $\mathsf{DRT}_{\mathsf{bc}}$) of replicating some general data (in the form of strings) such as (a batch of) content ids (cids) [2] to the blockchain network with 4 orderers (tolerating 1 faulty node), 7 orderers (tolerating 2 faulty nodes) and 10 orderers (tolerating 3 faulty nodes). Note that the ordering service is deployed on the peer nodes without delegating to extra nodes. Each transaction submitted to blockchain network contains various number of content ids as payload,

[2]We use cids here for all experiments since we are now examining the influence on the number of nodes, not data type; also, as we also consider that the cyber data to be replicated to IPFS, and only the returned content ids are stored in B2CSM blockchain network, the cyber data type (i.e., N-CSM, T-CSM or A-CSM) would not impact writing throughput to blockchain.

which yields different transaction size and would be updated to state database via smart contract. The transaction arrival rate shows how many transactions are simultaneously submitted via multi-threads. Note that if we examine one specific CSM class, e.g., N-CSM, which possesses the same cyber data format, then the throughput $\mathsf{DRT}_{\mathsf{CSM}}$ follows the same pattern with $\mathsf{DRT}_{\mathsf{bc}}$ with respect to various number of orderer (or peer) nodes.



(a) $\mathsf{DRT}_{\mathsf{bc}}$, 4 orderers/peers (tolerating 1 faulty node) (b) $\mathsf{DRT}_{\mathsf{bc}}$, 7 orderers/peers (tolerating 2 faulty nodes) (c) $\mathsf{DRT}_{\mathsf{bc}}$, 10 orderers/peers (tolerating 3 faulty nodes)

Figure 3.14 B2CSM's $\mathsf{DRT}_{\mathsf{bc}}$ with different number of orderers/peers.

From the throughput $\mathsf{DRT}_{\mathsf{bc}}$, we have the following observations: (i) Increasing the transaction size, namely incorporating more cids in a transaction, can significantly improve the throughput. However, the transaction in blockchain network has size limit for the sake of communication efficiency, e.g., once the payload size of the cids exceeds about 105 KBytes in our testing, the replication usually fails. (ii) With increased number of orderer nodes that can tolerate more faulty nodes, the throughput is slightly decreased. This is reasonable since more orderer nodes reaching consensus would cause more communication latency. (iii) The throughput can reach around 700 Kbytes/s (or higher with more engineering optimizations) for replicating content ids to blockchain network. Though such a throughput is relatively slower than a distributed database-enabled system, e.g., the throughput for HBase is about 5Mbytes/s [11], yet the advantage lies in the robustness assurance, as characterized by the aforementioned security properties.

3.5 Summary

In this chapter, we initiated the study of robust and automated cyber security management (CSM). This includes the formulation of three classes of CSM functions in relation to cyber threat intelligence sharing and a detailed description of the design of blockchain-based robust and automated CSM (B2CSM). We presented the implementation of a prototype B2CSM system. Real-world cyber data based experimental results show that our system is useful in practice. Noticeably, our designs can be extended to any information management system where blockchain empowers the robust decentralized storage.

CHAPTER 4

FAIR P2P CONTENT DELIVERY VIA BLOCKCHAIN

This chapter presents the design of blockchain-empowered computation in the specific peer-to-peer (P2P) content delivery setting [61].

4.1 Introduction

The peer-to-peer (P2P) content delivery systems are permissionless decentralized services to seamlessly replicate contents to the end consumers. Typically these systems [28,87] encompass a large ad-hoc network of deliverers such as normal Internet users or small organizations, thus overcoming the bandwidth bottleneck of the original content providers. In contrast to giant pre-planned content delivery networks (i.e., CDNs such as Akamai [3] and CloudFlare [27]), P2P content delivery can crowdsource unused bandwidth resources of tremendous Internet peers, thus having a wide array of benefits including robust service availability, bandwidth cost savings, and scalable peak-demand handling [5, 10].

Recently, renewed attentions to P2P content delivery are gathered [5, 55, 143] due to the fast popularization of decentralized storage networks (DSNs) [14,41,47,106, 137]. Indeed, DSNs feature decentralized and robust *content storage*, but lack well-designed *content delivery* mechanisms catering for a prosperous content consumption market in the P2P setting, where the content shall not only be reliably stored but also must be always quickly *retrievable* despite potentially malicious participants [48,144].

The primary challenge of designing a proper delivery mechanism for complementing DSNs is to realize the strict guarantee of "fairness" against adversarial peers. In particular, a fair P2P content delivery system has to promise well-deserved items (e.g., retrieval of valid contents, well-paid rewards to spent bandwidth) to all participants [43]. Otherwise, free-riding parties can abuse the system [44, 94, 119] and

cause rational ones to escape, eventually resulting in possible system collapse [59]. We reason as follows to distinguish two types of quintessential fairness, namely *delivery fairness* and *exchange fairness*, in the P2P content delivery setting where three parties, i.e., content *provider*, content *deliverer* and content *consumer*, are involved.

Exchange fairness is not delivery fairness. Exchange fairness [12, 17, 31, 37, 88, 101], specifically for digital goods (such as signatures and videos), refers to ensuring one party's input keep *confidential* until it does learn the other party's input. Unfortunately, in the P2P content delivery setting, it is insufficient to merely consider exchange fairness because a content deliverer would expect to receive rewards proportional to the bandwidth resources it spends. Noticeably, exchange fairness fails to capture such new desiderata related to bandwidth cost, as it does not rule out that a deliverer may receive no reward after transferring a huge amount of *encrypted* data to the other party, which clearly breaks the deliverer's expectation on being well-paid but does not violate exchange fairness at all.

Consider FairSwap [37] as a concrete example: the deliverer first sends the encrypted content and semantically secure digest to the consumer, then waits for a confirmation message from the consumer (through the blockchain) to confirm her receiving of these ciphertext, so the deliverer can reveal his encryption key to the content consumer via the blockchain; but, in case the consumer aborts, all bandwidth used to send ciphertext is wasted, causing no reward for the deliverer. A seemingly enticing way to mitigate the above attack on delivery fairness in FairSwap could be splitting the content into n smaller chunks and run FairSwap protocols for each chunk, but the on-chain cost would grow linear in n, resulting in prohibitive on-chain cost for large contents such as movies. Adapting other fair exchange protocols for delivery fairness would encounter similar issues like FairSwap. Hence, the efficient construction satisfying delivery fairness remains unclear.

To capture the "special" exchanged item for deliverers, we formulate delivery fairness, stating that deliverers can receive rewards (nearly) proportional to the contributed bandwidth for delivering data to the consumers.

Insufficiencies of existing "delivery fairness". A range of existing literature [75, 92, 130–132] involve delivery fairness for P2P delivery. However, to our knowledge, no one assures delivery fairness in the *cryptographic* sense, as we seek to do. Specifically, they [75, 92, 130–132] are presented in *non-cooperative game-theoretic* settings where independent attackers free ride spontaneously without communication of their strategies, and the attackers are rational with the intentions to maximize their own benefits. Therefore, these works boldly ignore that the adversary intends to break the system. Unfortunately, such rational assumptions are particularly elusive to stand in ad-hoc open systems accessible by all malicious evils. The occurrences of tremendous real-world attacks in ad-hoc open systems [40,102] hint us how vulnerable the prior studies' heavy assumptions can be and further weaken the confidence of using them in real-world P2P content delivery.

Lifting for "exchange fairness" between provider and consumer. Besides the natural delivery fairness, it is equally vital to ensure exchange fairness for providers and consumers in a basic context of P2P content delivery, especially with the end goal to complement DSNs and enable some content providers to sell contents to consumers with delegating costly delivery/storage to a P2P network. In particular, the content provider should be guaranteed to receive payments proportional to the amount of correct data learned by the consumer; vice versa, the consumer only has to pay if indeed receiving qualified content.

Naïve attempts of tuning a fair exchange protocol [12, 13, 37, 39, 88, 101] into P2P content delivery can guarantee neither delivery fairness (as analyzed earlier) nor exchange fairness: simply running fair exchange protocols twice between the deliverers and the content providers and between the deliverers and the consumers, respectively,

would leak valuable contents, raising the threat of massive content leakage. Even worse, this idea disincentivizes the deliverers as they have to pay for the whole content before making a life by delivering the content to consumers.

Our contributions. Overall, it remains an open problem to realize such strong fairness guarantees in P2P content delivery to protect *all* providers, deliverers, and consumers. We for the first time formalize such security intuitions into a well-defined cryptographic problem on fairness, and present a couple of efficient blockchain-based protocols to solve it. In sum, our contributions are:

1. *Formalizing P2P content delivery with delivery fairness.* We formulate the P2P content delivery problem with desired security goals, where fairness ensures that every party is fairly treated even others arbitrarily collude or are corrupted.

2. *Verifiable fair delivery.* We put forth a novel delivery fairness notion between a sender and a receiver dubbed verifiable fair delivery (VFD): a non-interactive honest verifier can check if a sender indeed sends a sequence of valid data chunks to a receiver as long as not both the sender and the receiver are corrupted.

   This primitive is powerful in the sense that: (i) the verifier only has to be non-interactive and honest, so it can be easily instantiated via the blockchain; (ii) qualified data can be flexibly specified through a global predicate known by the sender, the receiver and the verifier, so the predicate validation can be tuned to augment VFD in a certain way for the full-fledged P2P delivery scenario.

3. *Lifting* VFD *for full-fledged P2P content delivery.* We specify VFD to validate that each data chunk is signed by the original content provider, and wrap up the concrete instantiation to design an efficient blockchain-enabled fair P2P content delivery protocol FairDownload, which allows: (i) the consumer can retrieve content via downloading, i.e., *view-after-delivery*; (ii) minimal involvement of the content provider in the sense that only two messages are needed from the provider during the whole course of the protocol execution; (iii) one-time contract deployment and preparation while repeatable delivery of the same content to different consumers.

   Thanks to the carefully instantiated VFD, the provider's content cannot be modified by the deliverer, so we essentially can view the fairness of consumer and provider as a fair exchange problem for digital goods between two parties. To facilitate the "two-party" exchange fairness, we leverage the proof-of-misbehavior method (instead of using heavy cryptographic proofs for honesty [101]), thus launching a simple mechanism to allow the consumer to dispute and prove that the provider indeed sells wrong content inconsistent to a certain digest; along the way, we dedicatedly tune this component for

better efficiency: (i) universal composability security [37] is explicitly given up to employ *one-way security* in the stand-alone setting; (ii) the generality of supporting any form of dispute on illegitimate contents [37] is weaken to those inconsistent to digest in form of Merkle tree root.

4. *Less latency for streaming delivery.* Though the protocol FairDownload is efficient as well as minimize the provider's activities, it also incurs considerable latency since the consumer can get the content only after all data chunks are delivered. To accommodate the streaming scenario where the consumer can *view-while-delivery*, we propose another simple while efficient protocol FairStream, where each data chunk can be retrieved in $O(1)$ communication rounds. Though the design requires more involvement of the provider, whose overall communication workload, however, is much smaller than the content itself. FairStream realizes the same security goals as the FairDownload protocol.

5. *Optimal on-chain and delivery complexities.* Both the downloading and streaming protocols achieve asymptotically optimal $\tilde{O}(\eta + \lambda)$ on-chain computational costs even when dispute occurs. The on-chain costs only relates to the small chunk size parameter η and the even smaller security parameter λ. This becomes critical to preserve low-cost of blockchain-based p2p content delivery. Moreover, in both protocols, the deliverer only sends $O(\eta+\lambda)$ bits amortized for each chunk. Considering the fact that λ is much less than η, this corresponds to asymptotically optimal deliverer communication, and is the key to keep P2P downloading and P2P streaming highly efficient.

6. *Optimized implementations.* We implement[1] and optimize FairDownload and FairStream. Various non-trivial optimizations are performed to improve the on-chain performance including efficient on-chain implementation of ElGamal verifiable decryption over bn-128 curve. Extensive experiments are also conducted atop Ethereum Ropsten network, showing real-world applicability.

4.2 Preliminaries

Notations and abbreviations. The notations and abbreviations used throughout the dissertation are as follows:

- *Security parameter.* All cryptographic algorithms are parameterized by a *security parameter* $\lambda \in \mathbb{N}$ given (sometimes implicitly) to the algorithms.
- *Integer set.* By $[n]$ it denotes the set of integers $\{1, \ldots, n\}$, and by $[a, b]$ it denotes the set of integers $\{a, \ldots, b\}$.
- *String concatenation.* By $x||y$ it means a string concatenating strings x and y.

[1]Code availability: https://github.com/Blockchain-World/FairThunder.git

- *Uniform random sampling.* By $\leftarrow_{\$}$ it denotes uniformly random sampling.
- *Probabilistic and deterministic algorithms.* We use $y \leftarrow A(x)$ to denote that the output y is generated by probabilistic algorithm A using internal randomness r; In addition, we use $y := A(x)$ to denote that the output y is produced by the deterministic algorithm A. With explicit randomness r, it would be denoted with $y := A(r, x)$.
- *Prefix relationship.* By $\preceq$ it denotes the prefix relationship.
- *Abbreviations.* P.P.T refers to probabilistic algorithms with a polynomial running time; ITM is short for interactive Turing machine. See details in [22].

Symmetric encryption. A *semantically secure* (fixed-length) symmetric encryption scheme SE is made of $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, where the key generation algorithm $k \leftarrow_{\$} \mathsf{KeyGen}(\lambda)$ takes as input the security parameter λ and generates a key k; the probabilistic encryption algorithm $c \leftarrow \mathsf{Enc}(m, k)$ takes as input the key k and the message m and outputs the ciphertext c; the deterministic decryption algorithm $m := \mathsf{Dec}(c, k)$ takes as input the ciphertext c and the key k and outputs the recovered the message m.

Public key encryption. A public key encryption scheme PKE consists of three algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ where the key generation algorithm $(pk, sk) \leftarrow_{\$} \mathsf{KeyGen}(\lambda)$ takes as input the security parameter λ and outputs a pair of secret key sk and public key pk; the probabilistic encryption algorithm $c \leftarrow \mathsf{Enc}(pk, m)$ takes as input the public key pk and the message m and outputs the ciphertext c; the deterministic decryption algorithm $m := \mathsf{Dec}(c, sk)$ takes as input the ciphertext c and the secret key sk and recovers the message m. The PKE scheme satisfies the standard *correctness* and *semantic security* properties [53].

Digital signature. An *existential unforgeability under chosen message attack* (EU-CMA) secure digital signature scheme [54] SIG contains algorithms $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$: $\mathsf{SIG.KeyGen}(\lambda) \rightarrow (pk, sk)$. The key generation algorithm takes as input the security parameter λ and outputs a pair of public key pk and secret key sk; $\mathsf{SIG.Sign}(sk, m) \rightarrow \sigma$. The signing algorithm takes as input the secret key sk and

the message m and produces the signature σ; $\mathsf{SIG.Verify}(pk, m, \sigma) \rightarrow \{0, 1\}$. This deterministic verification algorithm takes as input the public key pk, the message m and the signature σ and outputs a boolean 1 or 0 indicating whether σ is valid on m relative to pk or not.

Merkle tree. This consists of a tuple of algorithms $(\mathsf{BuildMT}, \mathsf{GenMTP}, \mathsf{VerifyMTP})$. $\mathsf{BuildMT}$ accepts as input a sequence of elements $m = (m_1, m_2, \cdots, m_n)$ and outputs the Merkle tree MT with root that commits m. Note that we let $\mathsf{root}(\mathsf{MT})$ to denote the Merkle tree MT's root. GenMTP takes as input the Merkle tree MT (built for m) and the i-th element m_i in m, and outputs a proof π_i to attest the inclusion of m_i at the position i of m. $\mathsf{VerifyMTP}$ takes as input the root of Merkle tree MT, the index i, the Merkle proof π_i, and m_i, and outputs either 1 (accept) or 0 (reject). The security of Merkle tree scheme ensures that: for any P.P.T. adversary $\mathcal{A}$, any sequence m and any index i, conditioned on MT is a Merkle tree built for m, $\mathcal{A}$ cannot produce a fake Merkle tree proof fooling $\mathsf{VerifyMTP}$ to accept $m'_i \neq m_i \in m$ except with negligible probability given m, MT and security parameters.

Verifiable decryption. We consider a specific verifiable public key encryption (VPKE) scheme consisting of $(\mathsf{VPKE.KGen}, \mathsf{VEnc}, \mathsf{VDec}, \mathsf{ProvePKE}, \mathsf{VerifyPKE})$ and allowing the decryptor to produce the plaintext along with a proof attesting the correct decryption [21]. Specifically, KGen outputs a public-private key pair, i.e., $(h, k) \leftarrow \mathsf{VPKE.KGen}(1^\lambda)$ where λ is a security parameter. The public key encryption satisfies semantic security. Furthermore, for any $(h, k) \leftarrow \mathsf{VPKE.KGen}(1^\lambda)$, the $\mathsf{ProvePKE}_k$ algorithm takes as input the private key k and the cipher c, and outputs a message m with a proof π; while the $\mathsf{VerifyPKE}_h$ algorithm takes as input the public key h and (m, c, π), and outputs 1/0 to accept/reject the statement that $m = \mathsf{VDec}_k(c)$. Besides the semantic security, the verifiable decryption scheme need satisfy the following extra properties:

Table 4.1 Comparison of Different Related Representative Approaches

Schemes	Exchange (Incentive)	Delivery Fairness	Confidentiality	Exchange Fairness	On-chain Costs
BitTorrent [28]	Files ↔ Files (Tit-for-Tat)	×	×	Not fully	n/a
Dandelion [132]	Files ↔ Credits (Reputation)	×	×	Not fully	n/a
T-Chain [131]	Files ↔ Files (Tit-for-Tat)	×	√	Not fully	n/a
Gringotts [55]	Bandwidth ↔ Coins (Monetary)	*multiple* chunks	×	×	$O(n)$
CacheCash [5]	Bandwidth ↔ Coins (Monetary)	all chunks	×	×	$[o(1), O(n)]$
Our Protocols	Bandwidth/Files ↔ Coins (Monetary)	*one* chunk	√	√	$\tilde{O}(1)$

- *Completeness.* $Pr[\mathsf{VerifyPKE}_h(m, c, \pi) = 1 | (m, \pi) \leftarrow \mathsf{ProvePKE}_k(c)] = 1$, for $\forall\ c$ and $(h, k) \leftarrow \mathsf{KGen}(1^\lambda)$;
- *Soundness.* For any $(h, k) \leftarrow \mathsf{KGen}(1^\lambda)$ and c, no probabilistic poly-time (P.P.T.) adversary $\mathcal{A}$ can produce a proof π fooling $\mathsf{VerifyPKE}_h$ to accept that c is decrypted to m' if $m' \neq \mathsf{VDec}_k(c)$ except with negligible probability;
- *Zero-Knowledge.* The proof π can be simulated by a P.P.T. simulator $\mathcal{S}_{\mathsf{VPKE}}$ taking as input only public knowledge m, h, c, hence nothing more than the truthness of the statement $(m, c) \in \{(m, c) | m = \mathsf{VDec}_k(c)\}$ is leaked.

4.3 Related Work

Here we review the pertinent technologies and discuss their insufficiencies in the specific context of P2P content delivery. Table 4.1 summarizes the advantages provided by our protocol when compared with other representative related works.

P2P information exchange schemes. Many works [28, 75, 92, 119, 130–132] focused on the basic challenge to incentivize users in the P2P network to voluntarily

exchange information. However, these schemes have not been notably successful in combating free-riding problem and strictly ensuring the fairness. Specifically, the schemes in BitTorrent [28], BitTyrant [119], FairTorrent [130], PropShare [92] support direct reciprocity (i.e., the willingness for participants to continue exchange basically depends on their past direct interactions, e.g., the *Tit-for-Tat* mechanism in BitTorrent) for participants, which cannot accommodate the *asymmetric* interests (i.e., participants have distinct types of resources such as bandwidth and cryptocurrencies to trade between each others) in the P2P content delivery setting. For indirect reciprocity (e.g., reputation, currency, credit-based) mechanisms including Eigentrust [75], Dandelion [132], they suffer from Sybil attacks, e.g., a malicious peer could trivially generate a sybil peer and "deliver to himself" and then rip off the credits. We refer readers to [131] for more discussions about potential attacks to existing P2P information exchange schemes. For T-chain [131], it still considers rational attackers and cannot strictly ensure the delivery fairness as an adversary can waste a lot of bandwidth of deliverers though the received content is encrypted.

More importantly, all existing schemes, to our knowledge, are presented in the non-cooperative game-theoretic setting, which means that they only consider independent attackers free ride spontaneously without communication of their strategies, and the attackers are rational with the intentions to maximize their own benefits. However, such rational assumptions are elusive to guarantee the fairness for parties in the ad-hoc systems accessible by all malicious entities. Our protocol, on the contrary, assures the delivery fairness in the cryptographic sense. Overall, our protocol can rigorously guarantee fairness for all participating parties, i.e., deliverers with delivery fairness, providers and consumers with exchange fairness. Also, the fairness in the P2P information exchange setting is typically measured due to the discrepancy between the number of pieces uploaded and received over a long period [72] for a participant. If we examine each concrete delivery session, there is

no guarantee of fairness. This further justifies that the P2P information exchange schemes are not directly suitable for the specific P2P content delivery setting.

Fair exchange and fair MPC. There are also intensive works on fair exchange protocols in cryptography. It is well-known that a fair exchange protocol cannot be designed to provide complete exchange fairness without a trusted third party (TTP) [116], which is a specific impossible result of the general impossibility of fair multi-party computation (MPC) without honest majority [26]. Some traditional ways hinge on a TTP [12, 13, 88, 104] to solve this problem, which has been reckon hard to find such a TTP in practice. To avoid the available TTP requirement, some other studies [17, 31, 51, 120] rely on the "gradual release" approach, in which the parties act in turns to release their private values bit by bit, such that even if one malicious party aborts, the honest party can recover the desired output by investing computational resources (in form of CPU time) comparable to that of the adversary uses. Recently, the blockchain offers an attractive way to instantiate a non-private TTP, and a few results [15, 25, 37, 39, 81, 101] leverage this innovative decentralized infrastructure to facilitate fair exchange and fair MPC despite the absence of honest majority. Unfortunately, all above fair exchange and fair MPC protocols fail to guarantee *delivery fairness* in the specific P2P content delivery setting, as they cannot capture the fairness property for the special exchanged item (i.e., bandwidth).

State channels. A state channel establishes a private P2P medium, managed by pre-set rules, allowing the involved parties to update state unanimously by exchanging authenticated state transitions off-chain [57]. Though our protocols can be reckon as the application of payment channel networks (PCNs) (or more general state channels [105]) yet there are two key differences: i) fairness in state channels indicates that an honest party (with valid state transition proof) can always withdraw the agreed balance from the channel [57], while our protocols, dwelling on the delivery fairness in the specific context of P2P content delivery, ensure the

bandwidth contribution can be quantified and verified to generate such a valid state transition proof; ii) state channels essentially allow any two parties to interact, while our protocols target the interaction among any three parties with a totally different payment paradigm [5] for P2P content delivery.

Decentralized content delivery. There exist some systems that have utilized the idea of exchanging bandwidth for rewards to incentivize users' availability or honesty such as Dandelion [132], Floodgate [109]. However, different drawbacks impede their practical adoption, as discussed in [5]. Here we elaborate the comparison with two protocols, i.e., Gringotts [55], CacheCash [5], that target the similar P2P content delivery scenario.

Application Scenario. Typically, the P2P content delivery setting involves asymmetric exchange interests of participants, i.e., the consumers expect to receive a specific content identified by a certain digest in time, while the providers and the deliverers would share their content (valid due to the digest) and bandwidth in exchange of well-deserved payments/credits, respectively. Unfortunately, Gringotts and CacheCash fail to capture this usual scenario, and cannot support the content providers to sell content over the P2P network, due to the lack of content confidentiality and exchange fairness. In greater detail, both Gringotts and CacheCash delegate a copy of raw content to the deliverers, which results in a straightforward violation of exchange fairness, i.e., a malicious consumer can pretend to be or collude with a deliverer to obtain the plaintext content without paying for the provider.

