Secure and Usable Decentralized Storage

TIANYI ZHANG

Supervisor: Prof. Qiang Tang
Associate Supervisor: Prof. Alan Fekete

A thesis submitted in fulfilment of
the requirements for the degree of
Master of Philosophy

School of Computer Science
Faculty of Engineering
The University of Sydney
Australia

15 October 2025

Abstract

Recent advancements in the Dynamic-Committee Proactive Secret Sharing (DPSS) protocol have introduced mechanisms for periodically refreshing shares and dynamically changing the parties holding them. DPSS is particularly relevant in modern applications such as blockchain systems, where committees are responsible for managing confidential information, and decentralised storage networks, where participant involvement is inherently dynamic. Despite its potential, DPSS faces significant challenges, particularly its high communication complexity of $\mathcal{O}(n^3\ell + \lambda n^3)$ when refreshing data, which becomes a bottleneck when dealing with large-scale data.

In this work, we address these challenges by introducing the concept of Dynamic-Committee Proactive Information Dispersal (DPID), a novel approach that can be viewed as a relaxed version of DPSS without the confidentiality requirement. We provide a formal model for DPID and construct practical schemes that achieve substantially reduced communication complexity to $\mathcal{O}(\ell + \lambda n^2)$. To bridge the gap between DPID and DPSS, we also present a general framework for compiling DPID into DPSS, preserving efficiency while reintroducing confidentiality. By integrating our DPID construction, we achieve the first DPSS scheme that is optimized for large-scale data, offering dramatically lower communication overhead. We validate the efficiency and scalability of our approach through comprehensive experiments, which demonstrate clear improvements over state-of-the-art methods. Our work not only advances the theoretical understanding of proactive information dispersal but also provides a practical foundation for deploying DPSS in real-world systems.

Statement of Originality

This is to certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes.

I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

Name: Tianyi Zhang

Signature:

Date: 30 June 2025

List of Publications

1. Zhenliang Lu, **Tianyi Zhang**, Alan Fekete, Kwok Yan Lam, Qiang Tang. Asynchronous Dynamic Committee Proactive Secret Sharing for Large Data. *45th IEEE International Conference on Distributed Computing Systems* 2025, **ICDCS 2025**.

Authorship Attribution Statement

The contents of this thesis are based on the following published paper. I was one of the main contributor in this paper.

1. Zhenliang Lu, **Tianyi Zhang**, Alan Fekete, Kwok Yan Lam, Qiang Tang. Asynchronous Dynamic Committee Proactive Secret Sharing for Large Data. *45th IEEE International Conference on Distributed Computing Systems* 2025, **ICDCS 2025**.

My particular contributions to this have been:

• Chapter 4: I implemented the algorithm in code, designed the experiment, analysed the experiment results and analysed its security.

• Chapter 5: I designed the DPID1 refresh algorithm to utilise the honest majority subcommittee and analysed its security.

• Chapter 6: I designed the DPID2 refresh algorithm to utilise the Weak Set Cover protocol, implemented the algorithm in code, designed the experiment, analysed the experiment results and analysed its security.

Name: Tianyi Zhang
Signature:
Date:

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

Name: Qiang Tang
Signature:
Date:

Acknowledgements

I would like to express my sincere gratitude to everyone who has supported and guided me throughout the journey of completing this master's thesis.

First and foremost, I extend my deepest thanks to my advisor, Prof. Qiang Tang and Dr. Zhenliang Lu, for their invaluable guidance, insightful feedback, and unwavering encouragement. Their expertise and dedication have been instrumental in shaping this research and in helping me navigate the challenges along the way.

Special thanks are due to my colleagues and friends in the USYD Crypto Lab for their continuous support. I would like to say thank you to Tian Qiu, Ya-Nan Li, Dr.Hanwen Feng, Xinrui Zhang, Tiancheng Mai, Yuchen Ye, Quanhao Chen, Sam Polgar and Prof.Alan Fekete.

Finally, I wish to express my heartfelt appreciation to my parents, grandparents, little brother and Qinqin Cai for their unconditional love, patience, and encouragement.

Success is not final, failure is not fatal, it is the courage to continue that counts. Thank you all for your contributions to this journey.

I gratefully acknowledge the support of generative AI tools, which have been invaluable for ideation, troubleshooting, and the generation and testing of code. This research was supported by an Australian Government Research Training Program (RTP) Scholarship for my tuition fees.

CONTENTS

List of Figures

CHAPTER 1

Introduction

1.1 Evolution and Challenge of Data Storage

History of Data Storage. The evolution of data storage reflects humanity's continuous effort to preserve and access information efficiently. From ancient clay tablets to modern digital systems, each technological advancement has transformed our ability to manage and transmit information. Early civilisations relied on physical media such as stone carvings, clay tablets and papyrus scrolls. While these physical media are revolutionary for their time, they were heavily limited by their susceptibility to environmental damage and the labour-intensive nature of reproduction. Later, the invention of paper, which can be used as a durable and scalable medium for recording information, marked a pivotal moment in ancient China. This innovation spread across Asia and Europe, culminating in Johannes Gutenberg's printing press in the 15th century, which democratised access to knowledge by enabling the mass production of books and documents. These developments laid the solid foundation for the modern era of information storage, setting the stage for the digital revolution.

The 20th century witnessed the advent of electronic data storage, a significant leap that redefined how information could be stored, accessed and manipulated with superb usability. Magnetic tape storage, introduced in

the 1950s, provided a more reliable and efficient alternative to the traditional paper-based storage systems. Over the following decades, data storage technology evolved rapidly, transitioning from magnetic tapes to floppy disks and eventually to hard drives. Each iteration offered greater capacity, durability and accessibility to the data storage. These advancements in storage coincided with the rise of personal computing in the 1980s, bringing powerful data storage capabilities to individuals and small businesses. The ability to easily store and retrieve digital information revolutionised all industries, enabling new forms of communication and collaboration. However, the decisive change came in the early 21st century with the emergence of cloud computing and centralised storage solutions. [**4**, **16**, **39**]

1.1.1 From Centralised Storage to Decentralised Storage

Centralised Cloud Storage. Cloud storage, pioneered by platforms such as Amazon Web Services [**39**], Microsoft Azure [**4**] and Google Cloud [**16**], fundamentally altered the landscape of data storage and management. By offering virtually unlimited storage capacity on demand, cloud storage eliminates the need for costly on-premise physical infrastructure, such as data centres, with unprecedented flexibility. Organisations could now scale their storage resources dynamically, adapting to fluctuating demands without the constraints of traditional hardware storage. This shift not only reduced operational costs but also enabled businesses to focus more on the actual business logic, rather than setting up the data storage infrastructure from scratch. The global accessibility of cloud storage further redefined team collaboration, facilitating seamless knowledge sharing across geographic boundaries. Whether it is a multinational corporation coordinating across continents or a remote team working on a shared project, cloud storage has become the backbone of modern team collaboration, breaking down barriers of time and space.

Moreover, the pay-as-you-go model democratised access to enterprise-level storage, allowing individuals, small businesses and startups to compete on the same playing field with larger corporations. This economic model lowered the barrier to entry for emerging companies, fostering real innovation and levelling the competitive landscape. The distributed architecture of cloud systems, with data replicated across multiple geographic locations, enhances both availability and fault-tolerant capabilities. By ensuring proper redundancy and fail-over mechanisms, cloud providers minimise the risk of data loss due to hardware failures, natural disasters or any other disruptions. Such reliability has rendered cloud storage indispensable for both businesses and individuals, providing assurance in an increasingly data-driven world.

Security Concerns. However, the rise of cloud storage also brought security issues to the forefront of public consciousness like never before. As organisations began migrating their sensitive business data, such as customer data, to the cloud, concerns about potential data breaches, unauthorised access and cyberattacks emerged. For the first time, security was no longer a concern limited to IT departments or specialised industries only. Instead, it became a critical issue affecting businesses, governments and individuals.

Recognising this, cloud storage providers have significantly invested in proper security measures to tackle these challenges. For instance, Amazon Web Services [**39**] introduced robust encryption protocols for data at rest and in transit, ensuring that sensitive information remains protected from unauthorised access. Microsoft Azure [**4**] implemented advanced identity and access management systems like Managed Identity, allowing organisations to regulate access to their data by assigning distinct permission tiers. Google Cloud [**16**], meanwhile, integrated machine learning-driven threat detection systems to identify and mitigate potential security risks in real-time.

These efforts by cloud providers have not only improved the security of their platforms and customer data but also raised the bar for the entire network security industry. Modern security methods such as multi-factor authentication, end-to-end encryption and zero-trust security models have become standard security features, reflecting the growing importance of security in the design and operation of cloud data storage systems. Furthermore, compliance with international security standards, such as GDPR [**34**] and ISO 27001 [**21**], has become a key selling point for cloud providers, as organisations increasingly prioritise regulatory compliance alongside data protection.

Despite the significant efforts made by cloud storage providers to enhance security and usability, the centralised nature of these systems continues to present critical challenges. One of the most concerning issues is the vulnerability of service outages. When a centralised cloud service experiences downtime due to technical failures, cyberattacks or natural disasters, customer data becomes unavailable to the users. High-profile incidents, such as the 2017 Amazon S3 outage [**38**] that disrupted countless websites and services, have highlighted the risks of a single point of failure. For businesses, such disruptions can result in significant financial losses, operational delays and damage to their reputation. For individual users, the consequence may be losing access to critical personal files, photos and confidential secrets at crucial moments. This dependency on centralised systems undermines the data availability that cloud storage promises as a key feature, exposing users to unforeseen risks.

Another pressing concern is the lack of transparency and control over data privacy. While cloud providers assert that they employ encryption and security protocols, users have no choice but to ultimately trust these companies to safeguard their data. However, this trust is not always well-placed. Centralised storage systems inherently grant providers full access to users' data,

whether it is for maintenance, data analytics, or any other purposes. This raises the unsettling possibility that sensitive data could be accessed, analysed, or even exploited without users' knowledge or consent. For instance, in 2019, it was revealed that a major cloud provider had allowed employees to review audio recordings from voice assistant devices [**41**], intentionally to improve service quality. Such incidents have fuelled scepticism about the potential privacy breaches of data stored in centralised systems, particularly as cloud providers increasingly monetise user data through AI learning, targeted advertising and other business models.

Moreover, the centralised architecture of cloud storage creates a single point of control, which can be exploited for censorship or surveillance. Governments and other entities can pressure cloud providers to hand over sensitive user data or restrict access to certain information, often without the knowledge or consent of the affected users. This centralisation of power contradicts the principles of data sovereignty and user privacy, raising ethical and legal questions about who truly owns and controls digital data. For businesses operating in highly regulated industries or regions with strict data privacy laws, this lack of control can pose significant compliance and financial risks.