Delivery Fairness. Gringotts typically requires the deliverer to receive a receipt (for acknowledging the resource contribution) only after multiple chunks are delivered, which poses the risk of losing bandwidth for delivering multiple chunks. For CacheCash, a set of deliverers are selected to distribute the chunks in parallel, which may cause the loss of bandwidth for all chunks in the worst case. Our protocols ensures that the unfairness of delivery is bounded to one chunk of size η.

On-chain Costs. Gringotts stores all chunk delivery records on the blockchain, and therefore the on-chain costs is in $O(n)$. While for CacheCash, the deliverers can obtain *lottery tickets* (i.e., similar to "receipts") from the consumer after each "valid" chunk delivery. The on-chain costs is highly pertinent to the winning probability p of tickets. For example, $p = \frac{1}{n}$ means that on average the deliverer owns a winning ticket after n chunks are delivered, or $p = 1$ indicates that the deliverer receives a winning ticket after each chunk delivery, leading to at most $O(n)$ on-chain costs of handling redeem transactions. In our protocols, the on-chain costs is bounded to $\tilde{O}(1)$.

4.4 Warm-Up: Verifiable Fair Delivery

We first warm up and set forth a building block termed *verifiable fair delivery* (VFD), which enables an honest verifier to verify that a sender indeed transfers some amount of data to a receiver. It later acts as a key module in the fair P2P content delivery protocol. The high level idea of VFD lies in: the receiver needs to send back a signed "receipt" in order to acknowledge the sender's bandwidth contribution and continuously receives the next data chunk. Consider that the data chunks of same size η are transferred *sequentially* starting from the first chunk, the sender can always use the latest receipt containing the chunk index to prove to the verifier about the total contribution. Intuitively the sender *at most* wastes bandwidth of transferring one chunk.

Syntax of VFD. The VFD protocol is among an interactive poly-time Turing-machine (ITM) sender denoted by $\mathcal{S}$, an ITM receiver denoted by $\mathcal{R}$, and a non-interactive Turing-machine verifier denoted by $\mathcal{V}$, and follows the syntax:

- **Sender.** The sender $\mathcal{S}$ can be activated by calling an interface $\mathcal{S}.\mathsf{send}()$ with inputting a sequence of n data chunks and their corresponding validation strings denoted by $((c_1, \sigma_{c_1}), \ldots, (c_n, \sigma_{c_n}))$, and there exists an efficient and global predicate $\Psi(i, c_i, \sigma_{c_i}) \rightarrow \{0, 1\}$ to check whether c_i is the i-th valid chunk due to σ_{c_i}; once activated, the sender $\mathcal{S}$ interacts with the receiver $\mathcal{R}$, and opens an interface $\mathcal{S}.\mathsf{prove}()$ that can be invoked to return a proof string π indicating the number of sent chunks;

- **Receiver.** The receiver $\mathcal{R}$ can be activated by calling an interface $\mathcal{R}.\mathsf{recv}()$ with taking as input the description of the global predicate $\Psi(\cdot)$ to interact with $\mathcal{S}$, and outputs a sequence of $((c_1, \sigma_{c_1}), \ldots, (c_{n'}, \sigma_{c_{n'}}))$, where $n' \in [n]$ and every (c_i, σ_{c_i}) is valid due to $\Psi(\cdot)$;
- **Verifier.** The verifier $\mathcal{V}$ inputs the proof π generated by $\mathcal{S}.\mathsf{prove}()$, and outputs an integer $\mathsf{ctr} \in \{0, \cdots, n\}$.

Security of VFD. The VFD protocol must satisfy the following security properties:

- **Termination**. If at least one of $\mathcal{S}$ and $\mathcal{R}$ is honest, the VFD protocol terminates within *at most* $2n$ rounds, where n is the number of content chunks;
- **Completeness**. If $\mathcal{S}$ and $\mathcal{R}$ are both honest and activated, after $2n$ rounds, $\mathcal{S}$ is able to generate a proof π which $\mathcal{V}$ can take as input and output $\mathsf{ctr} \equiv n$, while $\mathcal{R}$ can output $((c_1, \sigma_{c_1}), \ldots, (c_n, \sigma_{c_n}))$, which is same to $\mathcal{S}$'s input;
- **Verifiable η delivery fairness**. When one of $\mathcal{S}$ and $\mathcal{R}$ maliciously aborts, VFD shall satisfy the following delivery fairness requirements:
  - *Verifiable delivery fairness against $\mathcal{S}^*$.* For any corrupted P.P.T. sender $\mathcal{S}^*$ controlled by $\mathcal{A}$, it is guarantee that: the honest receiver $\mathcal{R}$ will always receive the valid sequence $(c_1, \sigma_{c_1}), \ldots, (c_{\mathsf{ctr}}, \sigma_{c_{\mathsf{ctr}}})$ if $\mathcal{A}$ can produce the proof π that enables $\mathcal{V}$ to output ctr.
  - *Verifiable delivery fairness against $\mathcal{R}^*$.* For any corrupted P.P.T. receiver $\mathcal{R}^*$ controlled by $\mathcal{A}$, it is ensured that: the honest sender $\mathcal{S}$ can always generate a proof π, which enables $\mathcal{V}$ to output *at least* $(\mathsf{ctr}-1)$ if $\mathcal{A}$ receives the valid sequence $(c_1, \sigma_{c_1}), \ldots, (c_{\mathsf{ctr}}, \sigma_{c_{\mathsf{ctr}}})$. At most $\mathcal{S}$ wastes bandwidth for delivering one content chunk of size η.

VFD Protocol Π_{VFD}. We consider the authenticated setting that the sender $\mathcal{S}$ and the receiver $\mathcal{R}$ have generated public-private key pairs $(pk_{\mathcal{S}}, sk_{\mathcal{S}})$ and $(pk_{\mathcal{R}}, sk_{\mathcal{R}})$ for digital signature, respectively; and they have announced the public keys to bind to themselves. Then, VFD with the global predicate $\Psi(\cdot)$ can be realized by the protocol Π_{VFD} hereunder among $\mathcal{S}$, $\mathcal{R}$ and $\mathcal{V}$ against P.P.T. and static adversary in the *stand-alone* setting[2] with the synchronous network assumption:

[2]We omit the *session id* (denoted as *sid*) in the stand-alone context for brevity. To defend against *replay* attack in concurrent sessions, it is trivial to let the authenticated messages include an *sid* field, which, for example, can be instantiated by the hash of the transferred data identifier root_m, the involved parties' addresses and an increasing-only nonce, namely $sid := \mathcal{H}(\mathsf{root}_m || \mathcal{V}\_address || pk_{\mathcal{S}} || pk_{\mathcal{R}} || nonce)$.

- **Construction of sender.** The sender $\mathcal{S}$, after activated via $\mathcal{S}.\mathsf{send}()$ with the input $((c_1, \sigma_{c_1}), \ldots, (c_n, \sigma_{c_n}))$, $pk_{\mathcal{S}}$ and $pk_{\mathcal{R}}$, starts a timer $\mathcal{T}_{\mathcal{S}}$ lasting two synchronous rounds, initializes a variable $\pi_{\mathcal{S}} := null$, and executes as follows:
  - For each $i \in [n]$: the sender sends $(\mathsf{deliver}, i, c_i, \sigma_{c_i})$ to $\mathcal{R}$, and waits for the response message $(\mathsf{receipt}, i, \sigma^i_{\mathcal{R}})$ from $\mathcal{R}$. If $\mathcal{T}_{\mathcal{S}}$ expires before receiving the response, breaks the iteration; otherwise $\mathcal{S}$ verifies whether $\mathsf{Verify}(\mathsf{receipt}||i||pk_{\mathcal{R}}||pk_{\mathcal{S}}, \sigma^i_{\mathcal{R}}, pk_{\mathcal{R}}) \equiv 1$ or not, if *true*, resets $\mathcal{T}_{\mathcal{S}}$, outputs $\pi_{\mathcal{S}} := (i, \sigma^i_{\mathcal{R}})$, and continues to run the next iteration (i.e., increasing i by one); if *false*, breaks the iteration;
  - Upon $\mathcal{S}.\mathsf{prove}()$ is invoked, it returns $\pi_{\mathcal{S}}$ as the VFD proof and halts.
- **Construction of receiver.** The receiver $\mathcal{R}$, after activated via $\mathcal{R}.\mathsf{recv}()$ with the input $pk_{\mathcal{S}}$ and $(pk_{\mathcal{R}}, sk_{\mathcal{R}})$, starts a timer $\mathcal{T}_{\mathcal{R}}$ lasting two synchronous rounds and operates as: for each $j \in [n]$: $\mathcal{R}$ waits for $(\mathsf{deliver}, j, c_j, \sigma_{c_j})$ from $\mathcal{S}$ and halts if $\mathcal{T}_{\mathcal{R}}$ expires before receiving the $\mathsf{deliver}$ message; otherwise $\mathcal{R}$ verifies whether $\Psi(j, c_j, \sigma_{c_j}) \equiv 1$ or not; if *true*, resets $\mathcal{T}_{\mathcal{R}}$, outputs (c_j, σ_{c_j}), and sends $(\mathsf{receipt}, i, \sigma^i_{\mathcal{R}})$ to $\mathcal{S}$ where $\sigma^i_{\mathcal{R}} \leftarrow \mathsf{Sign}(\mathsf{receipt}||i||pk_{\mathcal{R}}||pk_{\mathcal{S}}, sk_{\mathcal{R}})$, halts if *false*. Note that the global predicate $\Psi(\cdot)$ is efficient as essentially it just performs a signature verification.
- **Construction of verifier**. Upon the input $\pi_{\mathcal{S}}$, the verifier $\mathcal{V}$ parses it into $(\mathsf{ctr}, \sigma^{\mathsf{ctr}}_{\mathcal{R}})$, and checks whether $\mathsf{Verify}(\mathsf{receipt}||\mathsf{ctr}||pk_{\mathcal{R}}||pk_{\mathcal{S}}, \sigma^{\mathsf{ctr}}_{\mathcal{R}}, pk_{\mathcal{R}}) \equiv 1$ or not; if *true*, it outputs ctr, or else outputs 0. Recall that Verify is to verify signatures.

Lemma 1. *In the synchronous authenticated network and stand-alone setting, the protocol Π_{VFD} satisfies termination, completeness and the verifiable η delivery fairness against non-adaptive P.P.T. adversary corrupting one of the sender and the receiver.*

Proof. If both the sender and the receiver are honest, there would be $2n$ communication rounds since for every delivered chunk, the sender obtains a "receipt" from the receiver for acknowledging bandwidth contribution. If one malicious party aborts, the other honest one would also terminate after its maintained timer expires, resulting in less than $2n$ communication rounds. Therefore, the termination property is guaranteed.

In addition, when both the sender $\mathcal{S}$ and the receiver $\mathcal{R}$ are honest, the *completeness* of VFD is immediate to see: in each round, $\mathcal{S}$ would start to deliver the next chunk after receiving the receipt from $\mathcal{R}$ within 2 rounds, i.e., a

round-trip time. After $2n$ synchronous rounds, $\mathcal{R}$ receives the chunk-validation pairs $((c_1, \sigma_{c_1}), \cdots, (c_n, \sigma_{c_n}))$ and $\mathcal{S}$ outputs the last receipt as a proof π, which is taken as input by the verifier $\mathcal{V}$ to output n demonstrating $\mathcal{S}$'s delivery contribution.

For the η delivery fairness of VFD, on one hand, the malicious $\mathcal{S}^*$ corrupted by $\mathcal{A}$ may abort after receiving the ctr-th ($1 \leq \mathsf{ctr} \leq n$) receipt. In that case, $\mathcal{R}$ is also guaranteed to receive a valid sequence of $((c_1, \cdots, \sigma_{c_1}), \cdots, (c_{\mathsf{ctr}}, \sigma_{c_{\mathsf{ctr}}}))$ with overwhelming probability, unless $\mathcal{A}$ can forge $\mathcal{R}$'s signature. However, it requires $\mathcal{A}$ to break the underlying EU-CMA signature scheme, which is of negligible probability. On the other hand, for the malicious $\mathcal{R}^*$ corrupted by $\mathcal{A}$, if $\mathcal{V}$ takes the honest $\mathcal{S}$'s proof and can output ctr, then $\mathcal{S}$ *at most* has sent $(\mathsf{ctr}+1)$ chunk-validation pairs, i.e., (c_i, σ_{c_i}), to $\mathcal{A}$. Overall, $\mathcal{S}$ *at most* wastes bandwidth of delivering one chunk of size η. Hence, the η delivery fairness of VFD is rigorously guaranteed.

4.5 Formalizing P2P Content Delivery

4.5.1 System Model

Participating parties. We consider the following explicit entities (i.e., interactive Turing machines by cryptographic convention) in the context of P2P content delivery:

- *Content Provider* is an entity (denoted by $\mathcal{P}$) that owns the original content m composed of n data chunks,[3] satisfying a public known predicate $\phi(\cdot)$,[4] and $\mathcal{P}$ is willing to sell to the users of interest. Meanwhile, the provider would like to delegate the delivery of its content to a third-party (viz. a deliverer) with promise to pay $\ddot{\mathbb{B}}_{\mathcal{P}}$ for each successfully delivered chunk.
- *Content Deliverer* (denoted by $\mathcal{D}$) contributes its idle bandwidth resources to deliver the content on behalf of the content provider $\mathcal{P}$ and would receive the payment proportional to the amount of delivered data. In the P2P delivery scenario, deliverers can be some small organizations or individuals, e.g., the *RetrievalProvider* in Filecoin [48].

[3]Remark that the content m is *dividable* in the sense that each chunk is independent to other chunks, e.g., each chunk is a small 10-second video.

[4]Throughout the study, we consider that the predicate ϕ is in the form of $\phi(m) = [\mathsf{root}(\mathsf{BuildMT}(m)) \equiv \mathsf{root}_m]$, where root is the Merkle tree root of the content m. In practice, it can be aquired from a semi-trusted third party, such as BitTorrent forum sites [88] or VirusTotal [70].

- *Content Consumer* is an entity (denoted by $\mathcal{C}$) that would pay $\ddot{\mathbb{B}}_{\mathcal{C}}$ for each chunk in the content m by interacting with $\mathcal{P}$ and $\mathcal{D}$.

Adversary. Following modern cryptographic practices [78], we consider the adversary $\mathcal{A}$ with following standard abilities:

- *Static corruptions*. The adversary $\mathcal{A}$ can corrupt some parties only before the course of protocol executions;
- *Computationally bounded*. The adversary $\mathcal{A}$ is restricted to P.P.T. algorithms;
- *Synchronous authenticated channel*. We adopt the synchronous network model of authenticated point-to-point channels to describe the ability of $\mathcal{A}$ on controlling communications, namely, for any messages sent between honest parties, $\mathcal{A}$ is consulted to delay them up to a-priori known Δ but cannot drop, reroute or modify them. W.l.o.g., we consider a global clock in the system, and $\mathcal{A}$ can delay the messages up to a clock round [81, 84].

Arbiter Smart Contract $\mathcal{G}$. The system is in a hybrid model with oracle access to an arbiter smart contract $\mathcal{G}$. The contract $\mathcal{G}$ is a stateful ideal functionality that leaks all its internal states to the adversary $\mathcal{A}$ and all parties, while allowing to pre-specify some immutable conditions (that can be triggered through interacting with $\mathcal{P}$, $\mathcal{D}$, and $\mathcal{C}$) to transact "coins" over the cryptocurrency ledger, thus "mimicking" the contracts in real life transparently. In practice, the contract can be instantiated through many real-world blockchains such as Ethereum [148]. Throughout this study, the details of the arbiter contracts $\mathcal{G}$ follow the conventional pseudo-code notations in the seminal work due to Kosba et al. [84].

4.5.2 Design Goals

Now we formulate the problem of fair content delivery with an emphasis on the delivery fairness, which to our knowledge is the first formal definition to abstract the necessary security/utility requirements of delegated P2P content delivery.

Syntax. A fair P2P content delivery protocol $\Pi = (\mathcal{P}, \mathcal{D}, \mathcal{C})$ is a tuple of three P.P.T. interactive Turing machines (ITMs) consisting of two explicit phases:

- *Preparation phase.* The provider $\mathcal{P}$ takes as input public parameters and the content $m = (m_1, \ldots, m_n) \in \{0,1\}^{\eta \times n}$ that satisfies $\phi(m) \equiv 1$, where η is chunk size in bit and n is the number of chunks and it outputs some auxiliary data, e.g., encryption keys; the deliverer $\mathcal{D}$ takes as input public parameters and outputs some auxiliary data, e.g., encrypted content; the consumer $\mathcal{C}$ does not involve in this phase. Note $\mathcal{P}$ deposits a budget of $n \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$ in ledger to incentivize $\mathcal{D}$ so it can *minimize* bandwidth usage in the next phase.
- *Delivery phase.* The provider $\mathcal{P}$ and the deliverer $\mathcal{D}$ take as input their auxiliary data obtained in the preparation phase, respectively, and they would receive the deserved payment; the consumer $\mathcal{C}$ takes as input public parameters and outputs the content m with $\phi(m) \equiv 1$. Note $\mathcal{C}$ has a budget of $n \cdot \ddot{\mathbb{B}}_{\mathcal{C}}$ in ledger to "buy" the content m satisfying $\phi(m) \equiv 1$, where $\ddot{\mathbb{B}}_{\mathcal{C}} > \ddot{\mathbb{B}}_{\mathcal{P}}$.

Furthermore, the fair P2P content delivery protocol Π shall meet the following security requirements.

Completeness. For any content predicate $\phi(\cdot)$ with $\phi(m) = [\mathsf{root}(\mathsf{BuildMT}(m)) \equiv \mathsf{root}_m]$, conditioned on $\mathcal{P}, \mathcal{D}$ and $\mathcal{C}$ are all honest, the protocol Π attains:

- The consumer $\mathcal{C}$ would obtain the qualified content m satisfying $\phi(m) \equiv 1$, and its balance in the global $\mathsf{ledger}[\mathcal{C}]$ would decrease by $n \cdot \ddot{\mathbb{B}}_{\mathcal{C}}$, where $\ddot{\mathbb{B}}_{\mathcal{C}}$ represents the amount paid by $\mathcal{C}$ for each content chunk.
- The deliverer $\mathcal{D}$ would receive the payment $n \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$ over the global ledger, where $\ddot{\mathbb{B}}_{\mathcal{P}}$ represents the amount paid by $\mathcal{P}$ to $\mathcal{D}$ for delivering a content chunk to the consumer.
- The provider $\mathcal{P}$ would receive its well-deserved payments over the ledger, namely, $\mathsf{ledger}[\mathcal{P}]$ would increase by $n \cdot (\ddot{\mathbb{B}}_{\mathcal{C}} - \ddot{\mathbb{B}}_{\mathcal{P}})$ as it receives $n \cdot \ddot{\mathbb{B}}_{\mathcal{C}}$ from the consumer while it pays out $n \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$ to the deliverer.

Fairness. The protocol Π shall satisfy the following fairness requirements:

- *Consumer Fairness.* For $\forall$ corrupted P.P.T. $\mathcal{D}^*$ and $\mathcal{P}^*$ (fully controlled by $\mathcal{A}$), it is guaranteed to the honest consumer $\mathcal{C}$ with overwhelming probability that: the $\mathsf{ledger}[\mathcal{C}]$ decreases by $\ell \cdot \ddot{\mathbb{B}}_{\mathcal{C}}$ only if $\mathcal{C}$ receives a sequence of chunks $(m_1, \ldots, m_\ell) \preceq m$ where $\phi(m) \equiv 1$, Intuitively, this property states that $\mathcal{C}$ pays proportional to valid chunks it *de facto* receives.
- *Delivery η-Fairness.* For $\forall$ malicious P.P.T. $\mathcal{C}^*$ and $\mathcal{P}^*$ corrupted by $\mathcal{A}$, it is assured to the honest deliverer $\mathcal{D}$ that: if $\mathcal{D}$ sent overall $O(\ell \cdot \eta + 1)$ bits during the protocol, $\mathcal{D}$ should *at least* obtain the payment of $(\ell - 1) \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$. In other words, the unpaid delivery is bounded by $O(\eta)$ bits.

- *Provider η-Fairness.* For $\forall$ corrupted P.P.T. $\mathcal{C}^*$ and $\mathcal{D}^*$ controlled by $\mathcal{A}$, it is ensured to the honest provider $\mathcal{P}$ that: if $\mathcal{A}$ can output $\eta \cdot \ell$ bits consisted in the content m, the provider $\mathcal{P}$ shall receive at least $(\ell - 1) \cdot (\ddot{\mathbb{B}}_{\mathcal{C}} - \ddot{\mathbb{B}}_{\mathcal{P}})$ net income, namely, $\mathsf{ledger}[\mathcal{P}]$ increases by $(\ell - 1) \cdot (\ddot{\mathbb{B}}_{\mathcal{C}} - \ddot{\mathbb{B}}_{\mathcal{P}})$, with all except negligible probability. i.e., $\mathcal{P}$ is ensured that *at most* $O(\eta)$-bit content are revealed without being well paid.

Confidentiality against deliverer. This is needed to protect copyrighted data against probably corrupted deliverers, otherwise a malicious consumer can pretend to be or collude with a deliverer to obtain the plaintext content without paying for the provider, which violates the exchange fairness for $\mathcal{P}$. Informally, we require that the corrupted $\mathcal{D}^*$ on receiving protocol scripts (e.g., the delegated content chunks from the provider) cannot produce the provider's input content with all but negligible probability in a delivery session.

We remark that confidentiality is not captured by fairness, as it is trivial to see a protocol satisfying fairness might not have confidentiality: upon all payments are cleared and the consumers receives the whole content, the protocol lets the consumer send the content to the deliverer.

Timeliness. When at least one of the parties $\mathcal{P}$, $\mathcal{D}$ and $\mathcal{C}$ is honest (i.e., others can be corrupted by $\mathcal{A}$), the honest ones are guaranteed to halt in $O(n)$ synchronous rounds where n is the number of content chunks. At the completion or abortion of the protocol, the aforementioned fairness and confidentiality are always guaranteed.

Non-trivial efficiency. We require the necessary non-trivial efficiency to rule out possible trivial solutions:

- The messages sent to $\mathcal{G}$ from honest parties are uniformly bounded by $\tilde{O}(1)$ bits, which excludes a trivial way of using the smart contract to directly deliver the content.
- In the delivery phase, the messages sent by honest $\mathcal{P}$ are uniformly bounded by $n \cdot \lambda$ bits, where λ is a small cryptographic parameter, thus ensuring $n \cdot \lambda$ much smaller than the size of content $|m|$. This indicates that $\mathcal{P}$ can save its bandwidth upon the completion of preparation phase and excludes the idea of delivering by itself.

Remarks. We make the following discussions about the above definitions: (i) $\phi(\cdot)$ is a public parameter known to all parties before the protocol execution; (ii) our fairness requirements have already implied the case where the adversary corrupts one party of $\mathcal{P}$, $\mathcal{D}$ and $\mathcal{C}$ instead of two, since whenever the adversary corrupts two parties, it can let one of these corrupted two follow the original protocol; (iii) like all cryptographic protocols, it does not make sense to consider all parties are corrupted, so do we not; (iv) the deliverer and the provider might lose well-deserved payment, but *at most* lose that for one chunk, i.e., the level of unfairness is strictly bounded; (v) though we focus on the case of one *single* content deliverer, our formalism and design can be extended to capture *multiple* deliverers, for example, when the whole content is cut to multiple pieces and each piece is delegated to a distinct deliverer. The extension with strong fairness guarantee forms an interesting future work.

In addition, one might wonder that a probably corrupted content provider fails in the middle of a transmission, causing that the consumer does not get the entire content but has to pay a lot. Nevertheless, this actually is not a serious worry in the peer-to-peer content delivery setting that aims to complement decentralized content storage networks because there essentially would be a large number of deliverers, and at least some of them can be honest. As such, if a consumer encounters failures in the middle of retrieving the content, it can iteratively ask another deliverer to start a new delivery session to fetch the remaining undelivered chunks. Moreover, our actual constructions in Sections 4.6 and 4.7 essentially allow the consumers to fetch the content from any specific chunk instead of beginning with the first one.

4.6 Fair P2P Downloading Protocol Design

This section presents the fair P2P downloading protocol allowing the consumers to view the content after getting (partial or all) the chunks, termed as *view-after-delivery*.

4.6.1 Protocol Overview

At a high level, our protocol Π_{FD} can be constructed around the module of verifiable fair delivery (VFD) and proceeds in *Prepare*, *Deliver* and *Reveal* phases as illustrated in Figure 4.1. The core ideas of Π_{FD} can be over-simplified as follows:

- The provider $\mathcal{P}$ encrypts each chunk, signs the encrypted chunks, and delegates to the deliverer $\mathcal{D}$; as such, the deliverer (as the sender $\mathcal{S}$) and the consumer $\mathcal{C}$ (as the receiver $\mathcal{R}$) can run a specific instance of VFD, in which the global predicate $\Psi(\cdot)$ is instantiated to verify that each chunk must be correctly signed by $\mathcal{P}$; additionally, the non-interactive honest verifier $\mathcal{V}$ in VFD is instantiated via a smart contract, hence upon the contract receives a VFD proof from $\mathcal{D}$ claiming the in-time delivery of ctr chunks, it can assert that $\mathcal{C}$ indeed received ctr encrypted chunks signed by the provider, who can then present to reveal the decryption keys of these ctr chunks (via the smart contract).
- Nevertheless, trivial disclosure of decryption keys via the contract would cause significant on-chain cost up to linear in the number of chunks; we propose a *structured key generation scheme* composed of Algorithms 13, 14, 16 that allows the honest provider to reveal all ctr decryption keys via a short $\tilde{O}(\lambda)$-bit message; furthermore, to ensure confidentiality against the deliverer, the script to reveal decryption keys is encrypted by the consumer's public key; in case the revealed keys cannot decrypt the cipher chunk signed by $\mathcal{P}$ itself to obtain the correct data chunk, the consumer can complain to the contract via a short $\tilde{O}(\eta + \lambda)$-bit message to prove the error of decrypted chunk and get refund.

The protocol design of Π_{FD} can ensure the fairness for each participating party even others are all corrupted by non-adaptive P.P.T. adversary. The on-chain cost keeps *constant* regardless of the content size $|m|$ in the optimistic mode where no dispute occurs. While in the pessimistic case, the protocol also realizes asymptotically optimal on-chain cost, which is related to the chunk size η. Moreover, the deliverer $\mathcal{D}$ can achieve asymptotically optimal communication in the sense that $\mathcal{D}$ only sends $O(\eta + \lambda)$ bits amortized for each chunk, where η is the chunk size and λ is a small security parameter with $\lambda \ll \eta$. These properties contribute significantly to the efficiency and practicability of applying Π_{FD} to the P2P content delivery setting.

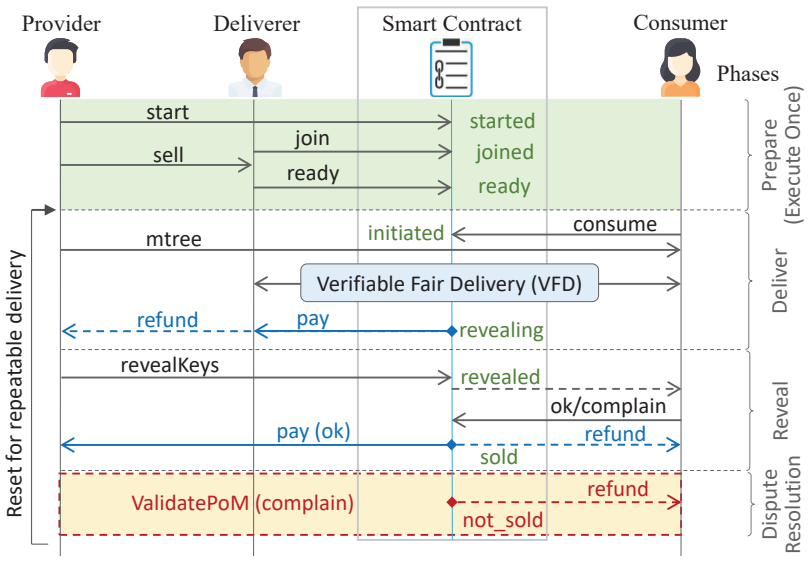


Figure 4.1 The overview of FairDownload protocol Π_{FD}.

4.6.2 Arbiter Contract for Downloading

The arbiter contract $\mathcal{G}_d^{\mathsf{ledger}}$ (abbr. $\mathcal{G}_d$) shown in Figures 4.2 and 4.3 are the stateful ideal functionality having accesses to ledger to assist the fair content delivery via downloading. We make the following remarks about the contract functionality:

- *Feasibility.* To demonstrate the feasibility of $\mathcal{G}_d$, we describe it by following the conventional pseudocode notation of smart contracts [84]. The description captures the essence of real-world smart contracts, since it: (i) reflects that the Turing-complete smart contract can be seen as a stateful program to transparently handle pre-specified functionalities; (ii) captures that a smart contract can access the cryptocurrency ledger to faithfully deal with conditional payments upon its own internal states.
- VFD.$\mathcal{V}$ *subroutine.* $\mathcal{G}_d$ can invoke the VFD verifier $\mathcal{V}$ as a subroutine. VFD's predicate $\Psi(\cdot)$ is instantiated to verify that each chunk is signed by the provider.
- ValidateRKeys *and* ValidatePoM *subroutines.* The subroutines allow the consumer to prove to the contract if the content provider $\mathcal{P}$ behaves maliciously. We defer the details to the next subsection.

The Arbiter Contract Functionality $\mathcal{G}_d^{\mathsf{ledger}}$ for P2P Downloading

The arbiter contract $\mathcal{G}_d$ has access to the ledger, and it interacts with the provider $\mathcal{P}$, the deliverer $\mathcal{D}$, the consumer $\mathcal{C}$ and the adversary $\mathcal{A}$. It locally stores the times of repeatable delivery θ, the number of content chunks n, the content digest root_m, the price $\ddot{\mathbb{B}}_{\mathcal{P}}$, $\ddot{\mathbb{B}}_{\mathcal{C}}$ and $\ddot{\mathbb{B}}_{\mathsf{pf}}$, the number of delivered chunks ctr (initialized as 0), addresses $pk_{\mathcal{P}}, pk_{\mathcal{D}}, pk_{\mathcal{C}}, vpk_{\mathcal{C}}$, revealed keys' hash erk_{hash}, state Σ and three timers $\mathcal{T}_{\mathsf{round}}$ (implicitly), $\mathcal{T}_{\mathsf{deliver}}$, and $\mathcal{T}_{\mathsf{dispute}}$.

Phase 1: Prepare

- On receive $(\mathsf{start}, pk_{\mathcal{P}}, \mathsf{root}_m, \theta, n, \ddot{\mathbb{B}}_{\mathcal{P}}, \ddot{\mathbb{B}}_{\mathcal{C}}, \ddot{\mathbb{B}}_{\mathsf{pf}})$ from $\mathcal{P}$:
  - assert $\mathsf{ledger}[\mathcal{P}] \geq (\theta \cdot (n \cdot \ddot{\mathbb{B}}_{\mathcal{P}} + \ddot{\mathbb{B}}_{\mathsf{pf}})) \wedge \Sigma \equiv \emptyset$
  - store $pk_{\mathcal{P}}, \mathsf{root}_m, \theta, n, \ddot{\mathbb{B}}_{\mathcal{P}}, \ddot{\mathbb{B}}_{\mathcal{C}}, \ddot{\mathbb{B}}_{\mathsf{pf}}$
  - let $\mathsf{ledger}[\mathcal{P}] := \mathsf{ledger}[\mathcal{P}] - \theta \cdot (n \cdot \ddot{\mathbb{B}}_{\mathcal{P}} + \ddot{\mathbb{B}}_{\mathsf{pf}})$ and $\Sigma := \mathsf{started}$
  - send $(\mathsf{started}, pk_{\mathcal{P}}, \mathsf{root}_m, \theta, n, \ddot{\mathbb{B}}_{\mathcal{P}}, \ddot{\mathbb{B}}_{\mathcal{C}}, \ddot{\mathbb{B}}_{\mathsf{pf}})$ to all entities
- On receive $(\mathsf{join}, pk_{\mathcal{D}})$ from $\mathcal{D}$:
  - assert $\Sigma \equiv \mathsf{started}$
  - store $pk_{\mathcal{D}}$ and let $\Sigma := \mathsf{joined}$
  - send $(\mathsf{joined}, pk_{\mathcal{D}})$ to all entities
- On receive $(\mathsf{prepared})$ from $\mathcal{D}$:
  - assert $\Sigma \equiv \mathsf{joined}$, and let $\Sigma := \mathsf{ready}$
  - send (ready) to all entities

Phase 2: Deliver

- On receive $(\mathsf{consume}, pk_{\mathcal{C}}, vpk_{\mathcal{C}})$ from $\mathcal{C}$:
  - assert $\theta > 0$
  - assert $\mathsf{ledger}[\mathcal{C}] \geq n \cdot \ddot{\mathbb{B}}_{\mathcal{C}} \wedge \Sigma \equiv \mathsf{ready}$
  - store $pk_{\mathcal{C}}$, $vpk_{\mathcal{C}}$ and let $\mathsf{ledger}[\mathcal{C}] := \mathsf{ledger}[\mathcal{C}] - n \cdot \ddot{\mathbb{B}}_{\mathcal{C}}$
  - start a timer $\mathcal{T}_{\mathsf{deliver}}$ and let $\Sigma := \mathsf{initiated}$
  - send $(\mathsf{initiated}, pk_{\mathcal{C}}, vpk_{\mathcal{C}})$ to all entities
- On receive $(\mathsf{delivered})$ from $\mathcal{C}$ or $\mathcal{T}_{\mathsf{deliver}}$ times out:
  - assert $\Sigma \equiv \mathsf{initiated}$
  - send $(\mathsf{getVFDProof})$ to $\mathcal{D}$, and wait for two rounds to receive $(\mathsf{receipt}, i, \sigma_{\mathcal{C}}^i)$, then execute $\mathsf{verifyVFDProof}()$ to let $\mathsf{ctr} := i$, and then assert $0 \leq \mathsf{ctr} \leq n$
  - let $\mathsf{ledger}[\mathcal{D}] := \mathsf{ledger}[\mathcal{D}] + \mathsf{ctr} \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$
  - let $\mathsf{ledger}[\mathcal{P}] := \mathsf{ledger}[\mathcal{P}] + (n - \mathsf{ctr}) \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$
  - store ctr, let $\Sigma := \mathsf{revealing}$, and send $(\mathsf{revealing}, \mathsf{ctr})$ to all entities

Figure 4.2 The downloading-setting arbiter functionality $\mathcal{G}_d^{\mathsf{ledger}}$.

The Arbiter Contract Functionality $\mathcal{G}_d^{\mathsf{ledger}}$ for P2P Downloading (Cont.)

Phase 3: Reveal

- On receive $(\mathsf{revealKeys}, erk)$ from $\mathcal{P}$:
  - assert $\Sigma \equiv \mathsf{revealing}$
  - store erk (essentially erk's hash) and start a timer $\mathcal{T}_{\mathsf{dispute}}$
  - let $\Sigma := \mathsf{revealed}$
  - send $(\mathsf{revealed}, erk)$ to all entities
- Upon $\mathcal{T}_{\mathsf{dispute}}$ times out:
  - assert $\Sigma \equiv \mathsf{revealed}$ and current time $\mathcal{T} \geq \mathcal{T}_{\mathsf{dispute}}$
  - $\mathsf{ledger}[\mathcal{P}] := \mathsf{ledger}[\mathcal{P}] + \mathsf{ctr} \cdot \ddot{\mathbb{B}}_{\mathcal{C}} + \ddot{\mathbb{B}}_{\mathsf{pf}}$
  - $\mathsf{ledger}[\mathcal{C}] := \mathsf{ledger}[\mathcal{C}] + (n - \mathsf{ctr}) \cdot \ddot{\mathbb{B}}_{\mathcal{C}}$
  - let $\Sigma := \mathsf{sold}$ and send (sold) to all entities
- On receive $(\mathsf{wrongRK})$ from $\mathcal{C}$ before $\mathcal{T}_{\mathsf{dispute}}$ times out:
  - assert $\Sigma \equiv \mathsf{revealed}$ and current time $\mathcal{T} < \mathcal{T}_{\mathsf{dispute}}$
  - if $(\mathsf{ValidateRKeys}(n, \mathsf{ctr}, erk) \equiv false)$:
    - let $\mathsf{ledger}[\mathcal{C}] := \mathsf{ledger}[\mathcal{C}] + n \cdot \ddot{\mathbb{B}}_{\mathcal{C}} + \ddot{\mathbb{B}}_{\mathsf{pf}}$
    - let $\Sigma := \mathsf{not\_sold}$ and send $(\mathsf{not\_sold})$ to all entities
- On receive $(\mathsf{PoM}, i, j, c_i, \sigma_{c_i}, \mathcal{H}(m_i), \pi^i_{\mathsf{MT}}, rk, erk, \pi_{\mathsf{VD}})$ from $\mathcal{C}$ before $\mathcal{T}_{\mathsf{dispute}}$ times out:
  - assert $\Sigma \equiv \mathsf{revealed}$ and current time $\mathcal{T} < \mathcal{T}_{\mathsf{dispute}}$
  - invoke the $\mathsf{ValidatePoM}(i, j, c_i, \sigma_{c_i}, \mathcal{H}(m_i), \pi^i_{\mathsf{MT}}, rk, erk, \pi_{\mathsf{VD}})$ subroutine, if $true$ is returned:
    - let $\mathsf{ledger}[\mathcal{C}] := \mathsf{ledger}[\mathcal{C}] + n \cdot \ddot{\mathbb{B}}_{\mathcal{C}} + \ddot{\mathbb{B}}_{\mathsf{pf}}$
    - let $\Sigma := \mathsf{not\_sold}$ and send $(\mathsf{not\_sold})$ to all entities

▷ Reset to the ready state for repeatable delivery

- On receive (reset) from $\mathcal{P}$:
  - assert $\Sigma \equiv \mathsf{sold}$ or $\Sigma \equiv \mathsf{not\_sold}$
  - set ctr, $\mathcal{T}_{\mathsf{deliver}}$, $\mathcal{T}_{\mathsf{dispute}}$ as 0
  - nullify $pk_{\mathcal{C}}$ and $vpk_{\mathcal{C}}$
  - let $\theta := \theta - 1$, and $\Sigma := \mathsf{ready}$
  - send (ready) to all entities

Figure 4.3 The continuation of downloading-setting arbiter functionality $\mathcal{G}_d^{\mathsf{ledger}}$.

4.6.3 Protocol Details

Now we present the details of fair P2P downloading protocol Π_{FD}. In particular, the protocol aims to deliver a content m made of n chunks[5] with a-priori known digest in

[5]W.l.o.g., we assume $n = 2^k$ for $k \in \mathbb{Z}$ for presentation simplicity.

the form of Merkle tree root, i.e., root_m. We omit the session id *sid* and the content digest root_m during the protocol description since they remain the same within a delivery session.

Phase I for Prepare. The provider $\mathcal{P}$ and the deliverer $\mathcal{D}$ interact with the contract functionality $\mathcal{G}_d$ in this phase as:

- The provider $\mathcal{P}$ deploys contracts and starts [6] Π_{FD} by taking as input the security parameter λ, the incentive parameters $\ddot{\mathbb{B}}_{\mathcal{P}}, \ddot{\mathbb{B}}_{\mathcal{C}}, \ddot{\mathbb{B}}_{\mathsf{pf}} \in \mathbb{N}$, where $\ddot{\mathbb{B}}_{\mathsf{pf}}$ is the *penalty fee*[7] in a delivery session to discourage the misbehavior from the provider $\mathcal{P}$, the number of times θ of repeatable delivery allowed for the contract, the n-chunk content $m = (m_1, \ldots, m_n) \in \{0,1\}^{\eta \times n}$ satisfying $\mathsf{root}(\mathsf{BuildMT}(m)) \equiv \mathsf{root}_m$ where root_m is the content digest in the form of Merkle tree root, and executes $(pk_{\mathcal{P}}, sk_{\mathcal{P}}) \leftarrow \mathsf{SIG.KGen}(1^{\lambda})$, and sends $(\mathsf{start}, pk_{\mathcal{P}}, \mathsf{root}_m, \theta, n, \ddot{\mathbb{B}}_{\mathcal{P}}, \ddot{\mathbb{B}}_{\mathcal{C}}, \ddot{\mathbb{B}}_{\mathsf{pf}})$ to $\mathcal{G}_d$.
- Upon $\Sigma \equiv \mathsf{joined}$, the provider $\mathcal{P}$ would execute:
  - Randomly samples a master key $mk \leftarrow_{\$} \{0,1\}^{\lambda}$, and runs Algorithm 13, namely $\mathsf{KT} \leftarrow \mathsf{GenSubKeys}(n, mk)$; stores mk and KT locally;
  - Uses the leaf nodes, namely $\mathsf{KT}[n-1 : 2n-2]$ (i.e., exemplified by Figure 4.4a) as the encryption keys to encrypt $(m_1, \ldots, m_n)$, namely $c = (c_1, \ldots, c_n) \leftarrow (\mathsf{SEnc}_{\mathsf{KT}[n-1]}(m_1), \ldots, \mathsf{SEnc}_{\mathsf{KT}[2n-2]}(m_n))$;
  - Signs the encrypted chunks to obtain the sequence $((c_1, \sigma_{c_1}), \cdots, (c_n, \sigma_{c_n}))$ where the signature $\sigma_{c_i} \leftarrow \mathsf{Sign}(i||c_i, sk_{\mathcal{P}}), i \in [n]$; meanwhile, computes $\mathsf{MT} \leftarrow \mathsf{BuildMT}(m)$ and signs the Merkle tree MT to obtain $\sigma_{\mathcal{P}}^{\mathsf{MT}} \leftarrow \mathsf{Sign}(\mathsf{MT}, sk_{\mathcal{P}})$, then locally stores $(\mathsf{MT}, \sigma_{\mathcal{P}}^{\mathsf{MT}})$ and sends $(\mathsf{sell}, ((c_1, \sigma_{c_1}), \cdots, (c_n, \sigma_{c_n})))$ to $\mathcal{D}$;
  - Waits for (ready) from $\mathcal{G}_d$ to enter the next phase.
- The deliverer $\mathcal{D}$ executes as follows during this phase:
  - Upon receiving $(\mathsf{started}, pk_{\mathcal{P}}, \mathsf{root}_m, \theta, n, \ddot{\mathbb{B}}_{\mathcal{P}}, \ddot{\mathbb{B}}_{\mathcal{C}}, \ddot{\mathbb{B}}_{\mathsf{pf}})$ from $\mathcal{G}_d$, executes $(pk_{\mathcal{D}}, sk_{\mathcal{D}}) \leftarrow \mathsf{SIG.KGen}(1^{\lambda})$, and sends $(\mathsf{join}, pk_{\mathcal{D}})$ to $\mathcal{G}_d$;
  - Waits for $(\mathsf{sell}, ((c_1, \sigma_{c_1}), \cdots, (c_n, \sigma_{c_n})))$ from $\mathcal{P}$ and then: for every (c_i, σ_{c_i}) in the sell message, asserts that $\mathsf{Verify}(i||c_i, \sigma_{c_i}, pk_{\mathcal{P}}) \equiv 1$; if hold, sends $(\mathsf{prepared})$ to $\mathcal{G}_d$, and stores $((c_1, \sigma_{c_1}), \cdots, (c_n, \sigma_{c_n}))$ locally;
  - Waits for (ready) from $\mathcal{G}_d$ to enter the next phase.

Till now, $\mathcal{P}$ owns a master key mk, the key tree KT, and the Merkle tree MT while $\mathcal{D}$ receives the encrypted content chunks and is ready to deliver.

[6] $\mathcal{P}$ can retrieve the deposits of $\ddot{\mathbb{B}}_{\mathcal{P}}$ and $\ddot{\mathbb{B}}_{\mathsf{pf}}$ back if there is no deliverer responds timely.
[7] $\ddot{\mathbb{B}}_{\mathsf{pf}}$ can be set proportional to $(n \times \ddot{\mathbb{B}}_{\mathcal{C}})$ in case $\mathcal{P}$ deliberately lowers it.

Algorithm 13 GenSubKeys Algorithm

Input: n, mk

Output: a $(2n-1)$-array KT

let KT be an array, $|\mathsf{KT}| = 2n-1$;
$\mathsf{KT}[0] = \mathcal{H}(mk)$;
if $n \equiv 1$ **then**
    return KT;
if $n > 1$ **then**
    for i in $[0, n-2]$ **do**
        $\mathsf{KT}[2i+1] = \mathcal{H}(\mathsf{KT}[i]||0)$;
        $\mathsf{KT}[2i+2] = \mathcal{H}(\mathsf{KT}[i]||1)$;
return KT;

Phase II for Deliver. The consumer $\mathcal{C}$, the provider $\mathcal{P}$, and the deliverer $\mathcal{D}$ interact with the contract $\mathcal{G}_d$ in this phase as:

- The consumer $\mathcal{C}$ would execute as follows:
  - Asserts $\Sigma \equiv \mathsf{ready}$, runs $(pk_{\mathcal{C}}, sk_{\mathcal{C}}) \leftarrow \mathsf{SIG.KGen}(1^\lambda)$ and $(vpk_{\mathcal{C}}, vsk_{\mathcal{C}}) \leftarrow \mathsf{VPKE.KGen}(1^\lambda)$, and sends $(\mathsf{consume}, pk_{\mathcal{C}}, vpk_{\mathcal{C}})$ to $\mathcal{G}_d$;
  - Upon receiving $(\mathsf{mtree}, \mathsf{MT}, \sigma_{\mathcal{P}}^{\mathsf{MT}})$ from $\mathcal{P}$ where $\mathsf{Verify}(\mathsf{MT}, \sigma_{\mathcal{P}}^{\mathsf{MT}}, pk_{\mathcal{P}}) \equiv 1$ and $\mathsf{root}(\mathsf{MT}) \equiv \mathsf{root}_m$, stores the Merkle tree MT and then activates the receiver $\mathcal{R}$ in the VFD subroutine by invoking $\mathcal{R}.\mathsf{recv}()$ and instantiating the external validation function $\Psi(i, c_i, \sigma_{c_i})$ as $\mathsf{Verify}(i||c_i, \sigma_{c_i}, pk_{\mathcal{P}})$, and then waits for the execution of VFD to return the delivered chunks $((c_1, \sigma_{c_1}), (c_2, \sigma_{c_2}), \cdots)$ and stores them; upon receiving the whole n-size sequence after executing the VFD module, sends $(\mathsf{delivered})$ to $\mathcal{G}_d$;
  - Waits for $(\mathsf{revealing}, \mathsf{ctr})$ from $\mathcal{G}_d$ to enter the next phase.
- The provider $\mathcal{P}$ executes in this phase as: upon receiving $(\mathsf{initiated}, pk_{\mathcal{C}}, vpk_{\mathcal{C}})$ from $\mathcal{G}_d$, asserts $\Sigma \equiv \mathsf{initiated}$, and sends $(\mathsf{mtree}, \mathsf{MT}, \sigma_{\mathcal{P}}^{\mathsf{MT}})$ to $\mathcal{C}$, and then enters the next phase.
- The deliverer $\mathcal{D}$ executes as follows during this phase:
  - Upon receiving $(\mathsf{initiated}, pk_{\mathcal{C}}, vpk_{\mathcal{C}})$ from $\mathcal{G}_d$: asserts $\Sigma \equiv \mathsf{initiated}$, and then activates the sender $\mathcal{S}$ in the VFD module by invoking $\mathcal{S}.\mathsf{send}()$ and instantiating the validation function $\Psi(i, c_i, \sigma_{c_i})$ as $\mathsf{Verify}(i||c_i, \sigma_{c_i}, pk_{\mathcal{P}})$, and feeds VFD module with input $((c_1, \sigma_{c_1}), \ldots, (c_n, \sigma_{c_n}))$;
  - Upon receiving $(\mathsf{getVFDProof})$ from $\mathcal{G}_d$, sends the latest receipt, namely $(\mathsf{receipt}, i, \sigma_{\mathcal{C}}^i)$ to $\mathcal{G}_d$;

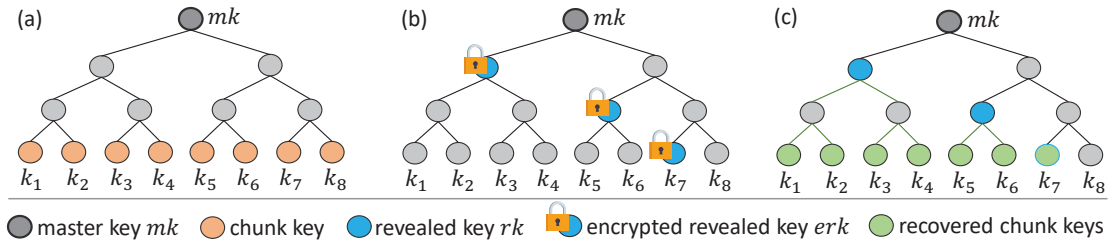


Figure 4.4 An example of the structured key derivation scheme in Π_{FD}.