Decentralised Storage. These limitations have, in turn, spurred growing interest in decentralised storage solutions, which aim to address the shortcomings of centralised systems while maintaining the security and usability that users expect from storage systems. Decentralised storage systems are built on a distributed architecture that fundamentally redefines how digital data is stored, managed and accessed.

Unlike traditional centralised storage models, which rely on a single or a few servers, decentralised storage leverages a large network of nodes to distribute data across multiple locations. This architecture inherently eliminates single points of failure, enhancing both reliability and fault tolerance

of the storage system. Key technologies underpinning decentralised storage include distributed hash tables, peer-to-peer networks and modern cryptographic protocols.

Secret Sharing. As a cryptographic technique, secret sharing plays a crucial role in enhancing data security and redundancy in decentralised systems. By splitting data into multiple shares and distributing them across different nodes, secret sharing ensures that no single node possesses complete data anymore, safeguarding data confidentiality and integrity. This approach also allows for data recovery even if some shares are lost or compromised, further strengthening the resilience of storage systems. For instance, systems like InterPlanetary File System [**28**] and Filecoin [**29**] utilise distributed hash tables to efficiently locate and retrieve data across a decentralised network, while Storj [**30**] employs peer-to-peer protocols to ensure seamless data transfer between nodes. These new technologies, combined with secret sharing, collectively enable decentralised storage systems to achieve high data availability, scalability and resilience against unforeseen network disruptions. The integration of proper secret sharing not only enhances security but also contributes to the overall robustness of the storage infrastructure, rendering it well-suited for applications requiring high levels of data protection and fault tolerance.

1.1.2 Secure Decentralised Storage in Asynchronous Networks with Dynamic Nodes

Decentralised storage systems are increasingly deployed in asynchronous networks where participating nodes may frequently join or leave. Ensuring secure and efficient storage under such conditions is critical for applications ranging from blockchain-based storage networks to distributed cloud infrastructures. However, current approaches struggle to provide both confidentiality and scalability under these settings. In particular, existing distributed

secret sharing protocols incur high communication costs, which become prohibitive for large data inputs or frequent reconfigurations. This section explores the design of a secure, decentralised storage protocol that accommodates dynamic node participation in asynchronous networks, while reducing communication complexity and maintaining strong confidentiality guarantees.

Asynchronous Network. In decentralised storage systems, asynchronous network communication is a fundamental characteristic that allows nodes to operate independently without requiring real-time coordination. Unlike synchronous networks, where nodes must adhere to a strict timing schedule, asynchronous networks tolerate variable message delivery times and node responsiveness. This flexibility is particularly advantageous in decentralised environments, where nodes may experience varying levels of latency, bandwidth, or even temporary disconnections. Asynchronous communication ensures that the system remains operational and can continue processing data requests even under suboptimal network conditions [**33**]. This property is critical for maintaining the availability, robustness and scalability of decentralised storage systems, as it allows them to function effectively across geographically distributed networks.

Dynamic Nodes. As decentralised storage systems gain traction, especially in distributed cloud services, the demand for dynamic and secure data management becomes increasingly critical. In distributed cloud services, some cloud service providers can go offline or close down and for others to join. Thus, we hope the cloud service providers and their storage nodes can change on a regular basis due to practical concerns. For example, in a decentralised network, nodes may join or leave the system unpredictably, requiring mechanisms to redistribute data shares dynamically. However, cryptographic keys used to secure data may become vulnerable over time due to advances in computing power or cryptographic attacks. Moreover, the

adversary may progressively compromise an increasing number of nodes. These challenges highlight the need for innovative approaches that combine the robustness of decentralised storage with advanced security and dynamic management capabilities.

In this dissertation, we consider a dynamic committee with node changes. In particular, we formalise the lifetime of the system as a sequence of discrete epochs, and for every epoch e, a group of selected nodes (called the committee) will store the data fragment. The participating nodes of different epochs could be different to support a dynamic committee. Besides, the adversary can corrupt different nodes in different epochs, referred to as a mobile adversary setting. Although the adversary may eventually compromise every node over the system's lifetime, it cannot control more than f committee members in any single epoch, where f is the maximum number of nodes that may become faulty during execution.

Dynamic-Committee Proactive Secret Sharing. One important primitive that supports both asynchronous networks and dynamic nodes is called Dynamic-Committee Proactive Secret Sharing (DPSS), which is a variant of secret sharing. DPSS has been widely studied and deployed in distributed systems and blockchain applications. [**11**, **36**, **45**–**47**].

In a DPSS, there is a dealer or data owner, who first distributes shares of a secret message S to a node committee, which consists of a set of n nodes. And at most f of these nodes, where $n \geq 3f + 1$, can be Byzantine nodes. Byzantine nodes here are nodes that may deviate from the protocol in arbitrary or malicious ways by sending conflicting or tampered messages. There is also a reconstruction procedure, in which anyone, sometimes referred to as a client, contacts nodes to reconstruct the original secret S.

As we expect a dynamic environment where the nodes that store shares may be periodically changed, DPSS goes beyond traditional secret-sharing

and has an extra complex Refresh phase, sometimes called Handoff phase, which occurs when nodes of epoch $e-1$ transfer their shares to nodes in a different committee for epoch e. Furthermore, for technical reasons, DPSS also has a stronger termination requirement that if some honest node terminates with a valid share, then all honest nodes can also terminate with some valid share.

Typical applications of DPSS include distributed systems or fault-tolerant consensus reconfiguration, where secrets are periodically reshared to accommodate changes in the committee. A prime application lies in the distributed storage system, which has been a standard research topic, usually focused on in-house data centres. With the rise of blockchain technology, however, these applications have extended to open and permissionless environments. In particular, multiple real-world decentralised storage networks such as Filecoin [**29**] have emerged. In such systems, storage nodes are dynamic and may continuously change, making DPSS indispensable for ensuring secure reconfiguration and long-term reliability.

1.1.3 Efficiency Bottleneck in DPSS

In this dissertation, we denote ℓ as the size of the secret, λ as a computational security parameter, e.g., the size of a digital signature, and $\mathcal{O}(n)$ as the size of every committee. All existing DPSS have at least a cubic cost in the communication complexity ($\mathcal{O}(n^3\ell + n^3\lambda)$) and quadratic storage complexity ($\mathcal{O}(n^2\ell + n^2\lambda)$) in the Refresh phase. See Table.1.1.

Clearly, the total size of the distributed storage is substantial, and when secrets must be re-distributed, often in batches, the storage requirements grow even larger. Consequently, the $n^3\ell$ term in the communication complexity becomes prohibitive for any moderately sized network, especially since the Refresh phase may need to be executed repeatedly. Such overhead renders

TABLE 1.1. Comparison for performance metrics of DPSS Refresh

Protocol	Network	Tolerance	Dynamic	Communication Complexity		Trusted Setup
				Optimistic*	Worst	
Herzberg et al. [26]	Sync	$n/2$	No	$\mathcal{O}(n^3\ell + \lambda n^3)$	$\mathcal{O}(n^3\ell + \lambda n^3)$	No
Desmedt et al. [19]	Sync	$n/2$	Yes	$\mathcal{O}(n^2\ell + \lambda n^2)$	–§	No
Wong et al. [44]	Sync	$n/2$	Yes	$\mathcal{O}(\exp(n\ell + \lambda n))$	$\mathcal{O}(\exp(n\ell + \lambda n))$	No
Baron et al. [5]	Sync	$(1/2-\varepsilon)n$	Yes	$\mathcal{O}(n^3\ell + \lambda n^3)$	$\mathcal{O}(n^3\ell + \lambda n^3)$	No
CHURP [32]	Sync	$n/2$	Yes	$\mathcal{O}(n^2\ell + \lambda n^2)$	$\mathcal{O}(n^3\ell + \lambda n^3)$	Yes
Goyal et al. [22]	Sync	$n/2$	Yes	$\mathcal{O}(n^2\ell + \lambda n^2)$	$\mathcal{O}(n^3\ell + \lambda n^3)$	Yes
Schultz-MPSS [37]	P-Sync	$n/3$	Yes	$\mathcal{O}(n^4\ell + \lambda n^4)$	$\mathcal{O}(n^4\ell + \lambda n^4)$	No
COBRA [43]	P-Sync	$n/3$	Yes	$\mathcal{O}(n^3\ell + \lambda n^3)$	$\mathcal{O}(n^4\ell + \lambda n^4)$	No
Cachin et al. [11]	Async	$n/3$	No	$\mathcal{O}(n^4\ell + \lambda n^4)$	$\mathcal{O}(n^4\ell + \lambda n^4)$	Yes
Zhou et al. [47]	Async	$n/3$	Yes	$\mathcal{O}(\exp(n\ell + \lambda n))$	$\mathcal{O}(\exp(n\ell + \lambda n))$	No
Shanrang [45]	Async	$n/4$	Yes	$\mathcal{O}(n^3 \log n\ell)$	$\mathcal{O}(n^4\ell)$	Yes
Y-VSS [36]	Async	$n/2$	Yes	$\mathcal{O}(n^4\ell + \lambda n^4)$	$\mathcal{O}(n^4\ell + \lambda n^4)$	No
Long Live [46]†	Async	$n/3$	Yes	$\mathcal{O}(n^3\ell + \lambda n^3)$	$\mathcal{O}(n^3\ell + \lambda n^3)$	No
DPID0+ [46]	Async	$n/3$	Yes	$\mathcal{O}(n\ell + \lambda n^3)$	$\mathcal{O}(n\ell + \lambda n^3)$	No
DPID1+ [46]	Async	$n/3$	Yes	$\mathcal{O}(\kappa\ell + \lambda n^3)$	$\mathcal{O}(\kappa\ell + \lambda n^3)$	No
DPID2+ [46]	Async	$n/3$	Yes	$\mathcal{O}(\ell + \lambda n^3)$	$\mathcal{O}(\kappa\ell + \lambda n^3)$	No

*Optimistic case means that all nodes are honest, and all messages will not be delayed.
§Desmedt et al. [19] doesn't support Byzantine setting. † the state-of-the-art DPSS.

confidential fault-tolerant storage with a dynamic committee impractical, due to its excessive communication and storage costs.

Thus, reducing DPSS's communication and storage overhead while preserving its strong confidentiality and availability guarantees is of great importance. In this dissertation, we explore integrating the Asynchronous Verifiable Information Dispersal (AVID) protocol into the DPSS. AVID can replace the costly Refresh phase with an erasure-coded, proof-driven pipeline that naturally fits fully asynchronous, dynamic-committee environments.