– Waits for $(\mathsf{revealing}, \mathsf{ctr})$ from $\mathcal{G}_d$ to halt.

At the end of this phase, $\mathcal{C}$ receives the sequence of encrypted chunks $(c_1, c_2, \dots)$, and $\mathcal{D}$ receives the payment for the bandwidth contribution of delivering chunks, and the contract records the number of delivered chunks ctr.

Phase III for Reveal. This phase is completed by $\mathcal{P}$ and $\mathcal{C}$ in the assistance of the arbiter contract $\mathcal{G}_d$, which proceeds as:

- The provider $\mathcal{P}$ proceeds as follows during this phase:
  - Asserts $\Sigma \equiv \mathsf{revealing}$, runs Algorithm 14, i.e., $rk \leftarrow \mathsf{RevealKeys}(n, \mathsf{ctr}, mk)$ to generate the revealed elements rk, and encrypt rk by running $erk \leftarrow \mathsf{VEnc}_{vpk_C}(rk)$, as exemplified by Figure 4.4b, and sends $(\mathsf{revealKeys}, erk)$ to $\mathcal{G}_d$; waits for (sold) from $\mathcal{G}_d$ to halt.
- The consumer $\mathcal{C}$ in this phase would first assert $\Sigma \equiv \mathsf{revealing}$ and wait for $(\mathsf{revealed}, erk)$ from $\mathcal{G}_d$ to execute the following:
  - Runs Algorithm 15, namely $\mathsf{ValidateRKeys}(n, \mathsf{ctr}, erk)$ to preliminarily check whether the revealed elements erk can recover the correct number (i.e, ctr) of keys. If $false$ is returned, sends $(\mathsf{wrongRK})$ to $\mathcal{G}_d$ and halts;
  - If $\mathsf{ValidateRKeys}(n, \mathsf{ctr}, erk) \equiv true$, decrypts erk to obtain $rk \leftarrow \mathsf{VDec}_{vsk_C}(erk)$, and then runs Algorithm 16, i.e., $ks = (k_1, \cdots, k_{\mathsf{ctr}}) \leftarrow \mathsf{RecoverKeys}(n, \mathsf{ctr}, rk)$, as exemplified by Figure 4.4c, to recover the chunk keys. Then $\mathcal{C}$ uses these keys to decrypt $(c_1, \cdots, c_{\mathsf{ctr}})$ to get $(m'_1, \cdots, m'_{\mathsf{ctr}})$, where $m'_i = \mathsf{SDec}_{k_i}(c_i), i \in [\mathsf{ctr}]$, and checks whether for every $m'_i \in (m'_1, \cdots, m'_{\mathsf{ctr}})$, $\mathcal{H}(m'_i)$ is the i-th leaf node in Merkle tree MT received from $\mathcal{P}$ in the *Deliver* phase. If all are consistent, meaning that $\mathcal{C}$ receives all the desired chunks and there is no dispute, $\mathcal{C}$ outputs $(m'_1, \cdots, m'_{\mathsf{ctr}})$, and then waits for (sold) from $\mathcal{G}_d$ to halt. Otherwise, $\mathcal{C}$ can raise complaint by: choosing one inconsistent position (e.g., the i-th chunk), and computes $(rk, \pi_{\mathsf{VD}}) \leftarrow \mathsf{ProvePKE}_{vsk_C}(erk)$ and $\pi^i_{\mathsf{M}} \leftarrow \mathsf{GenMTP}(\mathsf{MT}, \mathcal{H}(m_i))$, and then sends $(\mathsf{PoM}, i, j, c_i, \sigma_{c_i}, \mathcal{H}(m_i), \pi^i_{\mathsf{MT}}, rk, erk, \pi_{\mathsf{VD}})$ to the contract $\mathcal{G}_d$, where

Algorithm 14 RevealKeys Algorithm

Input: n, ctr, and mk

Output: rk, an array containing the minimum number of elements in KT that suffices to recover the ctr keys from $\mathsf{KT}[n-1]$ to $\mathsf{KT}[n+\mathsf{ctr}-2]$

```
let rk and ind be empty arrays;
KT ← GenSubKeys(n, mk);
if ctr ≡ 1 then
    rk appends (n − 1, KT[n − 1]);
    return rk;
for i in [0, ctr − 1] do
    ind[i] = n − 1 + i;
while true do
    let t be an empty array;
    for j in [0, ⌊|ind|/2⌋ − 1] do
        p_l = (ind[2j] − 1)/2;
        p_r = (ind[2j + 1] − 2)/2;  ▷ merge elements with the same parent node in KT
        if p_l ≡ p_r then
            t appends p_l;
        else
            t appends ind[2j];
            t appends ind[2j + 1];
    if |ind| is odd then
        t appends ind[|ind| − 1];
    if |ind| ≡ |t| then
        break;
    ind = t;
for x in [0, |ind| − 1] do
    rk appends (ind[x], KT[ind[x]]);
return rk;
```

i is the index of the incorrect chunk to be proved; j is the index of the element in erk that can induce the key k_i for the position i; c_i and σ_{c_i} are the i-th encrypted chunk and its signature received in the *Deliver* phase; $\mathcal{H}(m_i)$ is the value of the i-th leaf node in MT; π^i_{MT} is the Merkle proof for $\mathcal{H}(m_i)$; rk is decryption result from erk; erk is the encrypted revealed

Algorithm 15 ValidateRKeys Algorithm

Input: n, ctr and erk

Output: $true$ or $false$ indicating that whether the correct number (i.e., ctr) of decryption keys can be recovered

if $n \equiv \mathsf{ctr}$ and $|erk| \equiv 1$ and the position of $erk[0] \equiv 0$ **then**
  return $true$; ▷ root of KT
Initialize $chunks\_index$ as a set of numbers $\{n-1, \ldots, n+\mathsf{ctr}-2\}$;
for each $(i, \_)$ in erk **do**
  $d_i = \log(n) - \lfloor \log(i+1) \rfloor$;
  $l_i = i$, $r_i = i$;
  if $d_i \equiv 0$ **then**
    $chunks\_index$ removes i;
  else
    while $(d_i\text{--}) > 0$ **do**
      $l_i = 2l_i + 1$;
      $r_i = 2r_i + 2$;
  $chunks\_index$ removes the elements from l_i to r_i;
if $chunks\_index \equiv \emptyset$ **then**
  return $true$;
return $false$;

Algorithm 16 RecoverKeys Algorithm

Input: n, ctr, and rk

Output: a ctr-sized array ks

let ks be an empty array;
for each $(i, \mathsf{KT}[i])$ in rk **do**
  $n_i = 2^{(\log n - \lfloor \log(i+1) \rfloor)}$;
  $v_i = \mathsf{GenSubKeys}(n_i, \mathsf{KT}[i])$;
  ks appends $v_i[n_i - 1 : 2n_i - 2]$;
return ks;

key; π_{VD} is the verifiable decryption proof attesting to the correctness of decrypting erk to rk.

Dispute resolution. For the sake of completeness, the details of ValidatePoM subroutine is presented in Algorithm 17, which allows the consumer to prove that

Algorithm 17 $\mathsf{ValidatePoM}$ Algorithm

Input: $(i, j, c_i, \sigma_{c_i}, \mathcal{H}(m_i), \pi^i_{\mathsf{MT}}, rk, erk, \pi_{\mathsf{VD}})$

$(\mathsf{root}_m, n, erk_{\mathsf{hash}}, pk_{\mathcal{P}}, vpk_{\mathcal{C}})$ are stored in the contract and hence accessible

Output: $true$ or $false$

1: assert $j \in [0, |erk| - 1]$;
2: assert $\mathcal{H}(erk) \equiv erk_{\mathsf{hash}}$;
3: assert $\mathsf{VerifyPKE}_{vpk_{\mathcal{C}}}(erk, rk, \pi_{\mathsf{VD}}) \equiv 1$;
4: assert $\mathsf{Verify}(i||c_i, \sigma_{c_i}, pk_{\mathcal{P}}) \equiv 1$;
5: assert $\mathsf{VerifyMTP}(\mathsf{root}_m, i, \pi^i_{\mathsf{MT}}, \mathcal{H}(m_i)) \equiv 1$;
6: $k_i = \mathsf{RecoverChunkKey}(i, j, n, rk)$;
7: assert $k_i \neq \perp$;
8: $m'_i = \mathsf{SDec}(c_i, k_i)$;
9: assert $\mathcal{H}(m'_i) \neq \mathcal{H}(m_i)$;
10: **return** $false$ in case of any assertion error or $true$ otherwise;

Algorithm 18 $\mathsf{RecoverChunkKey}$ Algorithm

Input: (i, j, n, rk)

Output: k_i or $\perp$

1: $(x, y) \leftarrow rk[j]$; ▷ parse the j-th element in rk to get the key x and the value y
2: let k_path be an empty stack;
3: $ind = n + i - 2$; ▷ index in KT
4: **if** $ind < x$ **then**
5: **return** $\perp$;
6: **if** $ind \equiv x$ **then**
7: **return** y; ▷ $k_i = y$
8: **while** $ind > x$ **do**
9: k_path pushes 0 if ind is odd;
10: k_path pushes 1 if ind is even;
11: $ind = \lfloor (ind - 1)/2 \rfloor$;
12: let $b = |k\_path|$;
13: **while** $(b\text{--}) > 0$ **do**
14: pop k_path to get the value t;
15: $k_i = \mathcal{H}(y||t)$;
16: **return** k_i;

it decrypts a chunk inconsistent to the digest root_m. The time complexity is $O(\log n)$, which is critical to achieve the efficiency requirement. Additionally, we consider a natural case where an honest consumer $\mathcal{C}$ would not complain to the contract if receiving valid content.

Design highlights. We would like to highlight some design details in Π_{FD}: (i) the rk is an array containing several revealed elements, which are in the form of $(position, value)$. The erk shares the similar structure where the $position$ is same and $value$ is encrypted from the corresponding $rk.value$. The $position$ is the index in KT; (ii) to reduce the on-chain cost, the contract only stores the 256-bit hash of the $erk.value$ while emits the actual erk as event logs. During the dispute resolution, $\mathcal{C}$ submits the j-th erk element, and the contract would check the consistency of the submitted erk with its on-chain hash; (iii) Algorithm 15 allows the judge contract to perform preliminary check on whether the revealed elements can recover the desired number (i.e., ctr) of decryption keys, without directly executing the relatively complex contract part of $\mathsf{ValidatePoM}$ (i.e., Algorithm 17).

Repeatable delivery. The protocol Π_{FD} supports repeatable delivery, meaning that once a delivery session completes, the provider $\mathcal{P}$ can invoke the contract (by sending (reset) to $\mathcal{G}_d$) to reset to ready state, so that new consumers can join in and start a new protocol instance. Such a θ-time repeatable delivery mechanism can amortize the costs of contracts deployment and preparation (i.e., delegating encrypted chunks to a deliverer). Once θ decreases to 0, the provider $\mathcal{P}$ can either deploy a new contract (thus residing at a new contract address) or utilize the same contract address while re-run the *Prepare* phase. $\mathcal{P}$ may not need to delegate the encrypted chunks if a previously participating deliverer joins in.

4.6.4 Protocol Analysis

Now we provide the detailed proofs that the protocol Π_{FD} satisfies the design goals.

Lemma 2. *Conditioned that all parties $\mathcal{P}$, $\mathcal{D}$ and $\mathcal{C}$ are honest, Π_{FD} satisfies the completeness property in the synchronous authenticated network and stand-alone model.*

Proof. The completeness of Π_{FD} is immediate to see: when all three participating parties honestly follow the protocol, the provider $\mathcal{P}$ gets a net income of $n \cdot (\ddot{\mathbb{B}}_{\mathcal{C}} - \ddot{\mathbb{B}}_{\mathcal{P}})$; the deliverer $\mathcal{D}$ obtains the well-deserved payment of $n \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$; the consumer $\mathcal{C}$ receives the valid content m, i.e., $\phi(m) \equiv 1$.

Lemma 3. *In the synchronous authenticated model and stand-alone setting, conditioned that the underlying cryptographic primitives are secure, Π_{FD} satisfies the fairness requirement even when at most two parties of $\mathcal{P}$, $\mathcal{D}$ and $\mathcal{C}$ are corrupted by non-adaptive P.P.T. adversary $\mathcal{A}$.*

Proof. The fairness for each party in Π_{FD} can be reduced to the underlying cryptographic building blocks, which can be analyzed as follows:

- *Consumer fairness*. Consumer fairness means that the honest $\mathcal{C}$ only needs to pay proportional to what it *de facto* obtains even though malicious $\mathcal{P}^*$ and $\mathcal{D}^*$ may collude with each other. This case can be modeled as an adversary $\mathcal{A}$ corrupts both $\mathcal{P}$ and $\mathcal{D}$ to provide and deliver the content to the honest $\mathcal{C}$. During the *Deliver* phase, the VFD subroutine ensures that $\mathcal{C}$ receives the sequence $(c_1, \sigma_{c_1}), \ldots, (c_\ell, \sigma_{c_\ell})$, $\ell \in [n]$ though $\mathcal{A}$ may maliciously abort. Later $\mathcal{A}$ can only claim payment from the contract of $\ell \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$, which is paid by the $\mathcal{A}$ itself due to the collusion. During the *Reveal* phase, if $\mathcal{A}$ reveals correct elements in KT to recover the ℓ decryption keys, then $\mathcal{C}$ can decrypt to obtain the valid ℓ chunks. Otherwise, $\mathcal{C}$ can raise complaint by sending the ($\mathsf{wrongRK}$) and further (PoM) to the contract and gets refund. It is obvious to see that $\mathcal{C}$ either pays for the ℓ valid chunks or pays nothing. The fairness for the consumer is guaranteed unless $\mathcal{A}$ can: (i) break VFD to forge $\mathcal{C}$'s signature; (ii) find Merkle tree collision, namely find another chunk $m_i' \neq m_i$ in position i of m to bind to the same root_m so that $\mathcal{A}$ can fool the contract to reject $\mathcal{C}$'s complaint (by returning $false$ of $\mathsf{ValidatePoM}$) while indeed sent wrong chunks; (iii) manipulate the execution of smart contract in blockchain. However, according to the security guarantee of the underlying signature scheme, the second-preimage resistance of hash function in Merkle tree, and that the smart contract is modeled as an ideal functionality, the probability to break $\mathcal{C}$'s fairness is negligible. Therefore, the consumer fairness being secure against the collusion of malicious $\mathcal{P}^*$ and $\mathcal{D}^*$ is guaranteed.

- *Deliverer fairness*. Deliverer fairness states that the honest $\mathcal{D}$ receives the payment proportional to the expended bandwidth even though the malicious $\mathcal{P}^*$ and $\mathcal{C}^*$ may collude with each other. This amounts to the case that $\mathcal{A}$ corrupts both $\mathcal{P}$ and $\mathcal{C}$ and try to reap $\mathcal{D}$'s bandwidth contribution without paying. In the VFD subroutine, considering $\mathcal{D}$ delivers ℓ chunks, then it can correspondingly obtain either ℓ ($\ell \in [n]$) or $\ell - 1$ (i.e., $\mathcal{A}$ stops sending the ℓ-th receipt) receipts acknowledging the bandwidth contribution. Later $\mathcal{D}$ can use the latest receipt containing $\mathcal{C}$'s signature to claim payment $\ell \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$ or $(\ell-1) \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$ from the contract. At most $\mathcal{D}$ may waste bandwidth for delivering one chunk-validation pair of $O(\eta)$ bits. To break the security, $\mathcal{A}$ has to violate the contract functionality (i.e., control the execution of smart contract in blockchain), which is of negligible probability. Therefore, the deliverer fairness being secure against the collusion of malicious $\mathcal{P}^*$ and $\mathcal{C}^*$ is ensured.

- *Provider fairness*. Provider fairness indicates that the honest $\mathcal{P}$ receives the payment proportional to the number of valid content chunks that $\mathcal{C}$ receives. The malicious $\mathcal{D}^*$ can collude with the malicious $\mathcal{C}^*$ or simply create multiple fake $\mathcal{C}^*$ (i.e., Sybil attack), and then cheat $\mathcal{P}$ without real delivery. These cases can be modeled as an adversary $\mathcal{A}$ corrupts both $\mathcal{D}$ and $\mathcal{C}$. $\mathcal{A}$ can break the fairness of the honest $\mathcal{P}$ from two aspects by: (i) letting $\mathcal{P}$ pay for the delivery without truly delivering any content; (ii) obtaining the content without paying for $\mathcal{P}$. For case (i), $\mathcal{A}$ can claim that ℓ ($\ell \in [n]$) chunks have been delivered and would receive the payment $\ell \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$ from the contract. Yet this procedure would also update $\mathsf{ctr} := \ell$ in the contract, which later allows $\mathcal{P}$ to receive the payment $\ell \cdot \ddot{\mathbb{B}}_{\mathcal{C}}$ after $\mathcal{T}_{\mathsf{dispute}}$ expires unless $\mathcal{A}$ can manipulate the execution of smart contract, which is of negligible probability. Hence, $\mathcal{P}$ can still obtain the well-deserved payment $\ell \cdot (\ddot{\mathbb{B}}_{\mathcal{C}} - \ddot{\mathbb{B}}_{\mathcal{P}})$. For case (ii), $\mathcal{A}$ can either try to decrypt the delivered chunks by itself without utilizing the revealing keys from $\mathcal{P}$, or try to fool the contract to accept the PoM and therefore repudiate the payment for $\mathcal{P}$ though $\mathcal{P}$ honestly reveals chunk keys. The former situation can be reduced to the need of violating the semantic security of the underlying encryption scheme and the pre-image resistance of cryptographic hash functions, and the latter requires $\mathcal{A}$ to forge $\mathcal{P}$'s signature, or break the soundness of the verifiable decryption scheme, or control the execution of the smart contract. Obviously, the occurrence of aforementioned situations are in negligible probability. Overall, the provider fairness being secure against the collusion of malicious $\mathcal{D}^*$ and $\mathcal{C}^*$ is assured.

In sum, Π_{FD} strictly guarantees the fairness for $\mathcal{P}$ and $\mathcal{C}$, and the unpaid delivery for $\mathcal{D}$ is bounded to $O(\eta)$ bits. The fairness requirement of Π_{FD} is satisfied.

Lemma 4. *In the synchronous authenticated network and stand-alone model,* Π_{FD} *satisfies the confidentiality property against malicious deliverer corrupted by non-adaptive P.P.T. adversary* $\mathcal{A}$*.*

Proof. This property states that on input all protocol scripts and the corrupted deliverer's private input and all internal states, it is still computationally infeasible for the adversary to output the provider's input content. In Π_{FD}, each chunk delegated to $\mathcal{D}$ is encrypted using symmetric encryption scheme before delivery by encryption key derived from Algorithm 13. The distribution of encryption keys and uniform distribution cannot be distinguished by the P.P.T. adversary. Furthermore, the revealed on-chain elements erk for recovering some chunks' encryption keys are also encrypted utilizing the consumer $\mathcal{C}$'s pubic key, which can not be distinguished from uniform distribution by the adversary. Additionally, $\mathcal{C}$ receives the Merkle tree MT of the content m before the verifiable fair delivery (VFD) procedure starts. Thus to break the confidentiality property, the adversary $\mathcal{A}$ has to violate any of the following conditions: (i) the pre-image resistance of Merkle tree, which can be further reduce to the pre-image resistance of cryptographic hash function; and (ii) the security of the public key encryption scheme, essentially requiring at least to solve decisional Diffie-Hellman problem. The probability of violating the aforementioned security properties is negligible, and therefore, Π_{FD} satisfies the confidentiality property against malicious deliverer corrupted by $\mathcal{A}$.

Lemma 5. *If at least one of the three parties* $\mathcal{P}$*,* $\mathcal{D}$*,* $\mathcal{C}$ *is honest and others are corrupted by non-adaptive P.P.T. adversary* $\mathcal{A}$*,* Π_{FD} *satisfies the timeliness property in the synchronous authenticated network and stand-alone model.*

Proof. Timeliness states that the honest parties in the protocol Π_{FD} terminates in $O(n)$ synchronous rounds, where n is the number of content chunks, and when the protocol completes or aborts, the fairness and confidentiality are always preserved. As

the guarantee of confidentiality can be straightforwardly derived due to the lemma 4 even if malicious parties abort, we only focus on the assurance of fairness. Now we elaborate the following termination cases for the protocol Π_{FD} with the arbiter contract $\mathcal{G}_d$ and at least one honest party:

No abort. If all parties of $\mathcal{P}$, $\mathcal{D}$ and $\mathcal{C}$ are honest, the protocol Π_{FD} terminates in the *Reveal* phase, after $\mathcal{T}_{\mathsf{dispute}}$ expires. The *Prepare* phase and the *Reveal* phase need $O(1)$ synchronous rounds, and the *Deliver* phase requires $O(n)$ rounds where n is the number of content chunks, yielding totally $O(n)$ rounds for the protocol Π_{FD} to terminate and the fairness is guaranteed at completion since each party obtains the well-deserved items.

Aborts in the Prepare phase. This phase involves the interaction between the provider $\mathcal{P}$, the deliverer $\mathcal{D}$, and the arbiter contract $\mathcal{G}_d$. It is obvious this phase can terminate in $O(1)$ rounds if any party maliciously aborts or the honest party does not receive response after $\mathcal{T}_{\mathsf{round}}$ expires. Besides, after each step in this phase, the fairness for both $\mathcal{P}$ and $\mathcal{D}$ is preserved no matter which one of them aborts, meaning that $\mathcal{P}$ does not lose any coins in the ledger or leak any content chunks, while $\mathcal{D}$ does not waste any bandwidth resource.

Aborts in the Deliver phase. This phase involves the provider $\mathcal{P}$, the deliverer $\mathcal{D}$, the consumer $\mathcal{C}$, and the arbiter contract $\mathcal{G}_d$. It can terminate in $O(n)$ rounds. After $\mathcal{C}$ sends (consume) message to the contract, and then other parties aborts, $\mathcal{C}$ would get its deposit back once $\mathcal{T}_{\mathsf{round}}$ times out. The VFD procedure in this phase only involves $\mathcal{D}$ and $\mathcal{C}$, and the fairness is guaranteed whenever one of the two parties aborts, as analyzed in lemma 1. The timer $\mathcal{T}_{\mathsf{deliver}}$ in contract indicates that the whole n-chunk delivery should be completed within such a time period, or else $\mathcal{G}_d$ would continue with the protocol by informing $\mathcal{D}$ to claim payment and update ctr after $\mathcal{T}_{\mathsf{deliver}}$ times out. $\mathcal{D}$ is motivated not to maliciously abort until receiving the payment from the contract. At the end of this phase, $\mathcal{D}$ completes its task in the delivery session, while

for $\mathcal{P}$ and $\mathcal{C}$, they are motivated to enter the next phase and the fairness for them at this point is guaranteed, i.e., $\mathcal{P}$ decreases coins by $\mathsf{ctr} \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$ in ledger, but the contract has also updated ctr, which allows $\mathcal{P}$ to receive $\mathsf{ctr} \cdot \ddot{\mathbb{B}}_{\mathcal{C}}$ from the ledger if keys are revealed honestly, and $\mathcal{C}$ obtains the encrypted chunks while does not lose any coins in ledger.

Aborts in the Reveal phase. This phase involves the provider $\mathcal{P}$, the consumer $\mathcal{C}$, and the arbiter contract $\mathcal{G}_d$. It can terminate in $O(1)$ rounds after the contract sets the state as sold or $\mathsf{not\_sold}$. If $\mathcal{C}$ aborts after $\mathcal{P}$ reveals the chunk keys on-chain, $\mathcal{P}$ can wait until $\mathcal{T}_{\mathsf{dispute}}$ times out and attain the deserved payment $\mathsf{ctr} \cdot \ddot{\mathbb{B}}_{\mathcal{C}}$. If $\mathcal{P}$ reveals incorrect keys and then aborts, $\mathcal{C}$ can raise complaint within $\mathcal{T}_{\mathsf{dispute}}$ by sending message ($\mathsf{wrongRK}$) and further (PoM) to get refund. Hence, the fairness for either $\mathcal{P}$ and $\mathcal{C}$ is guaranteed no matter when and which one aborts maliciously in this phase.

Lemma 6. *In the synchronous authenticated network and stand-alone model, for any non-adaptive P.P.T. adversary $\mathcal{A}$, Π_{FD} meets the efficiency requirement that: the communication complexity is bounded to $O(n)$; the on-chain cost is bounded to $\widetilde{O}(1)$; the messages sent by the provider $\mathcal{P}$ after preparation are bounded to $n \cdot \lambda$ bits, where n is the number of chunks and λ is a small cryptographic parameter, and $n \cdot \lambda$ is much less than the content size $|m|$.*

Proof. The analysis regarding the non-trivial efficiency property can be conducted in the following three aspects:

- *Communication complexity*. In the *Prepare* phase, $\mathcal{P}$ delegates the signed encrypted chunks to $\mathcal{D}$, where the communication complexity is $O(n)$. Typically this phase is only executed once for the same content. In the *Deliver* phase, $\mathcal{P}$ sends the content Merkle tree MT to $\mathcal{C}$, and $\mathcal{D}$ activates the VFD subroutine to deliver the content chunks to $\mathcal{C}$. The communication complexity in this phase is also $O(n)$. In the *Reveal* phase, the revealed elements for recovering ctr keys is *at most* $O(\log n)$. Finally, if dispute happens, the communication complexity of sending PoM (mostly due to the merkle proof π^i_{MT}) to the contract is $O(\log n)$. Therefore, the communication complexity of the protocol Π_{FD} is $O(n)$.

- *On-chain costs*. In the *optimistic* case where no dispute occurs, the on-chain costs of Π_{FD} include: (i) the functions (i.e., start, join and prepared) in the *Prepare* phase are all $O(1)$; (ii) in the *Deliver* phase, the consume and delivered functions are $O(1)$. Note that in the delivered function, the cost of signature verification is $O(1)$ since $\mathcal{D}$ only needs to submit the latest receipt containing one signature of $\mathcal{C}$; (iii) the storage cost for revealed elements (i.e., erk) is *at most* $O(\log n)$, where n is the number of chunks. Hence, the overall on-chain cost is *at most* $O(\log n)$, namely $\widetilde{O}(1)$. In the *pessimistic* case where dispute happens, the on-chain cost is only related to the delivered chunk size η no matter how large the content size $|m|$ is (the relationship between the chunk size and costs in different modes is depicted in Section 4.8).

- *Message Volume for $\mathcal{P}$*. Considering that the contract is deployed and the deliverer is ready to deliver. Every time when a new consumer joins in, a new delivery session starts. The provider $\mathcal{P}$ shows up twice for: (i) sending the Merkle tree MT, which is in complexity of $O(\log n)$, to $\mathcal{C}$ in the *Deliver* phase, and (ii) revealing erk, which is in complexity of *at most* $O(\log n)$, to $\mathcal{C}$ in the *Reveal* phase. The total resulting message volume $O(\log n)$ can be represented as $n \cdot \lambda$ bits, where λ is a small cryptographic parameter, and $n \cdot \lambda$ is obviously much less than the content size of $|m|$.

Theorem 1. *Conditioned on that the underlying cryptographic primitives are secure, the protocol* FairDownload *satisfies the completeness, fairness, confidentiality against deliverer, timeliness and non-trivial efficiency properties in the synchronous authenticated network, $\mathcal{G}_d^{\mathsf{ledger}}$-hybrid and stand-alone model.*

Proof. Lemmas 2, 3, 4, 5, and 6 complete the proof.

4.7 Fair P2P Streaming Protocol Design

In this section, we present the P2P fair delivery protocol Π_{FS}, allowing *view-while-delivery* in the streaming setting.

4.7.1 Protocol Overview

As depicted in Figure 4.5, our protocol Π_{FS} works as three phases, i.e., *Prepare*, *Stream*, and *Payout*, at a high level. The core ideas for Π_{FS} are:

- Same as the *Prepare* phase in Π_{FD}, initially the content provider $\mathcal{P}$ would deploy the smart contract, encrypt content chunks, sign the encrypted chunks and delegate to the deliverer $\mathcal{D}$.
- The streaming process consists of $O(n)$ communication rounds, where n is the number of chunks. In each round, the consumer $\mathcal{C}$ would receive an encrypted chunk from $\mathcal{D}$ and a decryption key from $\mathcal{P}$; any party may abort in a certain round due to, e.g., untimely response or invalid message; especially, in case erroneous chunk is detected during streaming, $\mathcal{C}$ can complain and get compensated with a valid and short (i.e., $O(\eta + \lambda)$ bits) proof.
- Eventually all parties enter the *Payout* phase, where $\mathcal{D}$ and $\mathcal{P}$ can claim the deserved payment by submitting the latest receipt signed by the consumer before a timer maintained in contract expires; the contract determines the final internal state ctr, namely the number of delivered chunks or revealed keys, as the *larger* one of the indexes in $\mathcal{P}$ and $\mathcal{D}$'s receipts. If no receipt is received from $\mathcal{P}$ or $\mathcal{D}$ before the timer expires, the contract would treat the submitted index for that party as 0. Such a design is critical to ensure fairness.

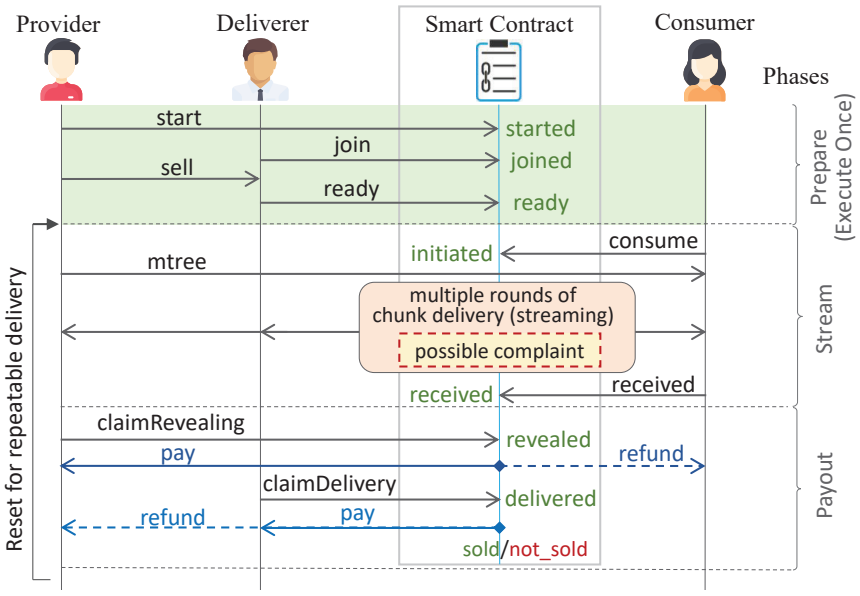


Figure 4.5 The overview of FairStream protocol Π_{FS}.

Figure 4.6 illustrates the concrete message flow of one round chunk delivery during the *Stream* phase. We highlight that a black-box call of the VFD module is not applicable to the streaming setting since VFD only allows the consumer $\mathcal{C}$ to obtain the encrypted chunks, which brings the advantage that the provider $\mathcal{P}$

merely needs to show up once to reveal a minimum number of elements and get all chunk keys to be recovered. However, the streaming procedure demands much less latency of retrieving each content chunk, leading to the intuitive design to let $\mathcal{C}$ receive both an encrypted chunk and a corresponding chunk decryption key in one same round. $\mathcal{P}$ is therefore expected to keep online and reveal each chunk key to $\mathcal{C}$. Overall, the protocol design in Π_{FS} requires relatively more involvement of the provider $\mathcal{P}$ compared with the downloading setting, but the advantage is that instead of downloading all chunks in $O(n)$ rounds before viewing, the consumer $\mathcal{C}$ now can retrieve each chunk with $O(1)$ latency. All other properties including each party's fairness, the on-chain computational cost, and the deliverer's communication complexity remain the same as those in the downloading setting.

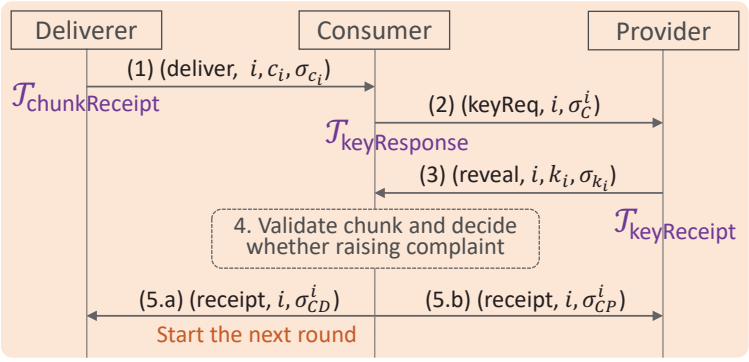


Figure 4.6 The message flow of one round chunk delivery in the *Stream* phase.

4.7.2 Arbiter Contract for Streaming

The arbiter contract $\mathcal{G}_s^{\mathsf{ledger}}$ (abbr. $\mathcal{G}_s$) illustrated in Figure 4.7 is a stateful ideal functionality that can access to ledger functionality to facilitate the fair content delivery via streaming. The timer $\mathcal{T}_{\mathsf{receive}}$ ensures that when any party maliciously aborts or the consumer $\mathcal{C}$ receives invalid chunk during the streaming process, the protocol Π_{FS} can smoothly continue and enter the next phase. The dispute resolution in contract is relatively simpler than the downloading setting since no verifiable decryption is needed. The timer $\mathcal{T}_{\mathsf{finish}}$ indicates that both $\mathcal{D}$ and $\mathcal{P}$ are supposed

to send the request of claiming their payment before $\mathcal{T}_{\mathsf{finish}}$ times out, and therefore it is natural to set $\mathcal{T}_{\mathsf{finish}} > \mathcal{T}_{\mathsf{receive}}$. Once $\mathcal{T}_{\mathsf{finish}}$ expires, the contract determines the final ctr by choosing the maximum index in $\mathcal{P}$ and $\mathcal{D}$'s receipts, namely $\mathsf{ctr}_{\mathcal{P}}$ and $\mathsf{ctr}_{\mathcal{D}}$, respectively, and then distributes the well-deserved payment for each party. Once the delivery session completes, the provider $\mathcal{P}$ can invoke the contract by sending (reset) to $\mathcal{G}_s$ to reset to the ready state and continue to receive new requests from consumers.

4.7.3 Protocol Details

Phase I for Prepare. This phase executes the same as the *Prepare* phase in the Π_{FD} protocol.

Phase II for Stream. The consumer $\mathcal{C}$, the deliverer $\mathcal{D}$ and the provider $\mathcal{P}$ interact with the contract $\mathcal{G}_s$ in this phase as:

- The consumer $\mathcal{C}$ interested in the content with digest root_m would initialize a variable $x := 1$ and then:
  - Asserts $\Sigma \equiv \mathsf{ready}$, runs $(pk_{\mathcal{C}}, sk_{\mathcal{C}}) \leftarrow \mathsf{SIG.KGen}(1^{\lambda})$, and sends $(\mathsf{consume}, pk_{\mathcal{C}})$ to $\mathcal{G}_s$;
  - Upon receiving $(\mathsf{mtree}, \mathsf{MT}, \sigma_{\mathcal{P}}^{\mathsf{MT}})$ from $\mathcal{P}$, asserts $\mathsf{Verify}(\mathsf{MT}, \sigma_{\mathcal{P}}^{\mathsf{MT}}, pk_{\mathcal{P}}) \equiv 1 \wedge \mathsf{root}(\mathsf{MT}) \equiv \mathsf{root}_m$, and stores the Merkle tree MT, or else halts;
  - Upon receiving the message $(\mathsf{deliver}, i, c_i, \sigma_{c_i})$ from $\mathcal{D}$, checks whether $i \equiv x \wedge \mathsf{Verify}(i||c_i, \sigma_{c_i}, pk_{\mathcal{P}}) \equiv 1$, if hold, starts (for $i \equiv 1$) a timer $\mathcal{T}_{\mathsf{keyResponse}}$ or resets (for $1 < i \leq n$) it, sends $(\mathsf{keyReq}, i, \sigma_{\mathcal{C}}^{i})$ where $\sigma_{\mathcal{C}}^{i} \leftarrow \mathsf{Sign}(i||pk_{\mathcal{C}}, sk_{\mathcal{C}})$ to $\mathcal{P}$ (i.e., the step (2) in Figure 4.6). If failing to check or $\mathcal{T}_{\mathsf{keyResponse}}$ times out, halts;
  - Upon receiving the message $(\mathsf{reveal}, i, k_i, \sigma_{k_i})$ from $\mathcal{P}$ before $\mathcal{T}_{\mathsf{keyResponse}}$ times out, checks whether $i \equiv x \wedge \mathsf{Verify}(i||k_i, \sigma_{k_i}, pk_{\mathcal{P}}) \equiv 1$, if failed, halts. Otherwise, starts to validate the content chunk based on received c_i and k_i: decrypts c_i to obtain m_i', where $m_i' = \mathsf{SDec}_{k_i}(c_i)$, and then checks whether $\mathcal{H}(m_i')$ is consistent with the i-th leaf node in the Merkle tree MT, if inconsistent, sends $(\mathsf{PoM}, i, c_i, \sigma_{c_i}, k_i, \sigma_{k_i}, \mathcal{H}(m_i), \pi_{\mathsf{MT}}^{i})$ to $\mathcal{G}_s$. If it is consistent, sends the receipts $(\mathsf{receipt}, i, \sigma_{\mathcal{CD}}^{i})$ to $\mathcal{D}$ and $(\mathsf{receipt}, i, \sigma_{\mathcal{CP}}^{i})$ to $\mathcal{P}$, where $\sigma_{\mathcal{CD}}^{i} \leftarrow \mathsf{Sign}(\mathsf{receipt}||i||pk_{\mathcal{C}}||pk_{\mathcal{D}}, sk_{\mathcal{C}})$ and $\sigma_{\mathcal{CP}}^{i} \leftarrow \mathsf{Sign}(\mathsf{receipt}||i||pk_{\mathcal{C}}||pk_{\mathcal{P}}, sk_{\mathcal{C}})$, and sets $x := x + 1$, and then waits for the next ($\mathsf{deliver}$) message from $\mathcal{D}$. Upon x is set to be $n + 1$, sends ($\mathsf{received}$) to $\mathcal{G}_s$;
  - Waits for the messages ($\mathsf{received}$) from $\mathcal{G}_s$ to halt.

The Arbiter Contract Functionality $\mathcal{G}_s^{\mathsf{ledger}}$ for p2p Streaming

The contract $\mathcal{G}_s$ can access to ledger, and it interacts with $\mathcal{P}$, $\mathcal{D}$, $\mathcal{C}$ and the adversary $\mathcal{A}$. It locally stores θ, n, root_m, $\ddot{\mathbb{B}}_{\mathcal{P}}$, $\ddot{\mathbb{B}}_{\mathcal{C}}$, $\ddot{\mathbb{B}}_{\mathsf{pf}}$, $\mathsf{ctr}_{\mathcal{D}}$, $\mathsf{ctr}_{\mathcal{P}}$, ctr (all $\mathsf{ctr}_{\mathcal{D}}$, $\mathsf{ctr}_{\mathcal{P}}$, ctr are initialized as 0), $pk_{\mathcal{P}}, pk_{\mathcal{D}}, pk_{\mathcal{C}}$, the penalty flag plt (initialized by $false$), the state Σ and three timers $\mathcal{T}_{\mathsf{round}}$ (implicitly), $\mathcal{T}_{\mathsf{receive}}$, $\mathcal{T}_{\mathsf{finish}}$.

Phase 1: Prepare

- This phase is the same as the *Prepare* phase in $\mathcal{G}_d$.

Phase 2: Stream

- On receive $(\mathsf{consume}, pk_{\mathcal{C}})$ from $\mathcal{C}$:
  - assert $\theta > 0$
  - assert $\mathsf{ledger}[\mathcal{C}] \geq n \cdot \ddot{\mathbb{B}}_{\mathcal{C}} \wedge \Sigma \equiv \mathsf{ready}$
  - store $pk_{\mathcal{C}}$ and let $\mathsf{ledger}[\mathcal{C}] := \mathsf{ledger}[\mathcal{C}] - n \cdot \ddot{\mathbb{B}}_{\mathcal{C}}$
  - start two timers $\mathcal{T}_{\mathsf{receive}}$, and $\mathcal{T}_{\mathsf{finish}}$
  - let $\Sigma := \mathsf{initiated}$ and send $(\mathsf{initiated}, pk_{\mathcal{C}})$ to all entities
- On receive $(\mathsf{received})$ from $\mathcal{C}$ before $\mathcal{T}_{\mathsf{receive}}$ times out:
  - assert current time $\mathcal{T} < \mathcal{T}_{\mathsf{receive}}$ and $\Sigma \equiv \mathsf{initiated}$
  - let $\Sigma := \mathsf{received}$ and send $(\mathsf{received})$ to all entities
- Upon $\mathcal{T}_{\mathsf{receive}}$ times out:
  - assert current time $\mathcal{T} \geq \mathcal{T}_{\mathsf{receive}}$ and $\Sigma \equiv \mathsf{initiated}$
  - let $\Sigma := \mathsf{received}$ and send $(\mathsf{received})$ to all entities

▷ Below is to resolve dispute during streaming in Π_{FS}

- On receive $(\mathsf{PoM}, i, c_i, \sigma_{c_i}, k_i, \sigma_{k_i}, \mathcal{H}(m_i), \pi^i_{\mathsf{MT}})$ from $\mathcal{C}$ before $\mathcal{T}_{\mathsf{receive}}$ expires:
  - assert current time $\mathcal{T} < \mathcal{T}_{\mathsf{receive}}$ and $\Sigma \equiv \mathsf{initiated}$
  - assert $\mathsf{Verify}(i||c_i, \sigma_{c_i}, pk_{\mathcal{P}}) \equiv 1$
  - assert $\mathsf{Verify}(i||k_i, \sigma_{k_i}, pk_{\mathcal{P}}) \equiv 1$
  - assert $\mathsf{VerifyMTP}(\mathsf{root}_m, i, \pi^i_{\mathsf{MT}}, \mathcal{H}(m_i)) \equiv 1$
  - $m'_i = \mathsf{SDec}(c_i, k_i)$
  - assert $\mathcal{H}(m'_i) \neq \mathcal{H}(m_i)$
  - let $\mathsf{plt} := true$
  - let $\Sigma := \mathsf{received}$ and send $(\mathsf{received})$ to all entities

Figure 4.7 The streaming-setting arbiter functionality $\mathcal{G}_s^{\mathsf{ledger}}$.

- The deliverer $\mathcal{D}$ initializes a variable $y := 1$ and executes as follows:
  - Upon receiving $(\mathsf{initiated}, pk_{\mathcal{C}})$ from $\mathcal{G}_s$, sends $(\mathsf{deliver}, i, c_i, \sigma_{c_i}), i = 1$ to $\mathcal{C}$ and starts a timer $\mathcal{T}_{\mathsf{chunkReceipt}}$;
  - Upon receiving the message $(\mathsf{receipt}, i, \sigma^i_{\mathcal{CD}})$ from $\mathcal{C}$ before $\mathcal{T}_{\mathsf{chunkReceipt}}$ times out, checks whether $\mathsf{Verify}(\mathsf{receipt}||i||pk_{\mathcal{C}}||pk_{\mathcal{D}}, \sigma^i_{\mathcal{CD}}, pk_{\mathcal{C}}) \equiv 1 \wedge i \equiv y$ or not, if succeed, continues with the next iteration: sets $y := y + 1$, sends

The Arbiter Contract Functionality $\mathcal{G}_s^{\mathsf{ledger}}$ for p2p Streaming (Cont.)

Phase 3: Payout

- On receive $(\mathsf{claimDelivery}, i, \sigma^i_{\mathcal{CD}})$ from $\mathcal{D}$:
  - assert current time $\mathcal{T} < \mathcal{T}_{\mathsf{finish}}$
  - assert $i \equiv n$ or $\Sigma \equiv \mathsf{received}$ or $\Sigma \equiv \mathsf{payingRevealing}$
  - assert $\mathsf{ctr} \equiv 0$ and $0 < i \leq n$
  - assert $\mathsf{Verify}(\mathsf{receipt}||i||pk_{\mathcal{C}}||pk_{\mathcal{D}}, \sigma^i_{\mathcal{CD}}, pk_{\mathcal{C}}) \equiv 1$
  - let $\mathsf{ctr}_{\mathcal{D}} := i$, $\Sigma := \mathsf{payingDelivery}$, and then send $(\mathsf{payingDelivery})$ to all entities
- On receive $(\mathsf{claimRevealing}, i, \sigma^i_{\mathcal{CP}})$ from $\mathcal{P}$:
  - assert current time $\mathcal{T} < \mathcal{T}_{\mathsf{finish}}$
  - assert $i \equiv n$ or $\Sigma \equiv \mathsf{received}$ or $\Sigma \equiv \mathsf{payingDelivery}$
  - assert $\mathsf{ctr} \equiv 0$ and $0 < i \leq n$
  - assert $\mathsf{Verify}(\mathsf{receipt}||i||pk_{\mathcal{C}}||pk_{\mathcal{P}}, \sigma^i_{\mathcal{CP}}, pk_{\mathcal{C}}) \equiv 1$
  - let $\mathsf{ctr}_{\mathcal{P}} := i$, $\Sigma := \mathsf{payingRevealing}$, and then send $(\mathsf{payingRevealing})$ to all entities
- Upon $\mathcal{T}_{\mathsf{finish}}$ times out:
  - assert current time $\mathcal{T} \geq \mathcal{T}_{\mathsf{finish}}$
  - let $\mathsf{ctr} := \max\{\mathsf{ctr}_{\mathcal{D}}, \mathsf{ctr}_{\mathcal{P}}\}$
  - let $\mathsf{ledger}[\mathcal{D}] := \mathsf{ledger}[\mathcal{D}] + \mathsf{ctr} \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$
  - if plt:
    let $\mathsf{ledger}[\mathcal{P}] := \mathsf{ledger}[\mathcal{P}] + (n - \mathsf{ctr}) \cdot \ddot{\mathbb{B}}_{\mathcal{P}} + \mathsf{ctr} \cdot \ddot{\mathbb{B}}_{\mathcal{C}}$
    let $\mathsf{ledger}[\mathcal{C}] := \mathsf{ledger}[\mathcal{C}] + (n - \mathsf{ctr}) \cdot \ddot{\mathbb{B}}_{\mathcal{C}} + \ddot{\mathbb{B}}_{\mathsf{pf}}$
  - else:
    let $\mathsf{ledger}[\mathcal{P}] := \mathsf{ledger}[\mathcal{P}] + (n - \mathsf{ctr}) \cdot \ddot{\mathbb{B}}_{\mathcal{P}} + \mathsf{ctr} \cdot \ddot{\mathbb{B}}_{\mathcal{C}} + \ddot{\mathbb{B}}_{\mathsf{pf}}$
    let $\mathsf{ledger}[\mathcal{C}] := \mathsf{ledger}[\mathcal{C}] + (n - \mathsf{ctr}) \cdot \ddot{\mathbb{B}}_{\mathcal{C}}$
  - if $\mathsf{ctr} > 0$:
    let $\Sigma := \mathsf{sold}$ and send (sold) to all entities
  - else let $\Sigma := \mathsf{not\_sold}$ and send $(\mathsf{not\_sold})$ to all entities

▷ Reset to the ready state for repeatable delivery

- On receive (reset) from $\mathcal{P}$:
  - assert $\Sigma \equiv \mathsf{sold}$ or $\Sigma \equiv \mathsf{not\_sold}$
  - set ctr, $\mathsf{ctr}_{\mathcal{D}}$, $\mathsf{ctr}_{\mathcal{P}}$, $\mathcal{T}_{\mathsf{receive}}$, $\mathcal{T}_{\mathsf{finish}}$ as 0
  - nullify $pk_{\mathcal{C}}$
  - let $\theta := \theta - 1$ and $\Sigma := \mathsf{ready}$
  - send (ready) to all entities

Figure 4.8 The continuation of streaming-setting arbiter functionality $\mathcal{G}_s^{\mathsf{ledger}}$.