Asynchronous Verifiable Information Dispersal. AVID is a decentralised storage primitive that uses erasure coding and non-interactive cryptographic proofs to split a large input into compact fragments. Each committee member can verify the integrity of its fragment without synchronous rounds, and the original data can be reconstructed from any subset of sufficiently many fragments, even if some nodes are Byzantine. In contrast to existing DPSS schemes that rely on computationally intensive cryptographic primitives, AVID incurs minimal processing overhead and achieves significantly lower communication complexity.

However, AVID by itself does not provide *secrecy* property, as an adversary who collects enough fragments can recover the underlying data. By contrast, DPSS offers proactive confidentiality over the secret message. As long as an adversary corrupts at most f nodes in each epoch, they learn nothing about the secret throughout its lifetime.

The principal challenge is to integrate AVID into DPSS's Refresh phase in a way that preserves DPSS's secrecy guarantees and closes the confidentiality gap, especially under the same fully asynchronous dynamic-committee, Byzantine setting.

Hence, a natural question we will be tackling in this dissertation is:

Can we leverage AVID to reduce the communication complexity of DPSS for large data inputs, while upholding its confidentiality guarantees?

1.2 Our Contributions

We answer the aforementioned question affirmatively. For the first time, we propose three Asynchronous Dynamic-Committee Proactive Information Dispersal (DPID) protocols, denoted as DPID0, DPID1 and DPID2, respectively.

In comparison to DPSS, the DPID protocols lack the *secrecy* property. To overcome this limitation and uphold confidentiality guarantees, we adopt the *hybrid encryption* paradigm. For large data objects, the dealer first encrypts the plaintext with a symmetric key. The symmetric key is then dispersed using DPSS, while the resulting ciphertext is distributed via DPID.

We now provide a more detailed explanation of our approach.

Better DPSS Constructions. We propose three new constructions of DPSS, with communication complexities of $\mathcal{O}(n\ell + n^3\lambda)$, $\mathcal{O}(k\ell + n^3\lambda)$ and $\mathcal{O}(\ell +$

$n^3\lambda)$ in its optimistic case, respectively. When ℓ is moderately large, the dominating term in the complexity is ℓ. Consequently, our new constructions significantly enhance the efficiency of DPSS. We summarise all existing DPSS constructions, including ours, in Table 1.1.

New Techniques for Constructing Better DPSS. As a major technical building block, we formulate a new primitive called Dynamic Committee Information Dispersal (DPID), the information dispersal analog with dynamic committee and asynchronous network, which is basically DPSS without *secrecy* property. Meanwhile, in information dispersal, any two clients can always recover the same data. However, in DPID, we need an additional property compared with information dispersal, that is, the recovered data is also the same for any subsequent epochs. We also present efficient constructions that satisfy this property.

Firstly, we present DPID0 based on ACVID. As a basic version of DPID, which is extremely simple, it doesn't need any trusted setup or consistent views, such as common coin or consensus. We note that DPSS at least needs a common coin, binary Byzantine agreement (ABA) or multi-valued validated Byzantine agreement (MVBA). We consider the witness of auxiliary information to include signatures of the old committee member, whose size equals the auxiliary information as $\mathcal{O}(n\lambda)$. Hence, the size of the message from an old committee member is $\mathcal{O}(\ell/f + n\lambda)$, and the communication complexity of DPID0 is $\mathcal{O}(n\ell + n^2\lambda)$. For simplicity, when discussing the communication complexity of DPID, we only consider the cost during the Refresh phase. Jumping ahead, this does not influence the final complexity of DPSS. Besides, we noted that for "yoso" security [**36**], where the old committee member sends only one message during Refresh, our DPID0 does not need a trusted setup and any consensus, at the same time satisfies "yoso" security.

We also have an easy improvement DPID1 built on DPID0. Essentially, we select a subcommittee from the nodes in the new committee, ensuring that the majority of its members are honest with high probability. We choose an honest majority subcommittee with size κ from the new committee with high probability, in this case, the communication complexity of DPID1 will then be proportional to n when $\ell > n^2\lambda$, and the communication complexity of DPID1 is $\mathcal{O}(\kappa\ell + \kappa\lambda n^2)$. As the introduction elaborates, the term related to data size is the dominant cost for a storage system.

Moreover, to minimise communication overhead, we design DPID2, where all committee members in epoch e are assumed to be honest. By replacing the subcommittee in DPID1 with a single node, the refresh protocol would now rely on a single node instead of κ nodes to reduce the communication complexity to $\mathcal{O}(\ell + \lambda n^2)$ by a factor of κ. After the dispersal phase in the first epoch, all fragments are fixed, ensuring that the fragments and auxiliary information in each subsequent epoch remain the same and originate from the same input. In DPID2, the optimistic communication complexity is constant when $\ell > \lambda n^3$. In an optimistic epoch, where all nodes are honest and the network operates synchronously, DPID2 achieves a fast path with communication complexity of only $\mathcal{O}(\ell + \lambda n^2)$. Even in the worst case, the protocol maintains security, with communication complexity $\mathcal{O}(\kappa\ell + \kappa n^2\lambda)$ when reverting back to DPID1. It effectively reduces the prior $O(n^3\ell)$ overhead to a quadratic dependency on n itself.

The performance of our DPID protocols is listed in Table 1.2. As we briefly elaborated above, the DPID is the core building block of DPSS with large data input. As shown in Table 1.1, the communication complexity has a huge improvement over the state-of-the-art. Lastly, special care is needed for constructing DPID, including avoiding the computationally heavy generic NIZKs; and subtleties exist when compiling DPID to a DPSS. See discussions below.

TABLE 1.2. Performance metrics of DPID

Protocol	Communciation Complexity		Trusted Setup
	Optimistic*	Worst	
DPID0	$\mathcal{O}(n\ell + \lambda n^2)$	$\mathcal{O}(n\ell + \lambda n^2)$	No
DPID1	$\mathcal{O}(\kappa\ell + \kappa\lambda n^2)$	$\mathcal{O}(\kappa\ell + \kappa\lambda n^2)$	No
DPID2	$\mathcal{O}(\ell + \lambda n^2)$	$\mathcal{O}(\kappa\ell + \kappa\lambda n^2)$	No

*Optimistic case means all nodes are honest and all messages have no delay

Implementation and Extensive Evaluations. We implemented our DPID and DPSS protocols and carried out extensive evaluations, and compared them with the state-of-the-art DPSS protocol LongLive DPSS [**46**]. Both our DPSS0 and DPSS2 clearly outperform LongLive DPSS consistently.

1.3 Technical Overview

As briefly mentioned earlier, the most critical component of our work is DPID. This protocol not only enables our final DPSS constructions but may also be directly applicable in distributed storage settings, where users may be willing to trade privacy for efficiency when dealing with a large volume of non-sensitive data. However, even for DPID, the refresh phase in the asynchronous setting introduces significant complexity.

DPID represents the first asynchronous dynamic committee information dispersal protocol. While DPSS can serve a similar purpose, DPID distinguishes itself by handling several technical challenges differently. To draw an analogy, DPID is somewhat akin to AVID and AVSS. However, if we temporarily set aside the requirement for *secrecy*, our goal is to achieve low communication complexity, in contrast to the higher complexity associated with DPSS. Below, we provide further details on this distinction and the technical innovations behind DPID. Additionally, we elaborate on the challenges and the techniques we employ to address them.

Information Dispersal. Decentralised storage systems have been a classical topic and recently attracted renewed attention due to the popularity

of blockchain. The model of this problem is that the dealer stores some data among a set of n nodes, which can be later recovered by anyone, often called a client, by contacting a sufficient number of nodes, where at most f nodes are Byzantine [**3**]. An adversary can control these Byzantine nodes during the epoch e. The basic requirement is that if any two honest clients try to recover, they can always recover the same data. Additionally, if the dealer is honest, then the recovered data must be the dealer's original input data.

A straightforward solution would involve each node storing the entire dataset, enabling the recovery dealer to output the data upon receiving identical contributions from $n - f$ nodes. However, this approach is clearly impractical, as nodes do not need to store the entire dataset, especially in a blockchain context where storage efficiency is critical, especially for large input data.

As a result, information dispersal is an innovative solution that considers each node to keep only a fragment of the original data, such that a threshold node's fragment can reconstruct the original data. Rabin [**35**] proposed the earliest concept using an information dispersal algorithm (IDA). Specifically, information dispersal consists of two phases: dispersal and retrieval. The basic idea of IDA is to split the data into fragments, and each node keeps only one fragment in the dispersal phase, so that a subset of the fragments can recover the previously dispersed data in the retrieval phase. However, IDA cannot handle Byzantine nodes in the recast phase. Some later work [**11**] considers the information dispersal verifiably in the asynchronous network, and calls it Asynchronous Verifiable Information Dispersal (AVID). In AVID, each fragment can be verified during the dispersal phase and each node stores a valid fragment. Later in the retrieval phase, a client can discard all invalid fragments received from Byzantine nodes when reconstructing the message.

Why do we need ACVID instead of AVID? To construct DPID, we also, for the first time, present an Asynchronous Complete Verifiable Information Dispersal protocol (ACVID). Compared with the conventional Asynchronous Verifiable Information Dispersal (AVID), the ACVID has one extra *completeness* property. If some honest node terminates with some valid fragment, then *all* honest nodes can also terminate the protocol with some valid fragment, which is a critical property for the system.