$(\mathsf{deliver}, i, c_i, \sigma_{c_i}), i = y$ to $\mathcal{C}$, and resets $\mathcal{T}_{\mathsf{chunkReceipt}}$ (i.e., the step (1) in Figure 4.6); otherwise $\mathcal{T}_{\mathsf{chunkReceipt}}$ times out, enters the next phase.

- The provider $\mathcal{P}$ initializes a variable $z := 1$ and executes as follows in this phase:
  - Upon receiving $(\mathsf{initiated}, pk_{\mathcal{C}})$ from $\mathcal{G}_s$: asserts $\Sigma \equiv \mathsf{initiated}$, and sends $(\mathsf{mtree}, \mathsf{MT}, \sigma_{\mathcal{P}}^{\mathsf{MT}})$ to $\mathcal{C}$;
  - Upon receiving the message $(\mathsf{keyReq}, i, \sigma_{\mathcal{C}}^{i})$ from $\mathcal{C}$, checks whether $i \equiv z \wedge \mathsf{Verify}(i||pk_{\mathcal{C}}, \sigma_{\mathcal{C}}^{i}, pk_{\mathcal{C}}) \equiv 1$, if succeed, sends $(\mathsf{reveal}, i, k_i, \sigma_{k_i})$, where $\sigma_{k_i} \leftarrow \mathsf{Sign}(i||k_i, sk_{\mathcal{P}})$, to $\mathcal{C}$ and starts (for $i \equiv 1$) a timer $\mathcal{T}_{\mathsf{keyReceipt}}$ or resets (for $1 < i \leq n$) it (i.e., the step (3) in Figure 4.6), otherwise enters the next phase;
  - On input $(\mathsf{receipt}, i, \sigma_{\mathcal{CP}}^{i})$ from $\mathcal{C}$ before $\mathcal{T}_{\mathsf{keyReceipt}}$ expires, checks whether $\mathsf{Verify}(\mathsf{receipt}||i||pk_{\mathcal{C}}||pk_{\mathcal{P}}, \sigma_{\mathcal{CP}}^{i}, pk_{\mathcal{C}}) \equiv 1 \wedge i \equiv z$ or not, if succeed, sets $z = z + 1$. Otherwise $\mathcal{T}_{\mathsf{keyReceipt}}$ times out, enters the next phase.

Phase III for Payout. The provider $\mathcal{P}$ and the deliverer $\mathcal{D}$ interact with the contract $\mathcal{G}_s$ in this phase as:

- The provider $\mathcal{P}$ executes as follows in this phase:
  - Upon receiving $(\mathsf{received})$ or $(\mathsf{delivered})$ from $\mathcal{G}_s$, or receiving the n-th $\mathsf{receipt}$ from $\mathcal{C}$ (i.e., z is set to be $n + 1$), sends $(\mathsf{claimRevealing}, i, \sigma_{\mathcal{CP}}^{i})$ to $\mathcal{G}_s$;
  - Waits for $(\mathsf{revealed})$ from $\mathcal{G}_s$ to halt.
- The deliverer $\mathcal{D}$ executes as follows during this phase:
  - Upon receiving $(\mathsf{received})$ or $(\mathsf{revealed})$ from $\mathcal{G}_s$, or receiving the n-th $\mathsf{receipt}$ from $\mathcal{C}$ (i.e., y is set to be $n + 1$), sends $(\mathsf{claimDelivery}, i, \sigma_{\mathcal{CD}}^{i})$ to $\mathcal{G}_s$;
  - Waits for $(\mathsf{delivered})$ from $\mathcal{G}_s$ to halt.

4.7.4 Protocol Analysis

Lemma 7. *Conditioned that all parties $\mathcal{P}$, $\mathcal{D}$ and $\mathcal{C}$ are honest, Π_{FS} satisfies the completeness property in the synchronous authenticated network model and stand-alone setting.*

Proof. If all parties $\mathcal{P}$, $\mathcal{D}$ and $\mathcal{C}$ are honest to follow the protocol, the completeness is obvious to see: the provider $\mathcal{P}$ receives a net income of $n \cdot (\ddot{\mathrm{B}}_{\mathcal{C}} - \ddot{\mathrm{B}}_{\mathcal{P}})$; the deliverer $\mathcal{D}$ obtains the payment of $n \cdot \ddot{\mathrm{B}}_{\mathcal{P}}$; the consumer $\mathcal{C}$ pays for $n \cdot \ddot{\mathrm{B}}_{\mathcal{C}}$ and attains the valid content m with $\phi(m) \equiv 1$.

Lemma 8. *In the synchronous authenticated network model and stand-alone setting, conditioned that the underlying cryptographic primitives are secure,* Π_{FS} *meets the fairness requirement even when at most two parties of* $\mathcal{P}$*,* $\mathcal{D}$ *and* $\mathcal{C}$ *are corrupted by non-adaptive P.P.T. adversary* $\mathcal{A}$*.*

Proof. The fairness for each party can be reduced to the underlying cryptographic building blocks. Specifically,

- *Consumer fairness*. The consumer fairness means that the honest $\mathcal{C}$ needs to pay proportional to what it *de facto* receives even though malicious $\mathcal{P}^*$ and $\mathcal{D}^*$ may collude with each other. This case can be modeled as a non-adaptive P.P.T. adversary $\mathcal{A}$ corrupts $\mathcal{P}$ and $\mathcal{D}$ to provide and deliver the content to $\mathcal{C}$. During the *Stream* phase, $\mathcal{C}$ can stop sending back the receipts any time when an invalid chunk is received and then raise complaint to the contract to get compensation. Considering that $\mathcal{C}$ receives a sequence of $(c_1, \sigma_{c_1}), \cdots, (c_\ell, \sigma_{c_\ell}), \ell \in [n]$ though $\mathcal{A}$ may abort maliciously. Then it is ensured that $\mathcal{A}$ can *at most* get ℓ receipts and claim payment of $\ell \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$ and $\ell \cdot \ddot{\mathbb{B}}_{\mathcal{C}}$, where the former is paid by $\mathcal{A}$ itself due to collusion. Overall, $\mathcal{C}$ either pays $\ell \cdot \ddot{\mathbb{B}}_{\mathcal{C}}$ and obtains ℓ valid chunks or pays nothing. To violate the fairness for $\mathcal{C}$, $\mathcal{A}$ has to break the security of signature scheme, i.e., forge $\mathcal{C}$'s signature. The probability is negligible due to the EU-CMA property of the underlying signature scheme. Therefore, the consumer fairness being against the collusion of malicious $\mathcal{P}^*$ and $\mathcal{D}^*$ is ensured. Note that breaking the security of the Merkle tree (i.e., finding another chunk $m'_i \neq m_i$ in position i of m to bind to the same root_m so as to fool the contract to reject $\mathcal{C}$'s PoM) or controlling the execution of smart contract in blockchain, which are of negligible probability due to the second-preimage resistance of hash function in Merkle tree and the fact that contract is modeled as an ideal functionality, can only repudiate the penalty fee $\ddot{\mathbb{B}}_{\mathsf{pf}}$ and would not impact $\mathcal{C}$'s fairness in the streaming setting.

- *Deliverer fairness*. The deliverer fairness states that the honest $\mathcal{D}$ receives the payment proportional to the contributed bandwidth even though the malicious $\mathcal{P}^*$ and $\mathcal{C}^*$ may collude with each other. This case can be modeled as the non-adaptive P.P.T. adversary $\mathcal{A}$ corrupts both $\mathcal{P}$ and $\mathcal{C}$ to reap $\mathcal{D}$'s bandwidth resource without paying. In the *Stream* phase, if the honest $\mathcal{D}$ delivers ℓ chunks, then it is guaranteed to obtain ℓ or $\ell - 1$ (i.e., $\mathcal{A}$ does not respond with the ℓ-th receipt) receipts. In the *Payout* phase, $\mathcal{A}$ cannot lower the payment for the honest $\mathcal{D}$ since $\mathcal{D}$ can send the ℓ-th or $(\ell - 1)$-th receipt to the contract, which would update the internal state $\mathsf{ctr}_{\mathcal{D}}$ as ℓ or $\ell - 1$. Once $\mathcal{T}_{\mathsf{finish}}$ times out, $\mathcal{D}$ can receive the well-deserved payment of $\ell \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$ or $(\ell - 1) \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$ from the contract, and *at most* waste bandwidth for delivering one chunk of size η. To violate the fairness for $\mathcal{D}$, $\mathcal{A}$ has to control the execution of smart contract to refuse $\mathcal{D}$'s

request of claiming payment though the request is valid. The probability to control the contract functionality in blockchain is negligible, and therefore the deliverer fairness being secure against the collusion of malicious $\mathcal{P}^*$ and $\mathcal{C}^*$ is assured.

- *Provider fairness*. The provider fairness indicates that the honest $\mathcal{P}$ receives the payment proportional to the number of valid chunks that $\mathcal{C}$ receives. The malicious $\mathcal{D}^*$ and $\mathcal{C}^*$ may collude with each other or $\mathcal{D}^*$ can costlessly create multiple fake $\mathcal{C}^*$ (i.e., Sybil attack), and then cheat $\mathcal{P}$ without truly delivering the content. These cases can be modeled as a non-adaptive P.P.T. adversary $\mathcal{A}$ corrupts both $\mathcal{D}$ and $\mathcal{C}$. There are two situations $\mathcal{P}$'s fairness would be violated: (i) $\mathcal{A}$ claims payment (paid by $\mathcal{P}$) without real delivery; (ii) $\mathcal{A}$ obtains content chunks without paying for $\mathcal{P}$. For case (i), $\mathcal{A}$ would try to maximize the payment paid by $\mathcal{P}$ by increasing the $\mathsf{ctr}_{\mathcal{D}}$ via the ($\mathsf{claimDelivery}$) message sent to the contract. However, the $\mathcal{G}_s$ would update the counter ctr as $\max\{\mathsf{ctr}_{\mathcal{D}}, \mathsf{ctr}_{\mathcal{P}}\}$ in contract after $\mathcal{T}_{\mathsf{finish}}$ times out, and the intention that $\mathcal{A}$ tries to maximize $\mathsf{ctr}_{\mathcal{D}}$ would correspondingly maximize ctr. Considering that $\mathcal{A}$ wants to claim the payment of $\ell \cdot \ddot{\mathbb{B}}_{\mathcal{P}}, \ell \in [n]$ by letting the ($\mathsf{claimDelivery}$) message contain the index of ℓ while no content is actually delivered, essentially the honest $\mathcal{P}$ can correspondingly receive the payment of $\ell \cdot \ddot{\mathbb{B}}_{\mathcal{C}}$, and therefore a well-deserved net income of $\ell \cdot (\ddot{\mathbb{B}}_{\mathcal{C}} - \ddot{\mathbb{B}}_{\mathcal{P}})$, unless $\mathcal{A}$ can manipulate the execution of smart contract. For case (ii), on one hand, each content chunk is encrypted before receiving the corresponding chunk key from $\mathcal{P}$. Hence, $\mathcal{A}$ has to violate the semantic security of the underlying symmetric encryption scheme to break the provider fairness, which is of negligible probability. On the other hand, during the streaming procedure, $\mathcal{P}$ can always stop revealing the chunk key to $\mathcal{A}$ if no valid receipt for the previous chunk key is responded in time. *At most* $\mathcal{P}$ would lose one content chunk of size η and receive well-deserved payment using the latest receipt. To violate the fairness, $\mathcal{A}$ again has to control the execution of smart contract, which is of negligible probability, to deny the payment for $\mathcal{P}$ though the submitted receipt is valid. Therefore, the provider fairness against the collusion of malicious $\mathcal{D}^*$ and $\mathcal{C}^*$ is guaranteed.

In sum, the fairness for $\mathcal{C}$ is strictly ensured in Π_{FS}, while for $\mathcal{P}$ and $\mathcal{D}$, the unpaid revealed content for $\mathcal{P}$ and the unpaid bandwidth resource of delivery are bounded to $O(\eta)$ bits. i.e., Π_{FS} satisfies the defined fairness property.

Lemma 9. *In the synchronous authenticated network and stand-alone model,* Π_{FS} *satisfies the confidentiality property against malicious deliverer corrupted by non-adaptive P.P.T. adversary* $\mathcal{A}$.

Proof. The confidentiality indicates that the deliverer $\mathcal{D}$ cannot learn any useful information about the content m besides a-priori known knowledge within a delivery session. It can be modeled as a non-adaptive P.P.T. adversary corrupts $\mathcal{D}$. In Π_{FS}, the possible scripts of leaking information of m include: (i) the encrypted content chunks delegated to $\mathcal{D}$; and (ii) the Merkle tree MT of the content m. To break the confidentiality property, $\mathcal{A}$ has to violate the pre-image resistance of cryptographic hash functions (for the encryption scheme and MT), which is of negligible probability. Hence, the confidentiality property against the malicious deliverer can be ensured.

Lemma 10. *If at least one of the three parties $\mathcal{P}$, $\mathcal{D}$ and $\mathcal{C}$ is honest and others are corrupted by non-adaptive P.P.T. adversary $\mathcal{A}$, Π_{FS} meets the timeliness property in the synchronous authenticated network and stand-alone model.*

Proof. The timeliness means that the honest parties in Π_{FS} can terminate in $O(n)$ synchronous rounds, where n is the number of content chunks, and when the protocol completes or aborts, the fairness and confidentiality are always preserved. Similarly, we focus on the analysis of fairness since the guarantee of confidentiality can be straightforwardly derived in light of the lemma 9 even if malicious parties abort. We distinguish the following termination cases for Π_{FS} with the arbiter contract $\mathcal{G}_s$ and at least one honest party:

<u>*No abort*</u>. If all of $\mathcal{P}$, $\mathcal{D}$ and $\mathcal{C}$ are honest, the protocol Π_{FS} terminates in the *Payout* phase, after $\mathcal{T}_{\mathsf{finish}}$ times out. Both the *Prepare* and *Payout* phases can be completed in $O(1)$ rounds, while the *Stream* phase needs $O(n)$ rounds, where n is the number of content chunks, resulting in $O(n)$ rounds for the protocol Π_{FS} to terminate and the fairness for all parties at completion are ensured as they obtain the well-deserved items.

<u>*Aborts in the Prepare phase*</u>. The analysis for this phase is the same as the Π_{FD} protocol in lemma 5.

Aborts in the Stream phase. This phase involves the provider $\mathcal{P}$, the deliverer $\mathcal{D}$, the consumer $\mathcal{C}$ and the arbiter contract $\mathcal{G}_s$, and it would terminate in $O(n)$ rounds due to the following cases: (i) $\mathcal{C}$ receives all the chunks and sends the ($\mathsf{received}$) message to contract; (ii) any party aborts during the streaming, and then the timer $\mathcal{T}_{\mathsf{receive}}$ times out in contract; (iii) $\mathcal{C}$ successfully raises complaint of $\mathcal{P}$'s misbehavior. During streaming, if $\mathcal{D}$ aborts, for example, after receiving the ℓ-th receipt for chunk delivery, then $\mathcal{C}$ is guaranteed to have received ℓ encrypted chunks at that time point. If $\mathcal{P}$ aborts, for example, after receiving the ℓ-th receipt for key revealing, then $\mathcal{C}$ is assured to have received ℓ keys for decryption at that time point. If $\mathcal{C}$ aborts, in the worst case, after receiving the ℓ-th encrypted chunk from $\mathcal{D}$ and the ℓ-th key from $\mathcal{P}$, at that time point, $\mathcal{D}$ is ensured to have obtained $\ell - 1$ receipts for the bandwidth contribution, while $\mathcal{P}$ is guaranteed to have received $\ell - 1$ receipts for key revealing, which means the fairness for $\mathcal{D}$ and $\mathcal{P}$ is still preserved according to the fairness definition, i.e., the unpaid delivery resource for $\mathcal{D}$ and the unpaid content for $\mathcal{P}$ are bounded to one chunk of $O(\eta)$ bits.

Aborts in the Payout phase. This phase involves the provider $\mathcal{P}$, the deliverer $\mathcal{D}$ and the arbiter contract $\mathcal{G}_s$, and it can terminate in $O(1)$ rounds. The fairness for the honest one is not impacted no matter when the other party aborts since $\mathcal{P}$ and $\mathcal{D}$ are independently claim the payment from contract. After $\mathcal{T}_{\mathsf{finish}}$ times out, the contract would automatically distribute the payment to all parties according to the internal state ctr.

Lemma 11. *In the synchronous authenticated network model and stand-alone setting, for any non-adaptive P.P.T. adversary $\mathcal{A}$, Π_{FS} satisfies the efficiency requirement: the communication complexity is bounded to $O(n)$; the on-chain cost is bounded to $\widetilde{O}(1)$; the messages transferred by the provider $\mathcal{P}$ after the setup phase are bounded to $n \cdot \lambda$ bits, where n is the number of chunks and λ is a cryptographic parameter, and $n \cdot \lambda$ is much less than the content size $|m|$.*

Proof. The analysis of efficiency guarantee in Π_{FS} can be conducted in the following three perspectives:

- *Communication complexity*. The *Prepare* phase is the same as the downloading setting, and therefore the time complexity is $O(n)$. In the *Stream* phase, $\mathcal{P}$ sends the Merkle tree MT of m and meanwhile $\mathcal{D}$ starts to deliver the delegated n chunks to $\mathcal{C}$. If dispute happens during streaming, the complexity of sending PoM is $O(\log n)$. Overall the communication complexity of this phase is $O(n)$. In the *Payout* phase, the ($\mathsf{claimDelivery}$) and ($\mathsf{claimRevealing}$) messages sent by $\mathcal{P}$ and $\mathcal{D}$ to contract is in $O(1)$. Hence, the total communication complexity of Π_{FS} is $O(n)$.
- *On-chain costs*. The *Prepare* phase yields on-chain costs of $O(1)$, which is same as the downloading setting. In the *Stream* phase, the on-chain cost of the $\mathsf{consume}$ function is $O(1)$ and the multiple rounds of content delivery (i.e., the streaming process) are executed off-chain. When dispute occurs during streaming, the on-chain cost is $O(\log n)$ (for verifying the Merkle proof), leading to a total on-chain costs of $O(\log n)$. In the *Payout* phase, the on-chain costs is $O(1)$ since $\mathcal{P}$ and $\mathcal{D}$ only need to submit the latest receipt consisting of one signature. Overall, the on-chain cost of Π_{FS} is $O(\log n)$, namely $\widetilde{O}(1)$.
- *Message volume for $\mathcal{P}$*. Considering that the contract is deployed and the deliverer is ready to deliver. Every time when a new consumer joins in, a new delivery session starts. The messages that $\mathcal{P}$ needs to send include: (i) the Merkle tree MT of m in the *Stream* phase is $O(\log n)$; (ii) the n chunk keys revealed to $\mathcal{C}$ is $O(n)$. Note that the message volume decrease from n chunks to n keys (e.g., 32 KB for a chunk v.s. 256 bits for a chunk key); (iii) the ($\mathsf{claimRevealing}$) message for claiming payment, which is $O(1)$ since only the latest receipt containing one signature needs to be submitted to $\mathcal{G}_s$. Overall, the resulting message volume can be represented as $n \cdot \lambda$, where λ is a small cryptographic parameter, which is much smaller than the content size $|m|$.

Theorem 2. *Conditioned that the underlying cryptographic primitives are secure, the protocol* $\mathsf{FairStream}$ *satisfies the completeness, fairness, confidentiality against deliverer, timeliness, and non-trivial efficiency properties in the synchronous authenticated network, $\mathcal{G}_s^{\mathsf{ledger}}$-hybrid and stand-alone model.*

Proof. Lemmas 7, 8, 9, 10, and 11 complete the proof.

Besides, we have the following corollary to characterize the latency relationship between $\mathsf{FairDownload}$ and $\mathsf{FairStream}$.

Corollary 1. In the synchronous authenticated setting without corruptions, the honest consumer $\mathcal{C}$ in Π_{FS} can: (i) retrieve the first chunk in $O(1)$ communication rounds once activating the Stream phase; (ii) retrieve every $(i+1)$-th content chunk in $O(1)$ communication rounds once the i-th content chunk has delivered. This yields less retrieval latency compared to that all chunks retrieved by the consumer in Π_{FD} delivers in $O(n)$ rounds after the Deliver phase is activated.

Proof. In Π_{FD}, the honest consumer $\mathcal{C}$ is able to obtain the keys only after the completion of the verifiable fair delivery module to decrypt the received chunks, meaning that the latency of retrieving the raw content chunks is in $O(n)$ communication rounds. While for Π_{FS}, in each round of streaming, the honest $\mathcal{C}$ can obtain one encrypted chunk from the deliverer $\mathcal{D}$ as well as one decryption key from the provider $\mathcal{P}$, and consequently the retrieval latency, though entailing relatively more involvement of $\mathcal{P}$, is only in $O(1)$ communication rounds.

Extension for delivering from any specific chunk. The protocol FairStream (as well as FairDownload) can be easily tuned to transfer the content from the middle instead of the beginning. Specifically, for the downloading setting, one can simply let the content provider reveal the elements that are able to recover a *sub-tree* of the key derivation tree KT for decrypting the transferred chunks. The complaint of incorrect decryption key follows the same procedure in §4.6. For the streaming setting, it is more straightforward as each chunk ciphertext and its decryption key are uniquely identified by the index and can be obtained in $O(1)$ rounds by the consumer, who can immediately complain to contract in the presence of an incorrect decryption result.

4.8 Implementation and Evaluations

To shed some light on the feasibility of FairDownload and FairStream, we implement, deploy and evaluate them in the *Ethereum Ropsten* network. The arbiter contract

is implemented in Solidity and split into *Optimistic* and *Pessimistic* modules, where the former is executed when no dispute occurs while the later is additionally called if dispute happens. Note that the contracts are only deployed once and may be used for multiple times to facilitate many deliveries, which amortizes the cost of deployment.

Cryptographic instantiations. The hash function is *keccak256* and the digital signature is via ECDSA over secp256k1 curve. The encryption of each chunk m_i with key k_i is instantiated as: parse m_i into t 32-byte blocks $(m_{i,1}, \ldots, m_{i,t})$ and output $c_i = (m_{i,1} \oplus \mathcal{H}(k_i||1), \ldots, m_{i,t} \oplus \mathcal{H}(k_i||t))$. The decryption is same to the encryption. We construct public key encryption scheme based on ElGamal: Let $\mathcal{G} = \langle g \rangle$ to be G_1 group over *alt-bn128* curve [122] of prime order q, where g is group generator; The private key $k \xleftarrow{R} \mathbb{Z}_q$, the public key $h = g^k$, the encryption $\mathsf{VEnc}_h(m) = (c_1, c_2) = (g^r, m \cdot g^{kr})$ where $r \xleftarrow{R} \mathbb{Z}_q$ and m is encoded into $\mathcal{G}$ with Koblitz's method [83], and the decryption $\mathsf{VDec}_k((c_1, c_2)) = c_2/c_1^k$. To augment ElGamal for verifiable decryption, we adopt Schnorr protocol [125] for Diffie-Hellman tuples with using Fiat-Shamir transform [46] in the random oracle model. Specifically, $\mathsf{ProvePKE}_k((c_1, c_2))$ is as: run $\mathsf{VDec}_k((c_1, c_2))$ to obtain m. Let $x \xleftarrow{R} \mathbb{Z}_q$, and compute $A = g^x$, $B = c_1^x$, $C = \mathcal{H}(g||A||B||h||c_1||c_2||m)$, $Z = x + kC$, $\pi = (A, B, Z)$, and output (m, π); $\mathsf{VerifyPKE}_h((c_1, c_2), m, \pi)$ is as: parse π to obtain (A, B, Z), compute $C' = \mathcal{H}(g||A||B||h||c_1||c_2||m)$, and verify $(g^Z \equiv A \cdot h^{C'}) \wedge (m^{C'} \cdot c_1^Z \equiv B \cdot c_2^{C'})$, and output 1/0 indicating the verification succeeds or fails.