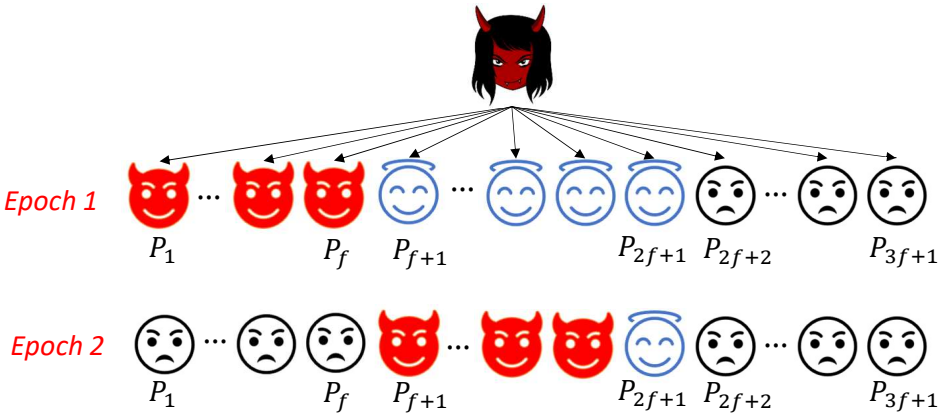

FIGURE 1.1. The very high level of corrupted nodes in AVID

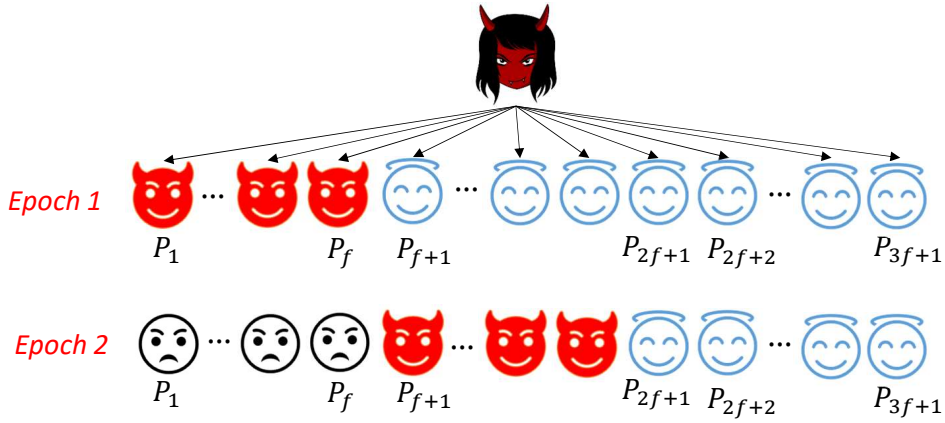

FIGURE 1.2. The very high level of corrupted nodes in ACVID

In fact, we mainly consider AVID in information dispersal by ensuring any recovery client outputs the same data in any epoch e. An adversary can control up to f nodes in each epoch and across both the old and new committees for a total of $2f$ nodes. It is possible that the corrupted nodes in the

old and new committees are the same, and the new committee has fragments at the end of epoch 1, while the old committee has fragments at the start of epoch 2. Specifically, as illustrated in Figure 1.1, at epoch 1, $\mathcal{P}_{f+1}, ..., \mathcal{P}_{2f+1}$ have received the fragment, and any client can recover the same data during epoch 1. However, at the start of epoch 2, if the adversary corrupts nodes $\mathcal{P}_{f+1}, ..., \mathcal{P}_{2f}$ and releases $\mathcal{P}_1, .., \mathcal{P}_f$, then $\mathcal{P}_{2f+1}$ is the only node that has a fragment. In this case, data can never be recovered anymore, and the Refresh protocol cannot even be completed, violating the critical *termination* property.

Therefore, to defend against a mobile adversary, we must ensure that each honest node retains at least one fragment by the end of every epoch. In reality, letting the dealer first call the dispersal black-box sub-protocol and then, if some honest nodes terminate without fragmenting, these nodes execute a recast black-box sub-protocol is an inefficient approach to achieve the ACVID. So, ideally, all honest nodes receive fragments once they terminate. In the state-of-the-art AVID [**2**,**3**,**18**], unlike using Reliable Broadcast (RBC) for the entire data, each node only outputs a portion of the data in information dispersal alongside with a portion of auxiliary information, which is a proof that can be used to verify whether the received fragment is valid or not. These AVID protocols use an online error correcting code to recover the auxiliary information, which needs at least $n - f + r$ messages to tolerate r bad messages. As previously described, only $n - 2f$ honest nodes have the correct fragment of auxiliary information at the start of the new epoch, so it is hard to recover the original data and auxiliary information to discard invalid messages, let alone execute the Refresh protocol.

As depicted in Fig. 1.2, ACVID enforces an additional completeness requirement. Consequently, at the beginning of each epoch, the outgoing committee retains at least $n - 2f$ honest nodes, each holding a valid fragment and the complete auxiliary information as proof. Now, we are ready

to construct DPID0. First, the nodes of the old committee send fragments to all nodes of the new committee. Each node of the new committee collects enough messages from distinct nodes of the old committee, and then recomputes its fragment and auxiliary information to store. The auxiliary information must verify all fragments without recasting, and it must be unforgeable to prevent incorrect data recovery. Note that at most $\mathcal{O}(n)$ fragments from $\mathcal{O}(n)$ different members, so the size of the auxiliary information is $\mathcal{O}(n\lambda)$. We also add a witness for the auxiliary information by collecting $f + 1$ valid digital signatures from distinct nodes in the current committee. In this scenario, each node of the outgoing committee sends to the incoming committee members a message containing its fragment, the associated auxiliary information and the corresponding witness.

Optimistic Case. Next, we consider improving the protocol further by having better performance in optimistic cases, while still keeping secure in *any* case. In the optimistic case, the network needs to be synchronous and all participating nodes must be honest. One simple idea in the optimistic case is that since now every node is honest, we may let anyone serve as a proxy to reduce both the DPID0 all-to-all, and the DPID1 all-to-κ, then κ to all communication to all-to-one and then aggregate, then to all again. The aggregation basically reconstructs the original data and redistributes it. The idea may sound trivial, however, we also need to preserve security in the worst case, where there might be Byzantine nodes in the committee. For example, if there is only one Byzantine node, which happens to be chosen as the "proxy" in that epoch, it may trivially reconstruct m and re-distribute $m' \neq m$, violating the *validity* property of the system.

To address this issue, we first propose employing zero-knowledge proofs, requiring the proxy to demonstrate strict adherence to the protocol and that the redistributed shares correspond exactly to the original input data. To

preserve efficiency, we also need the proof to be succinct. Although theoretically viable, using a generic NIZK or SNARK introduces substantial computational overhead. For the specific relation that nodes must prove, this inefficiency could entirely offset any latency gains from reduced communication costs.

Instead, we offer a new combinatorial technique for avoiding generic NIZK. We design simple combinatorial tricks that leverage a type of set cover mechanism to ensure that the owner also spreads some further auxiliary information from the beginning, which helps verify the digest of the reconstructed value and will be carried in Refresh in each epoch. The key insight is that *every* node will receive sufficient shares for the digest without blowing up communication. In this way, each participant can directly verify the correctness of the new fragments obtained during each Refresh phase. We defer details to Sections 6.1, 6.

Additionally, special care is required during the fallback phase when progress cannot be made in the optimistic case, necessitating the re-invocation of DPID1. To address this, we introduce an ACID protocol that ensures all honest nodes possess both the fragment and the complete auxiliary information upon termination. Without relying on a trusted setup, ACID performs no worse than the protocols proposed in [**2**,**3**,**18**] in the worst-case scenario.

Subtleties of compiling DPID to DPSS. Finally, we compile DPID into DPSS. A natural approach is to first select a random short key k, encrypt the large data m to obtain the ciphertext c, and then combine $\mathsf{DPID}(c)$ with $\mathsf{DPSS}(k)$. However, a subtle issue arises as a malicious dealer might execute $\mathsf{DPID}(c)$ and $\mathsf{DPSS}(k)$ using completely unrelated c and k, where c may not even be a valid ciphertext. Despite this, during the reconstruction phase, two receivers might still obtain the same c and k when attempting to decrypt the ciphertext without using the corresponding secret key.

In such cases, the outcome may not be consistent. This inconsistency is particularly problematic in scenarios where decryption could be randomised, such as in lattice-based encryption schemes. More formally, secure symmetric-key encryption schemes do not guarantee consistent decryption results when the decryption procedure fails. This violates the *agreement* property, which requires that all receivers reconstruct the same data even in the presence of a malicious dealer.

To address this issue, we leverage a recently formulated property called *key committing* [**23**], originally proposed in the context of message franking. When c and k are generated honestly and consistently, the decryption procedure will always yield the original message m. If c and k do not match, all receivers will consistently report decryption failure and output $\perp$, thereby preserving the *agreement* property.

CHAPTER 2

Related Works

Secret Sharing. Secret sharing schemes constitute fundamental building blocks in modern cryptography, enabling secure distribution of sensitive information among multiple parties. First conceptualised independently by Shamir [**40**] and Blakley [**9**] in 1979, these schemes address the critical challenge of protecting cryptographic keys while maintaining accessibility requirements.

Shamir's (k, n) threshold scheme [**40**] leverages polynomial interpolation over finite fields. For secret S, a random polynomial of degree $k-1$ is constructed:

$$f(x) = a_0 + a_1 x + \cdots + a_{k-1} x^{k-1} \tag{2.1}$$

where $a_0 = S$. Shares are generated as $(i, f(i))$ pairs for $i = 1, 2, ..., n$. Secret reconstruction uses Lagrange interpolation:

$$S = \sum_{i=1}^{k} f(i) \prod_{\substack{j=1 \\ j \neq i}}^{k} \frac{x_j}{x_j - x_i} \tag{2.2}$$

The scheme provides information-theoretic security when using a prime modulus $p > S$ [**7**].

Concurrently developed by Blakley [**9**], this scheme represents secrets as intersection points of hyperplanes in n-dimensional space. While mathematically elegant, it requires larger share sizes compared to Shamir's approach.

Threshold Cryptography. The concept of threshold cryptography extends beyond simple secret sharing to encompass more complex access structures. A threshold scheme is defined by a pair (k, n), where n is the total number of participants and k is the minimum number of participants required to reconstruct the secret. This framework was formalised by Ito, Saito, and Nishizeki [**27**], who introduced the notion of general access structures. These structures allow for more flexible sharing policies, such as hierarchical or weighted access, where certain participants may hold more significance than others.

For a given access structure γ, a secret sharing scheme must satisfy two key properties, which are achieved through careful mathematical constructions, often relying on combinatorial designs or algebraic geometry:

- *Correctness.* Any authorised subset of participants in γ can reconstruct the secret.
- *Privacy.* Any unauthorised subset of participants learns no information about the secret.

Verifiable Secret Sharing. One of the major advancements in secret sharing is the introduction of *verifiability*. Traditional schemes assume that the dealer and participants are honest, but in real-world scenarios, malicious behaviour and Byzantine nodes must be taken into consideration. Verifiable Secret Sharing (VSS) schemes, first proposed by Chor et al. [**15**], allow participants to verify the validity of their shares without revealing the secret. This is typically achieved using cryptographic commitments or zero-knowledge proofs.

For example, Feldman's VSS scheme [**20**] employs homomorphic commitments based on discrete logarithms:

$$C_i = g^{a_i} \mod p, \tag{2.3}$$

where g is a generator of a cyclic group, and a_i are the coefficients of the polynomial used to generate shares. Participants can verify their shares by checking:

$$g^{f(i)} \equiv \prod_{j=0}^{k-1} C_j^{i^j} \mod p. \tag{2.4}$$

While VSS schemes address the critical issue of ensuring the validity of shares in the presence of a potentially malicious dealer and some byzantine participants, they primarily operate under the assumption of a synchronous network model. In such models, participants are assumed to communicate within fixed time bounds, and protocols can rely on timely message delivery. However, in real-world distributed systems, network conditions are often asynchronous, with unpredictable delays and potential message losses. This limitation of VSS schemes in asynchronous environments has spurred the development of asynchronous verifiable information dispersal protocols.

Asynchronous Verifiable Information Dispersal. Meanwhile, Asynchronous Verifiable Information Dispersal (AVID) extends the principles of VSS to asynchronous settings, enabling robust secret sharing even when participants experience arbitrary communication delays. By leveraging techniques such as erasure coding [**8**] and Byzantine agreement [**31**], AVID protocols ensure that secrets can be reliably distributed and reconstructed without relying on synchronous communication, thus bridging the gap between theoretical secret-sharing schemes and practical distributed systems. The problem of verifiable information dispersal under the asynchronous network has

historically evolved along two parallel paths: one starting from *AVID*, and the other from *VSS* when confidentiality is required.

AVID is a critical primitive for ensuring data availability and integrity in distributed systems facing Byzantine faults and network disruptions. Its core objective is to allow a dealer to reliably disperse data across nodes such that honest nodes can reconstruct the data even if the dealer or a subset of nodes act maliciously under an asynchronous network. Over the past two decades, AVID protocols have evolved from foundational theoretical frameworks to practical systems with near-optimal efficiency.

The foundational work on AVID was pioneered by Cachin and Tessaro [**13**], who formalised the problem in asynchronous networks with Byzantine faults. Their protocol guarantees that a dealer can disperse data across a network of n nodes such that honest nodes eventually agree on the validity of the dispersed data, even if the dealer or up to $t < n/3$ nodes are malicious.

Cachin and Tessaro's protocol introduced the concept of *verifiable information dispersal*, ensuring that the dispersed data can be reconstructed and verified by honest nodes, even in the presence of adversarial behaviour. Their approach relied on erasure coding and cryptographic techniques to achieve robustness against Byzantine faults. Data is split into $\mathcal{O}(n)$ fragments via erasure coding (e.g., Reed-Solomon codes), ensuring that the data can be recovered if enough honest nodes store fragments, and each fragment is accompanied by a Merkle proof for verifiability. This seminal work established the theoretical framework for AVID. However, their protocol incurred a communication complexity of $\mathcal{O}(n^2\lambda + n\ell)$, where λ is the security parameter and ℓ is the data size, making it inefficient for large-scale systems. It later inspired a series of follow-up studies aimed at improving its efficiency and applicability.

A major focus of later research in AVID has been the reduction of communication complexity, which is critical for scalability in large-scale distributed systems. The quadratic communication overhead of early protocols, such as that of Cachin and Tessaro [**13**], posed significant limitations for practical deployment, especially in networks with a large number of nodes. Addressing this challenge, Tessaro [**42**] achieved a breakthrough by introducing linear threshold signatures (LTS). When integrating into the AVID framework. LTS significantly improved communication efficiency by compressing fragment verification messages into a single aggregated signature, thereby reducing the communication complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(nlogn)$.

The key innovation is to allow multiple parties to collaboratively generate a compact signature that can be verified as if it were produced by a single entity. Specifically, each node generates a partial signature for its fragment, and these signatures are aggregated into a single threshold signature. This eliminates the need for each node to broadcast its individual verification message, drastically reducing the total communication load. The aggregated signature can then be verified by any node in constant time, ensuring that the protocol remains efficient even as the network size grows. The protocol maintains robustness against Byzantine faults, as the threshold signature scheme ensures that malicious nodes cannot forge valid signatures without colluding with a sufficient number of honest nodes.

A significant leap forward was achieved by Das et al. [**18**], who proposed a near-optimal AVID protocol with communication complexity $\mathcal{O}(\ell + n^3\lambda)$. Their approach decoupled the data size ℓ from the network size n, making it scalable for large datasets. The key technical contributions included a two-tiered dispersal mechanism: a lightweight metadata layer using polynomial commitments for verification, and a data layer employing erasure coding for efficient fragment distribution. This design became a blueprint for modern AVID protocols, particularly in dynamic and adversarial environments.

Asynchronous Verifiable Secret Sharing. In parallel, Asynchronous Verifiable Secret Sharing (AVSS) protocols [**12**] extend VSS to asynchronous settings, ensuring both confidentiality and verifiability. Unlike AVID, AVSS inherently protects the secrecy of the dispersed data through cryptographic primitives, but at the cost of higher communication overhead, typically $\Omega(n^2\lambda)$ for secrets of size ℓ, where λ is the security parameter. Recent works, such as those by Alhaddad et al. [**2**], have bridged these paradigms by proposing hybrid approaches that trade partial confidentiality for efficiency.

A critical distinction lies in their objectives: AVID prioritises efficient dispersal of non-sensitive data with verifiability, while AVSS emphasises confidentiality for secret data. Recent advancements, such as the *key-committing* encryption scheme adopted in [**23**], further enable hybrid constructions (e.g., combining AVID and AVSS) to address scenarios where both efficiency and conditional confidentiality are required.

Mobile Adversary. However, all these works do not consider a mobile adversary, where the adversary can eventually corrupt all nodes in a long-lived system. In such a scenario, the adversary can collect the required number of shares to reconstruct the secret in VSS, thereby breaking confidentiality. Similarly, in AVID, the adversary can erase or revise stored fragments, preventing clients from recovering the original data.

To address these challenges, new schemes have been proposed to protect against mobile adversaries. One prominent approach is Proactive Secret Sharing (PSS) and Proactive Verifiable Secret Sharing (PVSS) [**24**]. In these PSS and PVSS protocols, nodes periodically re-randomise their shares without revealing the secret. This re-randomisation ensures that any shares previously compromised by the adversary become obsolete, thereby mitigating the risk of long-term data exposure.

However, PSS and PVSS protocols have significant limitations. First, they primarily focus on refreshing shares within a static committee, assuming that the set of participating nodes remains fixed over time. This assumption does not hold in dynamic environments where nodes may join or leave the committee, as is common in decentralised systems. Second, PSS and PVSS protocols do not inherently support *mobile adversaries* that can shift their corruption targets across different nodes over time. As a result, these protocols fail to provide robust security guarantees in settings with dynamic committees and highly adaptive adversaries.

Dynamic-Committee Proactive Secret Sharing. To overcome these challenges, previous works propose a generalisation scheme called Dynamic-Committee Proactive Secret Sharing (DPSS), where the secret shares can be refreshed to a different committee. While the secret remains unchanged in the new committee, the adversary learns nothing about the secret during this process at the same time.

However, some prior works either incur a high communication cost to reshare a secret [**11**, **36**, **37**, **43**–**47**], or assume the network is synchronous or partial synchronous [**5**, **19**, **22**, **26**, **32**, **37**, **43**, **44**], or consider suboptimal fault tolerance [**5**, **45**]. In this paper, we consider asynchronous DPSS with optimal fault tolerance. The current state-of-the-art of Asynchronous DPSS has communication complexity up to $\mathcal{O}(n^3\ell + n^3\lambda)$. Many confidential BFT-storage applications directly invoke DPSS as the underlying schemes. Clearly, this way is very inefficient, especially for data-storing applications themselves when the data size is large.

A recent interesting work of DPSS [**36**] (called Y-VSS) can tolerate up to $1/2$ corrupted nodes. This work needs to assume the dealer is honest and two committees do not overlap. This outcome is expected under the given assumption. Furthermore, our approach can be readily adapted to achieve $1/2$ fault tolerance within the same framework. However, if we consider

two committees that can intersect, then at the start of epoch 2, just $\mathcal{P}_{2f+1}$ have the correct information. Hence, in this case, all nodes need to keep the whole copy of the data instead of some fragments of the data. Otherwise, the recovery of data will fail.

As previously discussed, every node needs to keep the same data. We found that the construction is also easy for information dispersal, assuming the dealer is honest and up to 1/2 of the nodes are malicious. (1), if the committees do not overlap: in this case, the dealer first invokes reliable broadcast RBC to broadcast data, it make sure all honest nodes received the same data; then in the next epoch, all honest nodes invoke Asynchronous Data Dissemination (ADD) [**17**] with the stored message as input to make sure all honest nodes of new committee also have the same data. If we refresh the encrypted data in this way, and key refresh with Y-VSS [**36**], then the communication complexity is $\mathcal{O}(n\ell + n^4\lambda)$ instead of $\mathcal{O}(n^4\ell + n^4\lambda)$ [**36**]. (2), if the committees have an overlap: in this case, the dealer first invokes RBC to broadcast data M which can make sure all honest nodes received the same data M, then all honest nodes do a (threshold) signature of the hash output of M to generate a witness for this data M. In the next epoch, all honest nodes invoke RBC with the stored message (data and witness) as input, ensuring that all honest nodes have the same data. If we do fault-tolerant storage service without confidentiality in this way, then the communication complexity is $\mathcal{O}(n^2\ell + n^3\lambda)$. With confidentiality, then the communication complexity is $\mathcal{O}(n^2\ell + n^4\lambda)$.

Summary of Gaps. Drawing together the discussion above, the following limitations remain in the literature:

Network Assumptions. Many VSS/PVSS/DPSS-style systems assume synchronous or partially synchronous networks, or accept suboptimal

fault tolerance, which limits applicability to wide-area asynchronous network settings [**5**, **19**, **22**, **26**, **32**, **37**, **43**, **44**].

Scalability of Dispersal/Verification. Early AVID incurs quadratic communication [**13**]. Even near-optimal AVID reduces the data term to $\mathcal{O}(\ell)$ but retains a cubic control-plane term $\mathcal{O}(n^3\lambda)$ [**18**]. AVSS remains $\Omega(n^2\lambda)$ per shared secret [**12**].

Bulk Confidentiality Cost. Directly invoking asynchronous DPSS for large data leads to up to $\mathcal{O}(n^3\ell + n^3\lambda)$ communication, which is problematic for storage-heavy applications (see, e.g., [**11**, **36**, **37**, **43**–**47**]).

Mobile Adversaries. PSS/PVSS refresh shares but typically within a static committee while robust confidentiality need to allow nodes to freely join or leave the committee in each epoch [**24**].