4.8.1 Evaluating Downloading Protocol

Table 4.2 presents the on-chain costs for all functions in Π_{FD}. For the recent violent fluctuation of Ether price, we adopt a gas price at 10 Gwei to ensure over half of the mining power in Ethereum would mine this transaction[8], and an exchange rate of 259.4 USD per Ether, which is the average market price of Ether between

[8]https://ethgasstation.info/. Retrieved on March 6th, 2022.

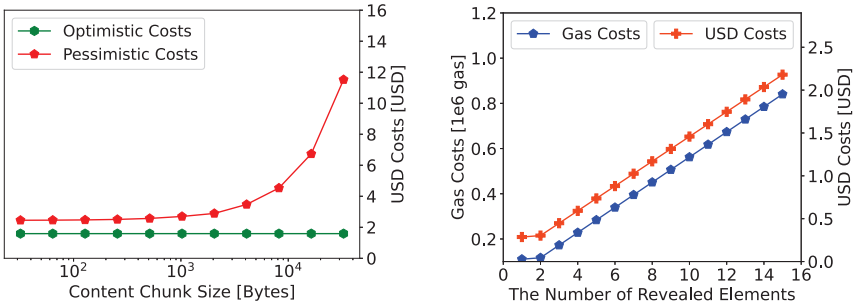
Table 4.2 The On-Chain Costs of All Functions in FairDownload

Phase	*Function*	*Caller*	*Gas Costs*	USD Costs
Deploy	(Optimistic)	$\mathcal{P}$	2 936 458	7.617
Deploy	(Pessimistic)	$\mathcal{P}$	2 910 652	7.550
Prepare	start	$\mathcal{P}$	110 751	0.287
	join	$\mathcal{D}$	69 031	0.179
	prepared	$\mathcal{D}$	34 867	0.090
Deliver	consume	$\mathcal{C}$	117 357	0.304
	delivered	$\mathcal{C}$	57 935	0.150
	verifyVFDProof	$\mathcal{D}$	56 225	0.146
Reveal	revealKeys	$\mathcal{P}$	113 041	0.293
	payout	$\mathcal{G}_d$	53 822	0.139
Dispute Resolution	wrongRK	$\mathcal{C}$	23 441	0.061
	PoM	$\mathcal{C}$	389 050	1.009

Jan./1st/2020 and Nov./3rd/2020 from coindesk[9]. We stress that utilizing other cryptocurrencies such as Ethereum classic[10] can much further decrease the price for execution. The price also applies to the streaming setting.

Optimistic costs. Without complaint the protocol Π_{FD} only executes the functions in *Deliver* and *Reveal* phases when a new consumer joins in, yielding the total cost of 1.032 USD for all involved parties except the one-time cost for deployment and the *Prepare* phase. Typically, such an on-chain cost is *constant* no matter how large the content size or the chunk size are, as illustrated in Figure 4.9(a). In a worse case, up to $\log n$ elements in Merkle tree need to be revealed. In that case, Figure 4.9(b) depicts the relationship between the number of revealed elements and the corresponding costs.

[9]https://www.coindesk.com/price/ethereum/. Retrieved on March 6th, 2022.
[10]https://ethereumclassic.org/. Retrieved on March 6th, 2022.



(a) Costs for various chunk size (b) Costs for *erk* revealing cost

Figure 4.9 The experiment results for the FairDownload protocol.

4.8.2 Evaluating Streaming Protocol

Table 4.3 illustrates the on-chain costs of all functions in FairStream. As the deployment of contract and the *Prepare* phase can be executed only *once*, we discuss the costs in both optimistic and pessimistic modes after a new consumer participates in, i.e., starting from the *Stream* phase. Specifically,

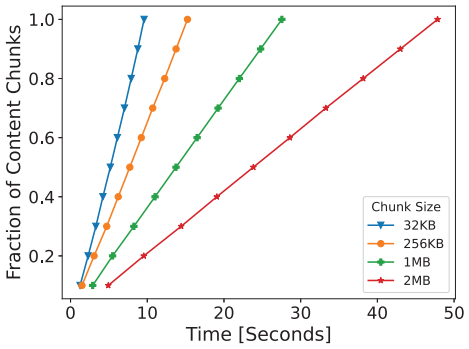
Optimistic costs. When no dispute occurs, the Π_{FS} protocol executes the functions in *Stream* and *Payout* phases except the PoM function for verifying proof of misbehavior, yielding a total cost of 0.933 USD for all involved parties. Note that only one of the (received) and (receiveTimeout) functions would be invoked. Meanwhile, the (claimDelivery) and (claimRevealing) functions may be called in different orders. The costs in the optimistic mode is *constant* regardless of the content and chunk size.

Pessimistic costs. When complaint arises, the total on-chain cost is 1.167 USD for all involved parties during a delivery session. The cost of the PoM function: (i) increases slightly in number of chunks n, since it computes $O(\log n)$ hashes to verify the Merkle tree proof; (ii) increase linearly in the the content chunk size η due to the chunk decryption in contract, which follows a similar trend to Figure 4.9(a) pessimistic costs but with lower costs since no verification of verifiable decryption proof is needed.

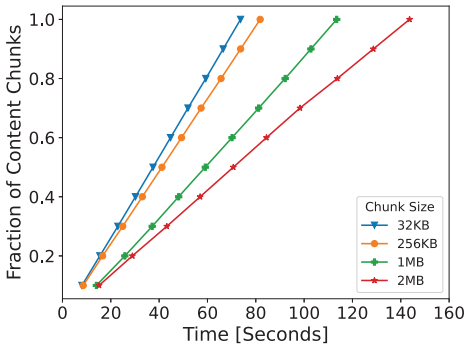
Table 4.3 The On-Chain Costs of All Functions in FairStream

Phase	Function	Caller	Gas Costs	USD Costs
Deploy	(Optimistic)	$\mathcal{P}$	1 808 281	4.691
Deploy	(Pessimistic)	$\mathcal{P}$	1 023 414	2.655
Prepare	start	$\mathcal{P}$	131 061	0.340
	join	$\mathcal{D}$	54 131	0.140
	prepared	$\mathcal{D}$	34 935	0.091
Stream	consume	$\mathcal{C}$	95 779	0.248
	received	$\mathcal{C}$	39 857	0.103
	receiveTimeout	$\mathcal{G}_s$	39 839	0.103
	PoM	$\mathcal{C}$	90 018	0.234
Payout	claimDelivery	$\mathcal{D}$	67 910	0.176
	claimRevealing	$\mathcal{P}$	67 909	0.176
	finishTimeout	$\mathcal{G}_s$	88 599	0.230

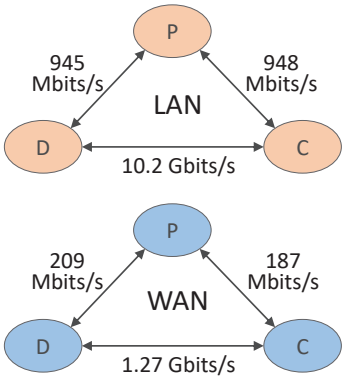
Streaming efficiency. To demonstrate feasibility of using Π_{FS} for p2p streaming, we evaluate the efficiency for streaming 512 content chunks with various chunk size. Figure 4.10(c) shows the experimental bandwidth among parties in LAN (i.e., three VM instances on three servers residing on the same rack connected with different switches, where servers are all Dell PowerEdge R740 and each is equipped with 2 Intel(R) Xeon(R) CPU Silver 4114 processors, 256 GB (16 slots×16 GB/slot) 2400MHz DDR4 RDIMM memory and 8 TB (8 slots×1TB/slot) 2.5 inch SATA hard drive. Each VM on the servers has the same configuration of 8 vCPUs, 24 GB memory and 800 GB hard drive) and WAN (i.e., three *Google cloud* VM instances are initialized in *us-east4-c*, *us-east1-b* and *europe-north1-a*, respectively. Each VM is configured with 2 vCPUs, 4 GB memory and 10 GB hard drive). Considering that $\mathcal{P}$



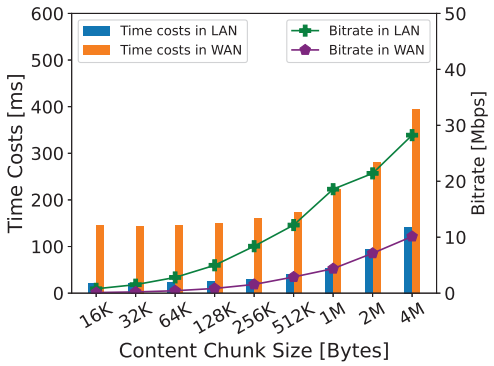
(a) Time costs of streaming 512 content chunks in LAN



(b) Time costs of streaming 512 content chunks in WAN



(c) Bandwidths among entities in the testing



(d) Average time costs and the corresponding bitrate for various chunk size

Figure 4.10 The performance of FairStream protocol in the LAN and WAN.

owns information to choose the proper deliverer $\mathcal{D}$ to ensure better delivery quality (e.g., less delay from $\mathcal{D}$ to $\mathcal{C}$), the link between $\mathcal{D}$ and $\mathcal{C}$ is therefore evaluated in a higher bandwidth environment. Figure 4.10(a) and 4.10(b) illustrates the experiment results of consecutively streaming 512 content chunks in both LAN and WAN and the corresponding time costs. We can derive the following observations: (i) obviously the time costs increase due to the growth of chunk size; (ii) the delivery process remains stable with only slight fluctuation, as reflected by the slope for each chunk size in Figure 4.10(a) and 4.10(b). Furthermore, Figure 4.10(d) depicts the average time costs for each chunk (over the 512 chunks) and the corresponding bitrate. The results show that the bitrate can reach 10 Mpbs even in the public network, which is potentially sufficient to support high-quality content streaming. For instance,

the video bitrate for HD 720 and HD 1080 are *at most* 4 Mbps and 8 Mbps, respectively [66].

4.9 Summary

In this chapter, we target the fairness problem in the P2P content delivery setting where we leverage blockchain to play the role of a trusted third party by conducting computation and verification in the presence of malicious behaviours. Specifically, we present the first two fair P2P content delivery protocols atop blockchains to support *fair P2P downloading* and *fair P2P streaming*, respectively. They enjoy strong fairness guarantees to protect any of the content provider, the content consumer, and the content deliverer from being ripped off by other colluding parties. Detailed complexity analysis and extensive experiments of prototype implementations are performed on top of real public blockchain platform (i.e., Ethereum). The measurements demonstrate that the proposed protocols are highly efficient.

CHAPTER 5

SUMMARY OF THE DISSERTATION

5.1 Conclusion

Blockchain is a disruptive technology that provides the promising infrastructure for the construction of future value transfer network, which advances the information transfer network embodied by Internet. By combining several computer science principles such as distributed system, cryptography, data structures, consensus algorithms, etc., it essentially offers many highly desirable properties (provenance, immutability, finality, transparency, availability, consistency, accountability etc.) that cannot be realized simultaneously in conventional protocol designs for different applications. In this dissertation, we first give a general architecture for blockchain-based decentralized applications and highlight two main functionalities that provided by blockchain, namely storage and computation. The key observation lies in the fact that blockchain is powerful to augment conventional centralized application designs, however, the transferring to blockchain-based decentralized architecture is non-trivial, which may suffer from many security and efficiency issues if not carefully treated. To this end, we employ three concrete application scenarios (i.e., IoT, cyber security, and P2P content delivery) to elaborate the solutions to the potential issues that emerged in the general blockchain-based decentralized application architecture.

Firstly, we observed and solved the censorship problem between external network and blockchain network. This is interesting since most works mainly highlight the advantages that blockchain itself can provide while the security of interactions with external world has been ignored. We proposed the design to achieve censorship resistance in the specific (industrial) IoT setting where data may either flow from the sensor network to the blockchain network (called inbound flow) or maybe the

other way around (called outbound flow, e.g., sending command from blockchain network to actuators). For the inbound flow, we leveraged a gossip-based diffusion mechanism and proposed an augmented consensus design while for the outbound flow, we proposed the mechanism to query from multiple parties and signature aggregation scheme is introduced to reduce the verification and communication complexity.

Secondly, we utilized the permissioned blockchain (i.e., Hyperledger Fabric) to provide the functionality of robust storage in the concrete cyber security management setting. Blockchain offers the similar functionality with conventional distributed database but with security guarantee, i.e., the full nodes in the blockchain network would reach consensus about the data before writing into ledger and the data retrieval can only be performed by specified operators whose privileges can be defined as access control policies maintaining in the smart contract. In the blockchain-enabled decentralized cyber security management system, a set of challenges require proper handling, e.g., to handle a large volume of cyber data, we proposed two methods where the first one splits the cyber data into chunks and then performs concurrent writing; the second one incorporates DSN by storing real cyber data into DSN while recording its reference on-chain. Several limitations and future extensions regarding this topic have been listed to inspire more investigations for this important topic. Noticeably, in effect, these designs can also be applied to any information management system that empowered by blockchain.

Thirdly, we focus on the computation functionality that blockchain provides and consider the specific P2P content delivery application scenario. We observed that traditional fairness definition is insufficient in the context of P2P content delivery. We thus proposed a more fine-grained fairness definition. We then leverage blockchain to play the role of traditionally assumed trusted third party, which has been proved necessary to ensure complete fairness. Blockchain can perform verification whenever the system participants misbehave, and also monetarily incentivize ones to

proactively join in due to the fact that it supports transparent coin transfer upon a certain conditions are satisfied. Our designs also considered many non-trivial system optimizations, e.g., minimized on-chain storage and computational costs, optimized delivery communication and necessary privacy guarantee.

Overall, surrounding the two main functionalities that blockchain provide, i.e., storage and computation, we employ three concrete application scenarios to elaborate the design of tackling all potential problems residing in a general blockchain-empowered decentralized application architecture, with the ultimate goal of making blockchain-based systems practical in real-life usage.

5.2 Reflection

Our proposed design in this dissertation can essentially be applied to more general scenarios. With these designs, the following reflections are highlighted:

Blockchain is powerful but not panacea. Blockchain itself can provide many aforementioned desirable properties. However, it is not panacea and for specific application scenario it may require concrete design. For example, blockchain is transparent to record state and handle execution logic, which leads to privacy concern. Thus for sensitive information posted on-chain or the privacy-preserved computation, the confidentiality or privacy properties need to be considered. Another shortcoming lies in its limited computation power, especially for public chain, thus minimizing on-chain computational costs is required for any practical protocol design.

Concrete instantiation and extension. Though blockchain enables robust storage and reliable computation, it still requires concrete instantiation. For example, *proper blockchain type*: a permissioned private blockchain is suitable for one organization to robustly manage data, a permissioned consortium blockchain is usually for multiple organizations to jointly conducting activities, e.g., information sharing. A public chain used in a permissionless environment allows anyone to join in or leave and is

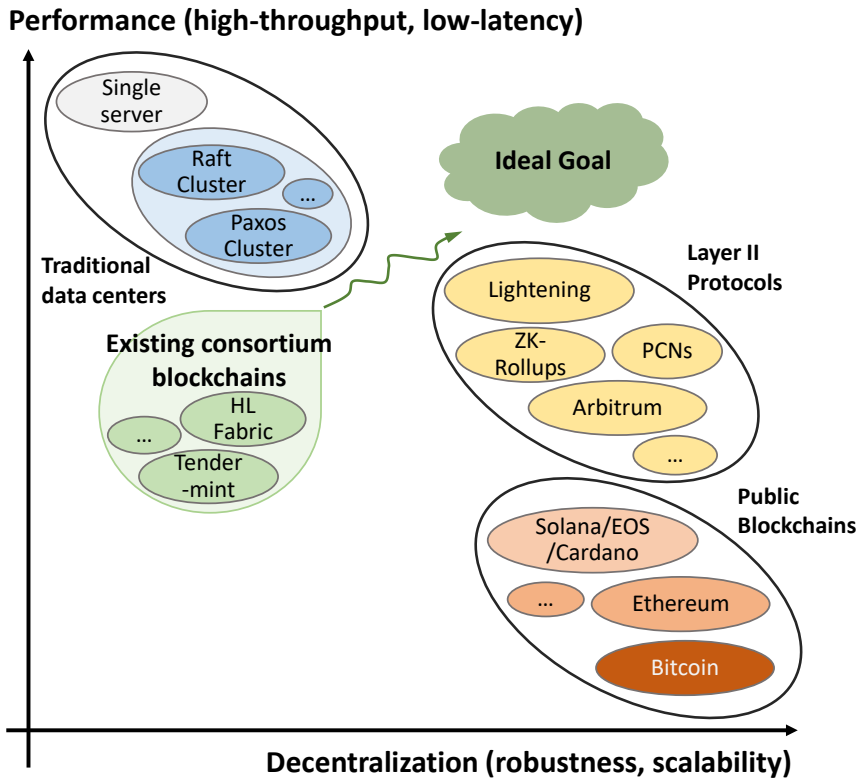


Figure 5.1 The trade-off between decentralization and performance.

usually for computation; *consensus type*: crash fault tolerant consensus mechanisms (compared with byzantine fault tolerant ones) are typically more efficient in terms of higher throughput or less latency yet require all participants to honestly behave; *extension with external components*: blockchain can be combined with other dedicated components to exhibit better capabilities, e.g., the combination with DSNs can reduce the on-chain storage costs, the combination with machine learning frameworks can offload learning tasks thus reducing on-chain computational costs; the combination with big data frameworks, e.g., Hadoop or Spark, can delegate the distributed computation to off-chain platforms thus also for reducing on-chain costs. However, the integration with these components requires deliberate design thinking.

Decentralization vs. efficiency. Blockchain-enabled decentralized applications gain the benefits of being more secure and robust, which, however, is traded by the performance sacrifice [96]. As shown in Figure 5.1, traditional data centers under-

pinned by single centralized server center or CFT algorithms possess relatively better performance (high throughput and low latency). The public blockchain platforms have the worst performance but with strongest security and robustness guarantee. The layer II protocols aim to improve the scalability of the original blockchain platforms (either permissioned or permissionless) thus exhibit better performance. Permissioned blockchain platforms typically have better performance than the public ones due to the underlying consensus mechanism (e.g., PBFT for the former vs. PoW for the latter). An ideal ultimate goal would be realizing decentralization while retaining the performance [129]. In practical blockchain-based decentralized applications, finding the balance between decentralization and performance is vital and needs comprehensive consideration of the system design goals.

Provably secure protocol design. Following the paradigm of modern cryptography, it is important to formally argue about the security of proposed blockchain-based cryptography-related designs or protocols. This is done via applying rigorous logical argumentation in the form of mathematical proofs. There are three fundamental principles that need to be specified for provably secure protocols: (i) *formal definitions*. A formal definition of the proposed protocol's required properties by defining the threat model, i.e., the capabilities that an adversary has. The goal of the adversary is formulated either in a game or in a simulation-based manner; (ii) *precise assumptions*. The assumptions are usually hard problems in the theory of computational complexity that seem far from answered today, e.g., factorization of large numbers. The security guarantee of the proposed protocols can be reduced to resolve these well-studied hard problems; (iii) *proofs of security*. Based on the definitions and assumptions, the proposed protocols need to be proven secure. In the game based security, we say a protocol is secure if the adversary's advantage is at most negligible in the security parameter, while in the simulation based security definition, the protocol is secure if the adversary cannot computationally distinguish

between the real-world protocol execution and its simulated version of the security experiment in polynomial time. Besides the basic principles, another perspective of rigorous security proofs usually distinguishes between the *standalone model* and the *Universally Composable (UC) model*. In the former, the security of the proposed protocol is analyzed in an isolated single execution manner, while for the latter, it captures the security of concurrent (parallel and sequential) protocol executions between many parties and even in composition with other protocols.

5.3 Future Vision

This dissertation presents the secure designs to tackle the potential problems that appear in a general blockchain-based decentralized application architecture including data flow between external networks with blockchain network, blockchain-based information management system, and blockchain-empowered reliable computation. Yet there still exist many unexplored and related problems, which naturally form interesting future extensions.

For the specific application scenarios presented in this dissertation, more studies can be conducted. For instance, in the cyber security management case, more CSM functions can be identified, more reliable identifiers for the cyber devices can be integrated, and desirable cyber intelligence sharing mechanism can be further investigated. In the P2P content delivery case, it is feasible to incorporate the digital right management schemes to preserve digital rights for sold contents, and a deliverer selection mechanism can be developed to ensure better delivery performance.

Besides the scenarios investigated in this dissertation, future studies would explore more blockchain-based decentralized application settings. One highlighted direction is decentralized identity, also referred to as *self-sovereign identity (SSI)*, which is anticipated to be the next-generation online identity model. Since SSI stresses the ownership and management of identity information by users themselves

(instead of relying on any third party), blockchain naturally can be employed to play such a role of a trusted third party. Decentralized identity can advance many existing applications in a more efficient and secure way. For example, in the (industrial) IoT setting, equipping each device with a unique decentralized identity is significant to construct desired security mechanisms including authentication, authorization and secure communication; in the cyber security management setting, the unique and sybil-resistant decentralized identifier for each cyber device can be used to replace the IP address, thus realizing identification and accountability etc.; in the context of multi-party agreement signing, the verifiable credentials in decentralized identity can be utilized to accelerate the agreement filling procedure, and the privacy of signers' identity information can also be preserved, e.g., a signer can use a credential containing her age information to fill in an agreement with a statement and a zero-knowledge proof about her age instead of filling in the real birth of date. Other general application settings include, e.g., big data (usually existing storage in big data ecosystem is only crash fault tolerant, e.g., HDFS in Hadoop ecosystem), machine learning (e.g., *federated learning* which leverages blockchain to provide a decentralized aggregator for trained sub-data sets) and so forth.

With these specific application scenarios, another interesting study lies in building a general framework for converting centralized applications to blockchain-based decentralized ones. Such a framework requires the formal model of blockchain, e.g., [84], the generalized abstraction of centralized application architecture, the blockchain-empowered decentralized application abstraction, and the critical thinking (e.g., *determinism* requirement in smart contract-based computation) of gap filling-in between the two abstractions.

REFERENCES

[1] Mohammad Thoip Abdullah, Sulhan Qidri, Wadi Nuryadi, and Septian Rheno Widianto. Failover cluster nodes and iscsi storage area network on virtualization windows server 2016. *Jurnal Online Informatika*, pages 89–96, 2020.

[2] Md Sahrom Abu, Siti Rahayu Selamat, Aswami Ariffin, and Robiah Yusof. Cyber threat intelligence–issue and challenges. *Indonesian Journal of Electrical Engineering and Computer Science*, pages 371–379, 2018.

[3] Akamai. `https://www.akamai.com/`. Accessed on December 2021.

[4] BSA-The Software Alliance and Galexia. Asia-pacific cybersecurity dashboard - a path to a secure global cyberspace. `http://www.bsa.org/APACcybersecurity`. Accessed on December 2021.

[5] Ghada Almashaqbeh. *CacheCash: A Cryptocurrency-based Decentralized Content Delivery Network*. PhD thesis, Columbia University, 2019.

[6] Hendrik Amler, Lisa Eckey, Sebastian Faust, Marcel Kaiser, Philipp Sandner, and Benjamin Schlosser. Defi-ning defi: Challenges and pathway. *arXiv Preprint arXiv:2101.05589*, 2021.