CHAPTER 3

Preliminaries

Notations. We define ℓ as the data size (or input value) and λ as the cryptographic security parameter that captures the size of the signature and the hash value. We use $\mathcal{H}$ to denote a collision-resistant hash function. A protocol message has a syntax of $\mathsf{MsgType}(\mathsf{ID}, \cdots)$, where $\mathsf{MsgType}$ defines the message type and ID is the session identifier representing a specific protocol instance. Throughout the paper, we always consider $\mathcal{O}(n_e) = \mathcal{O}(n)$ and $n_e = 3f_e + 1$ for any epoch e. Namely, our protocol is optimally resilient for any epoch and $\mathcal{O}(n_{e-1}) = \mathcal{O}(n_e) = \mathcal{O}(n)$.

3.1 Building Blocks

In this section, we introduce definitions for some underlying building blocks.

3.1.1 Digital Signature Scheme (DS)

A Digital Signature Scheme (DS) is a triple of probabilistic polynomial-time algorithms algorithms

$$\mathsf{DS} \;=\; \big(\mathsf{DS.Gen},\, \mathsf{DS.Sign},\, \mathsf{DS.Vrf}\big).$$

Key Generation. The algorithm

$$\mathsf{DS.Gen}(1^{\kappa}) \;\rightarrow\; (pk,\, sk)$$

takes as input the security parameter $\kappa \in \mathbb{N}$ and outputs a public key pk and a secret key sk.

Signing. The algorithm

$$\mathsf{DS.Sign}(sk,\, m) \;\rightarrow\; \sigma$$

takes as input the secret key sk and a message $m \in \{0,1\}^*$, and outputs a signature σ.

Verification. The algorithm

$$\mathsf{DS.Vrf}(pk,\, m,\, \sigma) \;\rightarrow\; b$$

takes as input the public key pk, the message m, and the signature σ, and outputs $b = 1$ if σ is valid under pk, and $b = 0$ otherwise.

To ensure the correctness of the signature scheme, for every security parameter κ and every message m, if

$$(pk, sk) \leftarrow \mathsf{DS.Gen}(1^\kappa) \quad \text{and} \quad \sigma \leftarrow \mathsf{DS.Sign}(sk, m),$$

then

$$\Pr\big[\mathsf{DS.Vrf}(pk, m, \sigma) = 1\big] \;=\; 1.$$

Additionally, there is no probabilistic polynomial-time adversary $\mathcal{A}$ that can, after making at most a polynomial number of adaptive signing queries to $\mathsf{DS.Sign}(sk, \cdot)$, produce a fresh forgery (m^*, σ^*) such that

$$m^* \notin Q \quad \text{and} \quad \mathsf{DS.Vrf}(pk, m^*, \sigma^*) = 1,$$

with non-negligible probability. Here Q is the set of messages queried by $\mathcal{A}$.

Using the triple of probabilistic polynomial-time algorithms together helps achieve the following properties:

- *Authenticity & Integrity.* Only the holder of the secret key sk can produce a valid signature that DS.Verify will accept.
- *Non-repudiation.* A signer cannot plausibly deny having generated a valid signature on a given message.
- *Efficiency.* All algorithms run in time polynomial in the security parameter κ and the length of the message $|m|$.

Within our DPID protocols, digital signatures (DS) are primarily employed to authenticate message senders and filter out messages with invalid signatures.

3.1.2 Erasure Code Scheme

A (k, n)-*erasure code scheme* [**8**] is a pair of deterministic algorithms

$$(\mathsf{ENC},\, \mathsf{DEC})$$

with the following guarantees: given a message M, the encoder ENC produces n fragments

$$\mathbf{m} \;=\; \{\, m_1,\, m_2,\, \ldots,\, m_n\} \;=\; \mathsf{ENC}(M)\,,$$

such that any subset of k out of the n fragments suffices to recover M via the decoder DEC.

Encoding (ENC).

$$\mathsf{ENC}:\; M \;\mapsto\; (m_1, \ldots, m_n)$$

Input: a message $M \in \{0,1\}^*$.

Output: an n-tuple of fragments $(m_1, \ldots, m_n)$.

The encoder must satisfy that any k-subset of these n fragments contains enough information to reconstruct M.

Decoding (DEC).

$$\mathsf{DEC}: \ (m_{i_1}, \ldots, m_{i_k}) \ \mapsto \ M \quad \text{for any } \{i_1, \ldots, i_k\} \subseteq [n].$$

Input: any k fragments $(m_{i_1}, \ldots, m_{i_k})$.

Output: the original message M.

To ensure that any k-subset of these n fragments can reconstruct the message M, DEC must satisfy that:

$$\forall M, \quad \mathsf{DEC}\big(\mathsf{ENC}(M)_{i_1,\ldots,i_k}\big) = M \quad \text{for all } 1 \leq i_1 < \cdots < i_k \leq n.$$

Using the pair of deterministic algorithms here helps achieve the following properties:

- *Reliability.* Any k out of the n fragments suffice to reconstruct M, tolerating up to $n - k$ erasures.
- *Efficiency.* Both ENC and DEC run in time polynomial in the message length and in n.
- *Storage Overhead.* The total storage across n fragments grows by a factor of at most n/k.

Within the DPID protocols, an erasure coding scheme is used to split the original secret into multiple shares assigned to individual nodes, and subsequently to reconstruct the secret from a qualified set of shares.

3.1.3 Cryptographic Hash Function

For a Cryptographic Hash Function ($\mathcal{H}$), let $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^\ell$ be a deterministic function that maps an arbitrary-length bit string x to a fixed-length digest d. We write

$$d \ \leftarrow \ \mathcal{H}(x)$$

to denote the evaluation of $\mathcal{H}$ on input x.

A secure cryptographic hash function is expected to achieve the following properties:

- *Preimage Resistance.* Given d, it is infeasible to find any x such that $\mathcal{H}(x) = d$.
- *Second-Preimage Resistance.* Given x, it is infeasible to find $x' \neq x$ with $\mathcal{H}(x') = \mathcal{H}(x)$.
- *Collision Resistance.* It is infeasible to find any distinct pair (x, x') such that $\mathcal{H}(x) = \mathcal{H}(x')$.

In the DPID protocols, the hash function is employed to map each message into a compact digest, thereby reducing its size and enabling efficient verification.

3.1.4 Reliable Broadcast (RBC)

Reliable Broadcast (RBC) is a protocol running among a set of n nodes [**10**], where a node called the dealer aims to broadcast an input message m to all the other nodes. Formally, the RBC protocol has the following properties:

- *Agreement*. If two honest nodes output v and v' respectively, then $v = v'$;
- *Totality*. If an honest node outputs v, then all honest nodes output v;
- *Validity*. If the sender is honest and inputs v, then all honest nodes output v.

In the DPID protocols, reliable broadcast (RBC) is used in the initial epoch to guarantee the secure and consistent distribution of messages across all nodes in the committee.

3.1.5 Asynchronous Data Dissemination (ADD)

The (ADD) protocol [**17**] works in a network of $n = 3f + 1$ nodes where up to f nodes could be malicious, a subset of at least $f + 1$ honest nodes start with a common message M and other honest nodes start with $\perp$, and it guarantees that all honest nodes eventually output M. In this dissertation, we consider that at least $f_{e-1} + 1$ honest nodes in the old committee C_{e-1} share a common message M, while the remaining honest nodes have $\perp$ as their input. The ADD algorithm guarantees that all honest nodes in the new committee C_e will eventually output M.

The ADD satisfies the following properties except with negligible probability:

- *Termination.* If at least $f_{e-1} + 1$ honest nodes of the old committee input the same value M and the rest of the honest nodes of the old committee input $\perp$, then each honest node of the new committee would output a value;
- *Validity.* If one honest node of the new committee outputs M, then M is the input of some honest node of the old committee.
- *Agreement.* If at least $f_{e-1} + 1$ honest nodes of old committee input the same value M and the rest of honest nodes of old committee have $\perp$, then each honest node of new committee would output M.

Note that ADD [**17**] ensures that all honest nodes within the same committee produce the same output value. Throughout the paper, whenever a node in the old committee invokes ADD, we assume that all messages from the old committee are sent to all nodes, including those in the new committee. The new committee members simply listen and simulate being honest nodes with $\perp$ as their input from the old committee, resulting in the new committee members also having the same output. The communication complexity of ADD is $\mathcal{O}(n|m| + n^2\lambda)$ assuming the size of the input is $|m|$.

In our DPID protocols, Asynchronous Data Dissemination (ADD) serves as the mechanism for securely delivering auxiliary information from the old committee to the newly formed committee, thereby ensuring continuity across epochs.

3.1.6 Committee Election (CE)

A (κ, ϵ)-Committee Election (CE) protocol [**25**] can be constructed directly from a threshold coin-tossing and executed among n nodes. If at least $f+1$ honest nodes participate, the protocol can output a κ-sized subcommittee C with an honest majority. Specifically, a protocol is said to be (κ, ϵ)-committee election if it satisfies the following properties except with negligible probability:

- *Termination.* If $f + 1$ honest nodes participate in the protocol, then all honest nodes output C;
- *Agreement.* All honest nodes output the same committee C;
- *Validity.* If any honest node outputs C, then $|C| = \kappa$, and the probability of every node $P_i \in C$ is the same while C has an honest majority;
- *Unpredictability.* The probability of predicting the returned committee is at most $1/\binom{n}{\kappa}$, unless at least one honest node has participated in the protocol execution.

Within the DPID protocols, Committee Election (CE) serves as the mechanism for securely electing an honest-majority subcommittee, which functions as the intermediary to relay messages to the new committee.

3.2 Models and Goals

We now describe our system model. The details of system modelling were illustrated as follows:

Setup. We consider a fully connected distributed system composed of a universe of clients/nodes $\Pi := \{\mathcal{P}_1, \mathcal{P}_2, \cdots\}$ and a dealer $\mathcal{P}_d$, all nodes (including the dealer) have a unique identity. The whole lifetime of the system is divided into fixed intervals of predetermined length called epochs. In each epoch e, a group of nodes C_e (called the committee for that epoch) has been specified, and the size of C_e is n_e.

Besides, suppose that the current epoch is e, we also need all nodes (non-committee members and committee members) to know this public information. For clarity, we refer to the current epoch e's committee as the new committee and the previous epoch $e-1$'s committee as the old committee. Concretely, in an epoch e, there exists a committee C_e of size n_e. For any epoch e, we assume that all involved threshold cryptosystems are properly set up within C_e, such that all participating nodes can get and only get their own secret keys in addition to all relevant public keys. Any node local keeps public information of epoch $e-1$ and e. For clarity, we refer to the current epoch e's committee as the new committee, and the previous epoch $e-1$'s committee as the old committee.