[7] Peter Amthor, Daniel Fischer, Winfried E Kühnhauser, and Dirk Stelzer. Automated cyber threat sensing and responding: Integrating threat intelligence into security-policy-controlled systems. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, pages 1–10. Canterbury CA, United Kingdom: ACM, 2019.

[8] Priyanka Andrew and Imad Antonios. Asynchronous gossip-based data propagation protocol. In *2015 Fifth International Conference on Digital Information and Communication Technology and Its Applications (DICTAP)*, pages 7–12. Beirut, Lebanon: IEEE, 2015.

[9] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the 13th EuroSys Conference (EUROSYS)*, pages 1–15. Porto, Portugal: ACM, 2018.

[10] Nasreen Anjum, Dmytro Karamshuk, Mohammad Shikh-Bahaei, and Nishanth Sastry. Survey on peer-assisted content delivery networks. *Computer Networks*, pages 79–95, 2017.

[11] Apache. Hbase. `https://hbase.apache.org/`. Accessed on March 2022.

[12] Nadarajah Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, pages 593–610, 2000.

[13] Mira Belenkiy, Melissa Chase, C Chris Erway, John Jannotti, Alptekin Küpçü, Anna Lysyanskaya, and Eric Rachlin. Making p2p accountable without losing privacy. In *Proceedings of the ACM Workshop on Privacy in Electronic Society*, pages 31–40, 2007.

[14] Juan Benet. IPFS-content addressed, versioned, p2p file system. *arXiv Preprint arXiv:1407.3561*, 2014.

[15] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *Advances in Cryptology – CRYPTO*, pages 421–439, 2014.

[16] Alysson Bessani, Joao Sousa, and Eduardo EP Alchieri. State machine replication for the masses with bft-smart. In *the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 355–362. Atlanta, GA, USA: IEEE, 2014.

[17] Manuel Blum. How to exchange (secret) keys. In *Proceedings of ACM Transactions on Computer Systems (TOCS)*, pages 175–193, 1983.

[18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. Cryptology ePrint Archive, Report 2018/483, 2018. https://eprint.iacr.org/2018/483.

[19] Paul Brody and Veena Pureswaran. Device democracy: Saving the future of the Internet of Things. `https://www.ibm.com/downloads/cas/Y5ONA8EV`. Accessed on December 2021.

[20] Sarah Brown, Joep Gommers, and Oscar Serrano. From cyber security information sharing to threat management. In *Proceedings of the 2nd ACM Workshop on Information Sharing and Collaborative Security*, pages 43–49. Denver, Colorado, USA: ACM, 2015.

[21] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Advances in Cryptology – CRYPTO*, pages 126–144, 2020.

[22] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. Newport Beach, CA, USA: IEEE, 2001.

[23] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *the 3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 173–186. Usenix, 1999.

[24] Shi-Cho Cha, Jyun-Fu Chen, Chunhua Su, and Kuo-Hui Yeh. A blockchain connected gateway for BLE-based devices in the internet of things. *IEEE Access*, pages 24639–24649, 2018.

[25] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 719–728. Dallas Texas, USA: ACM, 2017.

[26] Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC)*, pages 364–369. ACM, 1986.

[27] Cloudflare. `https://www.cloudflare.com/`. Accessed on December 2021.

[28] Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer Systems*, pages 68–72, 2003.

[29] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptol. ePrint Arch.*, pages 1–118, 2016.

[30] Michael Crosby, Pradan Pattanayak, Sanjeev Verma, and Vignesh Kalyanaraman. Blockchain technology: Beyond bitcoin. *Applied Innovation*, 2016.

[31] Ivan Bjerre Damgård. Practical and provably secure release of a secret and exchange of signatures. *Journal of Cryptology*, pages 201–222, 1995.

[32] Luc Dandurand and Oscar Serrano Serrano. Towards improved cyber security information sharing. In *2013 5th International Conference on Cyber Conflict (CYCON)*, pages 1–16. Tallinn, Estonia: IEEE, 2013.

[33] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 1–12, 1987.

[34] Docker. Docker swarm mode overlay network security model. `https://docs.docker.com/v17.09/engine/userguide/networking/overlay-security-model/`. Accessed on March 2022.

[35] Docker. Kubernetes. `https://www.docker.com/products/kubernetes`. Accessed on December 2021.

[36] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, pages 288–323, 1988.

[37] Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. Fairswap: How to fairly exchange digital goods. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 967–984. Toronto, Canada: ACM, 2018.

[38] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment hubs over cryptocurrencies. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 106–123. San Francisco, CA, USA: IEEE, 2019.

[39] Lisa Eckey, Sebastian Faust, and Benjamin Schlosser. Optiswap: Fast optimistic fair exchange. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, pages 543–557, 2020.

[40] ENISA. ENISA threat landscape 2020 - botnet. `https://www.enisa.europa.eu/publications/enisa-threat-landscape-2020-botnet`. Accessed on December 2021.

[41] Ethereum. Swarm. `https://swarm.ethereum.org/`. Accessed on December 2021.

[42] Patrick T Eugster, Rachid Guerraoui, A-M Kermarrec, and Laurent Massoulié. Epidemic information dissemination in distributed systems. *Computer*, pages 60–67, 2004.

[43] Bin Fan, John CS Lui, and Dah-Ming Chiu. The design trade-offs of bittorrent-like file sharing protocols. *IEEE/ACM Transactions on Networking (TON)*, pages 365–376, 2008.

[44] Michal Feldman, Kevin Lai, Ion Stoica, and John Chuang. Robust incentive techniques for peer-to-peer networks. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, pages 102–111, 2004.

[45] Md Sadek Ferdous, Mohammad Jabed Morshed Chowdhury, Mohammad A Hoque, and Alan Colman. Blockchain consensuses algorithms: A survey. *arXiv Preprint arXiv:2001.07091*, 2020.

[46] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 186–194. Springer, 1986.

[47] Filecoin. Filecoin: A decentralized storage network. `https://filecoin.io/filecoin.pdf`. Accessed on December 2021.

[48] Filecoin. Filecoin specification. `https://spec.filecoin.io/`. Accessed on December 2021.

[49] Gina Fisk, Calvin Ardi, Neale Pickett, John Heidemann, Mike Fisk, and Christos Papadopoulos. Privacy principles for sharing cyber security data. In *2015 IEEE Security and Privacy Workshops*, pages 193–197. San Jose, CA, USA: IEEE, 2015.

[50] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 281–310. Springer, 2015.

[51] Juan Garay, Philip MacKenzie, Manoj Prabhakaran, and Ke Yang. Resource fairness and composability of cryptographic protocols. In *Theory of Cryptography Conference*, pages 404–428, 2006.

[52] Daniela Gavidia, Spyros Voulgaris, and Maarten Van Steen. A gossip-based distributed news service for wireless mesh networks. In *WONS 2006: Third Annual Conference on Wireless On-demand Network Systems and Services*, pages 59–67, 2006.

[53] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, pages 270–299, 1984.

[54] Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, pages 281–308, 1988.

[55] Prateesh Goyal, Ravi Netravali, Mohammad Alizadeh, and Hari Balakrishnan. Secure incentivization for decentralized content delivery. In *the 2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge)*. USENIX Association, 2019.

[56] Mathias Gschwandtner, Lukas Demetz, Matthias Gander, and Ronald Maier. Integrating threat intelligence to enhance an organization's information security management. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, pages 1–8, 2018.

[57] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Layer-two blockchain protocols. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 201–226. Springer, 2020.

[58] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster asynchronous bft protocols. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 803–818. Virtual Event, USA: ACM, 2020.

[59] Garrett Hardin. The tragedy of the commons. *Journal of Natural Resources Policy Research*, pages 243–253, 2009.

[60] Songlin He, Eric Ficke, Mir Mehedi Ahsan Pritom, Huashan Chen, Qiang Tang, Qian Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu. Blockchain-based automated and robust cyber security management. *Journal of Parallel and Distributed Computing (JPDC)*, pages 62–82, 2022.

[61] Songlin He, Yuan Lu, Qiang Tang, Guiling Wang, and Chase Qishi Wu. Fair peer-to-peer content delivery via blockchain. In *the 26th European Symposium on Research in Computer Security (ESORICS)*, pages 348–369. Springer, 2021.

[62] Songlin He, Qiang Tang, and Chase Q Wu. Censorship resistant decentralized IoT management systems. In *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 454–459, 2018.

[63] Songlin He, Qiang Tang, Chase Q Wu, and Xuewen Shen. Decentralizing IoT management systems using blockchain for censorship resistance. *IEEE Transactions on Industrial Informatics (TII)*, pages 715–727, 2019.

[64] Udo Helmbrecht, Steve Purser, Graeme Cooper, Demosthenes Ikonomou, Louis Marinos, Evangelos Ouzounis, Marco Thorbrugge, Andreas Mitrakas, and Sarah Capogrossi. Cybersecurity cooperation: Defending the digital frontline. Technical report, ENISA, 2013.

[65] Hyperledger. The ordering service. `https://hyperledger-fabric.readthedocs.io/en/release-2.2/orderer/ordering_service.html`. Accessed on December 2021.

[66] IBM. Internet connection and recommended encoding settings. `https://support.video.ibm.com/hc/en-us/articles/207852117-Internet-connection-and-recommended-encoding-settings`. Accessed on December 2021.

[67] IBM. Introduction to hyperledger smart contracts for IoT best practices and patterns. `https://github.com/ibm-watson-iot/blockchain-samples/blob/master/docs/iotcp/HyperledgerContracts IntroBestPracticesPatterns.md`. Accessed on December 2021.

[68] Proofpoint Inc. Proofpoint emerging threats rules. `https://rules.emergingthreats.net/`. Accessed on March 2022.

[69] International Organization for Standardization. ISO/IEC 27305:2016: Information technology-security techniques-information security incident management. `https://www.iso.org/standard/60803.html`. Accessed on December 2021.

[70] Simon Janin, Kaihua Qin, Akaki Mamageishvili, and Arthur Gervais. Filebounty: Fair data exchange. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroSPW)*, pages 357–366, 2020.

[71] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten Van Steen. Gossip-based peer sampling. *ACM Transactions on Computer Systems (TOCS)*, 25(3):8, 2007.

[72] Carlee Joe-Wong, Youngbin Im, Kyuyong Shin, and Sangtae Ha. A performance analysis of incentive mechanisms for cooperative computing. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, pages 108–117. Nara, Japan: IEEE, 2016.

[73] Friedman Jon and Bouchard Mark. Definitive guide to cyber threat intelligence. Technical report, Technical Report, 2015.

[74] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. Arbitrum: Scalable, private smart contracts. In *27th USENIX Security Symposium (USENIX Security)*, pages 1353–1370, 2018.

[75] Sepandar D Kamvar, Mario T Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web (WWW)*, pages 640–651, 2003.

[76] Jiawen Kang, Rong Yu, Xumin Huang, Sabita Maharjan, Yan Zhang, and Ekram Hossain. Enabling localized peer-to-peer electricity trading among plug-in hybrid electric vehicles using consortium blockchains. *IEEE Transactions on Industrial Informatics (TII)*, pages 3154–3164, 2017.

[77] Vishal Karande, Erick Bauman, Zhiqiang Lin, and Latifur Khan. Sgx-log: Securing system logs with sgx. In *Proceedings of the Asia Conference on Computer and Communications Security (AsiaCCS)*, pages 19–30. Abu Dhabi United Arab Emirates: ACM, 2017.

[78] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2014.

[79] Anne-Marie Kermarrec and Maarten Van Steen. Gossiping in distributed systems. *ACM SIGOPS Operating Systems Review*, pages 2–7, 2007.

[80] Rami Khalil, Alexei Zamyatin, Guillaume Felley, Pedro Moreno-Sanchez, and Arthur Gervais. Commit-chains: Secure, scalable off-chain payments. *Cryptology ePrint Archive, Report 2018/642*, 2018.

[81] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 705–734. Springer, 2016.

[82] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *Self-Published Paper*, 2012.

[83] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, pages 203–209, 1987.

[84] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 839–858. San Jose, CA, USA: IEEE, 2016.

[85] Rob Van Kranenburg. The Internet of Things: A critique of ambient technology and the all-seeing network of RFID. `https://www.networkcultures.org/_uploads/notebook2_theinternetofthings.pdf`. Accessed on December 2021.

[86] Nir Kshetri. Can blockchain strengthen the internet of things? *IT Professional*, pages 68–72, 2017.

[87] Yoram Kulbak and Danny Bickson. The emule protocol specification. `http://www.jmule.org/files/emule.pdf`. Accessed on December 2021.

[88] Alptekin Küpçü and Anna Lysyanskaya. Usable optimistic fair exchange. In *Cryptographers' Track at the RSA Conference*, pages 252–267. Springer, 2010.

[89] Protocol Labs. IPFS cluster. `https://cluster.ipfs.io/documentation/guides/pinning/`. Accessed on December 2021.

[90] Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, pages 35–40, 2010.

[91] In Lee and Kyoochun Lee. The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *Business Horizons*, pages 431–440, 2015.

[92] Dave Levin, Katrina LaCurts, Neil Spring, and Bobby Bhattacharjee. Bittorrent is an auction: analyzing and improving bittorrent's incentives. *Proceedings of the ACM SIGCOMM Conference on Data Communication*, pages 243–254, 2008.

[93] Piotr Lipnicki, Daniel Lewandowski, Diego Pareschi, Waldemar Pakos, and Enrico Ragaini. Future of IoTSP–IT and OT integration. In *the 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 203–207. Barcelona: IEEE, 2018.

[94] Thomas Locher, Patrick Moore, Stefan Schmid, and Roger Wattenhofer. Free riding in bittorrent is cheap. In *HotNets*, 2006.

[95] Lucie Lozinski. How ringpop from Uber engineering helps distribute your application. `https://eng.uber.com/ringpop-open-source-nodejs-library/`. Accessed on March 2022.

[96] Yuan Lu. *Crowdsourcing Atop Blockchains*. PhD thesis, New Jersey Institute of Technology, 2020.

[97] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *Proceedings of the 39th Symposium on Principles of Distributed Computing (PODC)*, pages 129–138. Virtual Event, Italy: ACM, 2020.

[98] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. Demystifying incentives in the consensus computer. In *Proceedings of the 22nd SIGSAC Conference on Computer and Communications Security (CCS)*, pages 706–719. Denver Colorado, USA: ACM, 2015.

[99] Deepak Maram, Harjasleen Malvai, Fan Zhang, Nerla Jean-Louis, Alexander Frolov, Tyler Kell, Tyrone Lobban, Christine Moy, Ari Juels, and Andrew Miller. Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1348–1366. San Francisco, CA, USA: IEEE, 2021.

[100] Albert Marcella Jr and Doug Menendez. *Cyber forensics: a field manual for collecting, examining, and preserving evidence of computer crimes*. Auerbach Publications, 2007.

[101] Gregory Maxwell. The first successful zero-knowledge contingent payment. `https://bitcoincore.org/en/2016/02/26/zero-knowledge-contingent-payments-announcement/`. Accessed on December 2021.

[102] Muhammad Izhar Mehar, Charles Louis Shier, Alana Giambattista, Elgar Gong, Gabrielle Fletcher, Ryan Sanayhie, Henry M Kim, and Marek Laskowski. Understanding a revolutionary and flawed grand experiment in blockchain: the DAO attack. *Journal of Cases on Information Technology*, pages 19–32, 2019.

[103] Matthias Mettler. Blockchain technology in healthcare: The revolution starts here. In *2016 IEEE 18th International Conference on E-Health Networking, Applications and Services (HealthCom)*, pages 1–3. IEEE, 2016.

[104] Silvio Micali. Simple and fast optimistic protocols for fair electronic exchange. In *Proceedings of the 22nd Annual Symposium on Principles of Distributed Computing (PODC)*, pages 12–19, 2003.

[105] Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. Sprites and state channels: Payment networks that go faster than lightning. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 508–526. Springer, 2019.

[106] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing bitcoin work for data preservation. In *the Symposium on Security and Privacy (SP)*, pages 475–490. Berkeley, CA, USA: IEEE, 2014.

[107] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the SIGSAC Conference on Computer and Communications Security (CCS)*, pages 31–42. Vienna, Austria: ACM, 2016.

[108] Frederic P Miller, Agnes F Vandome, and John McBrewster. Levenshtein distance: Information theory, computer science, string (computer science), string metric, damerau? levenshtein distance, spell checker, hamming distance, 2009.

[109] Srijith K Nair, Erik Zentveld, Bruno Crispo, and Andrew S Tanenbaum. Floodgate: A micropayment incentivized p2p content delivery network. In *Proceedings of 17th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–7, 2008.

[110] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `http://www.bitcoin.org/bitcoin.pdf`. Accessed on December 2021.

[111] Satoshi Nakamoto. Bitcoin P2P e-cash paper. `https://satoshi.nakamotoinstitute.org/emails/cryptography/1/`. Accessed on December 2021.

[112] Till Neudecker and Hannes Hartenstein. Network layer aspects of permissionless blockchains. *IEEE Communications Surveys and Tutorials*, pages 838–857, 2018.

[113] Huansheng Ning, Hang Wang, Yujia Lin, Wenxi Wang, Sahraoui Dhelim, Fadi Farha, Jianguo Ding, and Mahmoud Daneshmand. A survey on metaverse: the state-of-the-art, technologies, applications, and challenges. *arXiv Preprint arXiv:2111.09673*, 2021.

[114] Oscar Novo. Blockchain meets IoT: An architecture for scalable access management in IoT. *IEEE Internet of Things Journal*, pages 1184–1195, 2018.

[115] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC)*, pages 305–319, 2014.

[116] Henning Pagnia and Felix C Gärtner. On the impossibility of fair exchange without a trusted third party. Technical report, Technical Report TUD-BS-1999-02, Darmstadt University of Technology, 1999.

[117] Alfonso Panarello, Nachiket Tapas, Giovanni Merlino, Francesco Longo, and Antonio Puliafito. Blockchain and IoT integration: A systematic survey. *Sensors*, page 2575, 2018.

[118] W. Michael Petullo, Ben Klimkowski, William Clay Moody, Joshua Bundt, and Michael Kranch. Cyber defense exercise artifacts. `https://www.flyn.org/CDX/`. Accessed on December 2021.

[119] Michael Piatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do incentives build robustness in bittorrent. In *Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2007.

[120] Benny Pinkas. Fair secure two-party computation. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 87–105. Springer, 2003.

[121] Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. *White Paper*, pages 1–47, 2017.

[122] Christian Reitwiessner. EIP-196: Precompiled contracts for addition and scalar multiplication on the elliptic curve alt_bn128. `https://eips.ethereum.org/EIPS/eip-196`. Accessed on December 2021.

[123] Sara Saberi, Mahtab Kouhizadeh, Joseph Sarkis, and Lejia Shen. Blockchain technology and its relationships to sustainable supply chain management. *International Journal of Production Research*, pages 2117–2135, 2019.

[124] Michael N. Schmitt. *Tallinn Manual 2.0 on the International Law Applicable to Cyber Operations*. Cambridge University Press, 2017.

[125] Claus Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology (CRYPTO)*, pages 239–252. Springer, 1989.

[126] Oscar Serrano, Luc Dandurand, and Sarah Brown. On the design of a cyber security data sharing system. In *Proceedings of the Workshop on Information Sharing and Collaborative Security*. Scottsdale, Arizona, USA: ACM, 2014.

[127] Giuseppe Settanni, Yegor Shovgenya, Florian Skopik, Roman Graf, Markus Wurzenberger, and Roman Fiedler. Acquiring cyber threat intelligence through security information correlation. In *the 3rd International Conference on Cybernetics (CYBCONF)*, pages 1–7. IEEE, 2017.

[128] Adi Shamir. How to share a secret. *Communications of the ACM*, pages 612–613, 1979.

[129] Ankur Sharma, Felix Martin Schuhknecht, Divya Agrawal, and Jens Dittrich. Blurring the lines between blockchains and database systems: the case of hyperledger fabric. In *Proceedings of International Conference on Management of Data (SIGMOD)*, pages 105–122, 2019.

[130] Alex Sherman, Jason Nieh, and Clifford Stein. Fairtorrent: a deficit-based distributed algorithm to ensure fairness in peer-to-peer systems. *IEEE/ACM Transactions on Networking (TON)*, pages 1361–1374, 2012.

[131] Kyuyong Shin, Carlee Joe-Wong, Sangtae Ha, Yung Yi, Injong Rhee, and Douglas S Reeves. T-chain: A general incentive scheme for cooperative computing. *IEEE/ACM Transactions on Networking (TON)*, pages 2122–2137, 2017.

[132] Michael Sirivianos, Jong Han Park, Xiaowei Yang, and Stanislaw Jarecki. Dandelion: Cooperative content distribution with robust incentives. In *USENIX Annual Technical Conference (USENIX ATC)*, 2007.

[133] Florian Skopik, Giuseppe Settanni, and Roman Fiedler. A problem shared is a problem halved: A survey on the dimensions of collective cyber defense through security information sharing. *Computers and Security*, pages 154–176, 2016.

[134] Joao Sousa and Alysson Bessani. From Byzantine consensus to BFT state machine replication: A latency-optimal transformation. In *the 9th European Dependable Computing Conference (EDCC)*, pages 37–48. Sibiu, Romania: IEEE, 2012.

[135] Joao Sousa, Alysson Bessani, and Marko Vukolic. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. In *the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 51–58. Luxembourg, Luxembourg: IEEE, 2018.

[136] Statista. Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025. `https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/`. Accessed on December 2021.

[137] StorJ. StorJ: a decentralized cloud storage network framework. `https://storj.io/storj.pdf`. Accessed on December 2021.

[138] Suricata. Suricata — open source IDS / IPS / NSM engine. `https://suricata-ids.org/download/`. Accessed on December 2021.

[139] Georgia Tech. Gt malware http daily feed 2018 dataset. `https://www.impactcybertrust.org/dataset_view?idDataset=836`. Accessed on December 2021.

[140] Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. *arXiv Preprint arXiv:1908.04756*, 2019.

[141] Parth Thakkar, Senthil Nathan, and Balaji Viswanathan. Performance benchmarking and optimizing hyperledger fabric blockchain platform. In *the 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 264–276. Milwaukee, WI, USA: IEEE, 2018.

[142] Qin Wang, Rujia Li, Qi Wang, and Shiping Chen. Non-fungible token (nft): Overview, evaluation, opportunities and challenges. *arXiv Preprint arXiv:2105.07447*, 2021.

[143] Wenbo Wang, Dusit Niyato, Ping Wang, and Amir Leshem. Decentralized caching for content delivery based on blockchain: A game theoretic perspective. In *International Conference on Communications (ICC)*, pages 1–6. Kansas City, MO, USA: IEEE, 2018.

[144] WARC. Global online content conssumption doubles in wake of COVID. `https://www.warc.com/newsandopinion/news/global-online-content-consumption-doubles-in-wake-of-covid/44130`. Accessed on December 2021.

[145] Sam M Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J Knottenbelt. Sok: Decentralized Finance (Defi). *arXiv Preprint arXiv:2101.08778*, 2021.

[146] Wikipedia. Flint water crisis. `https://en.wikipedia.org/wiki/Flint_water_crisis`. Accessed on December 2021.

[147] Wikipedia. Levenshtein distance. `https://en.wikipedia.org/wiki/Levenshtein_distance`. Accessed on December 2021.

[148] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, pages 1–32, 2014.

[149] Yang Xiao, Ning Zhang, Wenjing Lou, and Y Thomas Hou. A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys and Tutorials*, pages 1432–1465, 2020.

[150] Shouhuai Xu and Moti Yung. Expecting the unexpected: Towards robust credential infrastructure. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 201–221. Springer, 2009.

[151] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In *Proceedings of the SIGSAC Conference on Computer and Communications Security (CCS)*, pages 270–282. Vienna Austria: ACM, 2016.

[152] Fan Zhang, Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. Deco: Liberating web data using decentralized oracles for TLS. In *Proceedings of the SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1919–1938. Virtual Event, USA: ACM, 2020.

[153] Qian Zhu, Ruicong Wang, Qi Chen, Yan Liu, and Weijun Qin. IoT gateway: Bridgingwireless sensor networks into internet of things. In *the IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, pages 347–352. Hong Kong, China: IEEE, 2010.