Adversarial Model. We consider a very strong adversary (called *weak mobile corruption*) with probabilistic polynomial-time computing power, who can corrupt nodes at any time, then release some and corrupt others. For instance, the adversary can get all the faulty nodes' initial internal states and also can let these nodes deviate from the protocol arbitrarily. The only restriction is that no more than f_e are corrupted in each epoch e (such that

$3f_e + 1 \leq n_e$), and these are fully controlled by a probabilistic polynomial-time bounded adversary [**12**, **33**]. But over the whole lifetime, it is possible that every node has been corrupted. Once the adversary corrupts a node, it is assumed to be corrupted until the end of the current epoch. Following the epoch definition [**37**, **46**], we assume that the adversary does not actively attack the system across multiple epochs to allow us to overcome the impossibility [**1**]. Note that during the refresh between epochs $e - 1$ and e, the committees C_{e-1} and C_e may intersect, and the adversary may control a given node $\mathcal{P}_i$ in both the old and new committees. A node would be "released" by an adversary if it is no longer corrupted in a new epoch. Additionally, as [**11**, **46**], we assume channels that ensure a message can be received in an epoch if and only if it has been sent in the same epoch.

Communication Model. We make the assumption that there is an asynchronous network of connected nodes where each pair of nodes can communicate through a reliable, authenticated channel. In an asynchronous network, the adversary has the ability to arbitrarily delay and reorder messages within an epoch, as well as fully determine when the message is available to the receiver. The adversary cannot drop or modify this message among honest nodes except to delay it.

Dynamic-Committee Proactive Information Dispersal. Our first goal is to design an efficient Dynamic-committee proactive information dispersal (DPID) in an asynchronous network. In this protocol, each epoch has a specific committee responsible for the stored data, which may be a block when the protocol is used in a blockchain. Any client can recover the data if it can receive $f_e + 1$ valid fragments in the current epoch e, and the data is the same even if retrieved in a different epoch. Formally, for any node of new committee C_e in epoch e, the DPID satisfies the following properties with all but negligible probability:

- *Termination.* If a correct dealer initiates DPID, then all honest nodes eventually output a value.
- *Agreement.* If an honest node outputs a value, then all honest nodes eventually output a valid value.
- *Availability.* If an honest node can output a value, then any honest client eventually can reconstruct B' once it receives at least $f_e + 1$ valid values.
- *Correctness.* If an honest node can output a value, then any two honest clients eventually can reconstruct the same value B'. If the dealer was honest and input B, then $B' = B$.
- *Integrity.* If an honest node of C_e outputs a value, then any honest clients eventually can reconstruct a value B' during epoch e. If any honest node of C_{e-1} can output a value, then any honest client eventually can reconstruct a value B'' during epoch $e - 1$. Further, $B' = B''$.

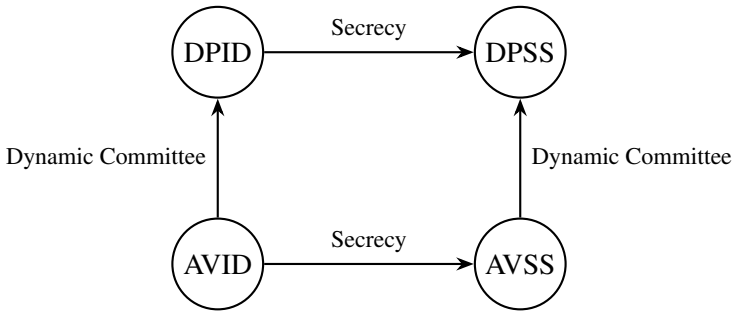

FIGURE 3.1. Relationship between AVID/AVSS/DPID/DPSS

Dynamic-Committee Proactive Secret Sharing. Our end goal is to design an efficient asynchronous Dynamic-committee proactive secret sharing (DPSS). In this protocol, each epoch has a specific committee responsible for the confidential data (or secret), and any client can always recover confidential data if it can receive $f_e + 1$ valid messages in the current epoch e, the confidential data remains the same even when recasting in a different epoch. The relationship among AVID, DPID, and DPSS is shown in Fig. 3.1. In particular, DPSS is essentially DPID endowed with the secrecy property. Besides that, the adversary learns no information about the data except by

recasting it. Formally, for any node of new committee C_e in epoch e, the DPSS satisfies the following properties with all but negligible probability:

- *Termination.* If a correct dealer initiates DPSS, then all honest nodes eventually output a valid share.
- *Agreement.* If an honest node outputs a valid secret share, then all honest nodes eventually output a valid share.
- *Availability.* If an honest node can output a secret share, then any honest client eventually can reconstruct some secret S' once it receives at least $f_e + 1$ valid secret share.
- *Correctness.* If an honest node can output a secret share, then any two honest clients eventually can reconstruct the same secret S'. If the dealer was honest and input S, then $S' = S$.
- *Integrity.* If an honest node of C_e outputs a value, then any honest clients eventually can reconstruct a secret S' during epoch e. If any honest node of C_{e-1} can output a value, then any honest client eventually can reconstruct a secret S'' during epoch $e - 1$. Further, $S' = S''$.
- *Secrecy.* If the adversary corrupts no more than f_e nodes in a committee of any epoch e, then the adversary learns no information about the secret S.

Remark: In both DPID and DPSS, the termination property implies liveness, that is, if the dealer is honest, then the adversary cannot prevent the message or secret from being transferred from the old committee to the new committee during Refresh phase.

Complexity Measures. The practicality of BFT protocols depends heavily on their computational complexity. In this paper, we mainly consider the following three metrics during the protocol:

- *Message complexity* [**12**]: the expected total number of messages generated by honest nodes;

- *Communication complexity* [**12**]: the expected total number of bits exchanged among honest nodes;
- *Time (round) complexity* [**14**]: the expected (asynchronous) rounds of communication before the protocol terminates.

CHAPTER 4

DPID0: Asynchronous Dynamic-Committee Proactive Information Dispersal

---

In this section, we present our first DPID, denoted as DPID0. This protocol relies solely on the public key infrastructure (PKI) for digital signatures. Notably, DPID0 operates independently of any consistent global view, such as a consensus mechanism or a shared coin.

High level overview of DPID0. The core of our design is to present a new information dispersal protocol. As briefly elaborated in the Introduction, the previous information dispersal only guarantees $f + 1$ honest nodes can receive correct fragments at the end of the dispersal phase. However, all honest nodes can terminate with the same auxiliary information as proof in this phase. This auxiliary information (sometimes also called proof information) can help verify every fragment received by nodes, whether valid or not. During Refresh, as the adversary is proactive, the f out of $f + 1$ honest nodes could be corrupted in the next epoch so that only one honest node with valid input participates in the Refresh phase. In contrast, other honest nodes participate in Refresh with no valid input value.

A possible solution is first to execute the dispersal subprotocol of AVID. If some honest node finds that it has not received the fragments once the dispersal phase terminates, it invokes the retrieval subprotocol of AVID. However, it is possible that it found the dealer had sent junk data in the first place. For example, the regenerated fragments may become inconsistent

with the auxiliary information. In such cases, nodes are uncertain about which fragment should be stored other than $\perp$. However, ensuring the validity of $\perp$ is challenging. If $\perp$ were considered valid, corrupt nodes could always send $\perp$ to the recast (retrieval) nodes. This would cause the recast nodes to repeatedly use $\perp$ for decoding, potentially leading to agreement issues. Let us consider the following scenario at the end of epoch $e-1$: If $f+1$ honest nodes store valid fragments, but these fragments can only reconstruct junk data, the remaining honest nodes will store $\perp$. At the start of epoch e, suppose only one honest node holds a valid fragment while all other honest nodes hold $\perp$. A new committee member requires $f+1$ valid fragments to proceed in this situation. However, since $\perp$ is treated as an invalid message, the Refresh process will become stuck, as it cannot gather enough valid fragments to reconstruct the data, violating the $termination$ property.

Another possible solution is as follows: if an honest node has not received the corresponding fragment and instead retrieves junk data from $f+1$ valid fragments, then the honest node stores the $f+1$ valid fragments instead of outputting a $\perp$. In this case, at least $f+1$ honest nodes will terminate with one fragment each, and f honest nodes will terminate with $f+1$ fragments. It is important to note that the size of $f+1$ fragments equals the size of the original data. As a result, the communication complexity during the Refresh process will be up to $\mathcal{O}(n^2\ell)$, where ℓ is the size of the data (for further details, see Algorithm 4).

In our DPID0, we let honest nodes instantly invoke the retrieval subprotocol once the dispersal subprotocol terminates. Thanks to the $agreement$ property of the dispersal subprotocol, we can ensure that all honest nodes eventually output the same in the retrieval subprotocol. If the data is not $\perp$, all honest nodes recompute their corresponding fragment and auxiliary information as the output. Otherwise, if the data is instead $\perp$, all honest

Algorithm 1 The DPID0 dispersal (ACVID), with epoch $e = 1$ and sender $\mathcal{P}_s$ for node $\mathcal{P}_i$

Initialisation: Let RBC$[e]$ refer to one instance of the reliable broadcast protocol, where $\mathcal{P}_\mathsf{s}$ is the sender of RBC$[e]$; $T_e \leftarrow \{\}$
Public parameter: n_e, f_e

if $\mathcal{P}_i = \mathcal{P}_\mathsf{s}$ and received input M **then** ▷ For the dealer
  invoke RBC$[e]$ with input M
upon receiving output M from RBC$[e]$ **do** ▷ For any node in the committee
  if $M \neq \perp$ **then**
    $\{m_j\}_{j\in[n_e]} \leftarrow \mathsf{Enc}(M, n_e, f_e)$
    $\mathbf{D}_e \leftarrow \{\mathcal{H}(m_1), \ldots, \mathcal{H}(m_{n_e})\}$
    $\sigma_i \leftarrow \mathsf{DS.Sig}(e, \mathcal{H}(\mathbf{D}_e))$
    multicast $\mathsf{FINAL}(e, \sigma_i)$ to C_e
    upon receiving $\mathsf{FINAL}(e, \sigma_j)$ from $\mathcal{P}_j (\in C_e)$ **do**
      if $\mathsf{DS.Vrf}(e, \mathcal{H}(\mathbf{D}_e), (j, \sigma_j))$ =1 **then**
        $T_e \leftarrow T_e \cup \{(j, \sigma_j)\}$
    upon $|T_e| = f_e + 1$ **do**
      $block[e] \leftarrow (m_i, \mathbf{D}_e, T_e)$, and output $block[e]$

nodes set both their fragment and auxiliary information to $\perp$. In either case, all honest nodes eventually share the same auxiliary information. Besides, all honest nodes must sign the auxiliary information in order to generate a witness that can convince any node that the auxiliary information is unique and valid for a certain epoch. By doing so, we ensure that the system remains robust against failures and adversarial behavior, even under non-ideal or partially synchronous network conditions.

We note that the instant invocation of the retrieval subprotocol once the dispersal terminates can be viewed as the process of the RBC (Reliable Broadcast) protocol. Therefore, in our DPID0 during epoch 1, all honest nodes first wait for the output of RBC, then compute the auxiliary information $\mathbf{D}_e$, the corresponding fragments, and sign the auxiliary information to form a witness that proves $\mathbf{D}_e$ is unique. This ensures that the consistency of the auxiliary information and all fragments is maintained across all honest nodes. Based on these principles, we have our ACVID protocol.

The details of DPID0**.** Now, we describe our DPID0 protocol. The formal description of the protocol is shown in Algorithm 1,2,3 and 4. The DPID0

Algorithm 2 Validation function of DPID0, with epoch e

Public parameter: n_e, f_e
function Verify$(i, block[e])$
1 **if** $block[e] \neq \emptyset$ **then**
2 parse $block[e] := (m_i, \mathbf{D}_e, T_e)$
3 **if** $|T_e| = f_e + 1$ and $\mathcal{H}(m_i) = \mathbf{D}_e[i]$ **then**
4 **if** DS.Vrf$(e, \mathcal{H}(\mathbf{D}_e), (j, \sigma_j)) = 1$ for any $(j, \sigma_j) \in T_e$ **then**
5 return 1
6 **else** return 0

Algorithm 3 The DPID0 Refresh, with epoch $e > 1$ for node $\mathcal{P}_i$

Initialisation: $T \leftarrow \{\}; T_e \leftarrow \{\}$
Public parameter: $n_{e-1}, f_{e-1}, n_e, f_e$
1 **if** $\mathcal{P}_i \in C_{e-1}$ **then** ▷ Old committee C_{e-1}
2 **if** Verify$(i, block[e-1]) = 1$ **then**
3 parse $block[e-1] := (m_i, \mathbf{D}_{e-1}, T_{e-1})$
4 ADD$[e](\mathbf{D}_{e-1})$
5 **multicast** FRESH(e, m_i) to C_e
6 **if** $\mathcal{P}_i \in C_e$ **then** ▷ New committee C_e
7 **upon** receiving FRESH(e, m_j) from $\mathcal{P}_j (\in C_{e-1})$ **do**
8 $T = T \cup \{(j, m_j)\}$
9 **wait** until ADD$[e]$ return $\mathbf{D}_{e-1}$
10 parse $\mathbf{D}_{e-1} := \{h_1, \cdots, h_{n_{e-1}}\}$
11 **for** any $(j, m_j) \in T$ **do**
12 **if** $\mathcal{H}(m_j) \neq h_j$ **then** discard (j, m_j) from T
13 **upon** $|T| = f_{e-1} + 1$ **do**
14 $M' \leftarrow$ Dec(T)
15 $\{\overline{m_j}\}_{j \in [n_e]} \leftarrow$ Enc(M', n_e, f_e)
16 $\mathbf{D}_e \leftarrow \{\mathcal{H}(\overline{m_1}), \ldots, \mathcal{H}(\overline{m_{n_e}})\}$
17 $\sigma_i \leftarrow$ DS.Sig$(e, \mathcal{H}(\mathbf{D}_e))$
18 **multicast** FINAL(e, σ_i) to C_e
19 **upon** receiving FINAL(e, σ_j) from $\mathcal{P}_j (\in C_e)$ **do**
20 **if** DS.Vrf$(e, \mathcal{H}(\mathbf{D}_e), (j, \sigma_j))$ =1 **then**
21 $T_e \leftarrow T_e \cup \{(j, \sigma_j)\}$
22 **upon** $|T_e| = f_e + 1$ **do**
23 $block[e] \leftarrow (\overline{m_i}, \mathbf{D}_e, T_e)$, and output $block[e]$

protocol mainly consists of two phases, namely, the dispersal phase (for epoch 1) and the refresh phase (for epoch $>$ 1). Specifically, the dealer disperses its data into every node inside the committee of epoch 1 through ACVID (Algorithm 1), and then the old committee refreshes its stored fragment into the new committee through DPID (Algorithm 3) in any epoch $e > 1$.

The detailed protocol proceeds as follows:

- *Dealer disperses data in epoch 1 (Algorithm 1 of* ACVID*)*: Upon receiving the input data M, the dealer $\mathcal{P}_s$ activates the $\mathsf{RBC}[e]$ protocol as the sender, with M as the input. Once any honest node in the epoch 1 committee receives the output of RBC, it computes the fragments using the deterministic encoding algorithm and generates the same **D** as the auxiliary information. After generating **D**, each honest node signs the hash of **D** and waits for $f_e + 1$ valid signatures for the same **D** from distinct nodes within the same committee to form a witness. Finally, the node combines the corresponding fragment, **D** and the collected signatures to form a $block[1]$, which serves as the output. Due to the agreement and termination properties of RBC, along with the deterministic nature of the encoding algorithm, all honest nodes will compute the same **D**. Note that we need to verify the stored fragment stored inside each node.

  Otherwise, the adversary could arbitrarily insert fake fragments into the nodes' local storage, which would compromise the security of the system. For example, let $M_1 \neq M_2$, $n_e = n_{e-1} = 4$, and $f_e = f_{e-1} = 1$. Then, $\{m_1, m_2, m_3, m_4\} \leftarrow \mathsf{Enc}(M, 4, 1)$ and $\{m'_1, m'_2, m'_3, m'_4\} \leftarrow \mathsf{Enc}(M', 4, 1)$. Let $\mathbf{D} \leftarrow \{\mathcal{H}(m_1)\}_{i\in[4]}$ and $\mathbf{D'} \leftarrow \{\mathcal{H}(m'_1)\}_{i\in[4]}$. Suppose $C_{e-1} := \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_4\}$ and $C_e := \{\mathcal{P}'_1, \mathcal{P}'_2, \mathcal{P}_3, \mathcal{P}_4\}$, where $\mathcal{P}_4 \in C_{e-1}$ is a malicious node during epoch $e-1$ and outputs (m'_4, D') in ACVID, while $\mathcal{P}_4 \in C_e$ is an honest node in epoch e. That is, the adversary releases control over $\mathcal{P}_4$, and $\mathcal{P}_3 \in C_e$ becomes the malicious node. If there is no witness for the auxiliary information **D**, then $\mathcal{P}_4$ will send $\{m'_4, \mathbf{D'}\}$ to all nodes in C_e, while $\mathcal{P}_3$ sends $\{m'_3, \mathbf{D'}\}$ to all nodes in C_e. The adversary delays all other messages from nodes in C_{e-1}. At the end of epoch e, $\mathcal{P}'_1$ will output $\{m'_1, \mathbf{D'}\}$, $\mathcal{P}'_2$ will output $\{m'_2, \mathbf{D'}\}$, and $\mathcal{P}_3$ will output $\{m'_3, \mathbf{D'}\}$. However, this scenario violates the security because any client can retrieve two different values in epochs $e-1$ and e.

- Refresh *for every epoch after epoch 1 (Algorithm 3)*: Each node $\mathcal{P}_i$ in the old committee first checks whether its local $block[e-1] := \{m_i, \mathbf{D}_{e-1}, T_{e-1}\}$ is valid (via Algorithm 2). If valid, the node then invokes ADD with $\mathbf{D}_{e-1}$ as input and sends the fragment to all nodes in C_e using the FRESH message.

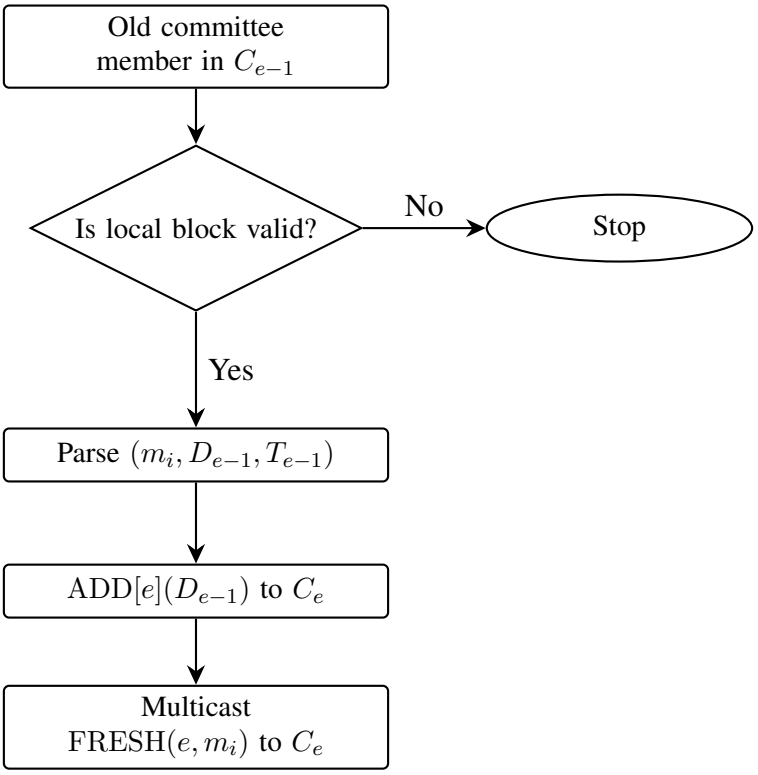

FIGURE 4.1. DPID0 Refresh — Old Committee

All nodes in the new committee C_e wait for the output of ADD$[e]$ and then filter out any invalid fragments received from nodes in the old committee C_{e-1}. Once they receive $f_{e-1} + 1$ valid fragments, they try to decode the original message M' and recompute $\mathbf{D}_e$. After that, all honest nodes sign the hash of $\mathbf{D}_e$. Once a witness for this epoch is formed, all nodes in C_e output. See the flowcharts 4.1 and 4.2 for further details.

- *Retrieval at any epoch (Algorithm 4)*: When a node $\mathcal{P}_j$ wants to retrieve data at any epoch e, it will multicast a RETRIEVAL(e) message to C_e. For any honest node in C_e, upon receiving a RETRIEVAL(e) message from $\mathcal{P}_j$, if its local $block[e]$ is valid, the node will send a

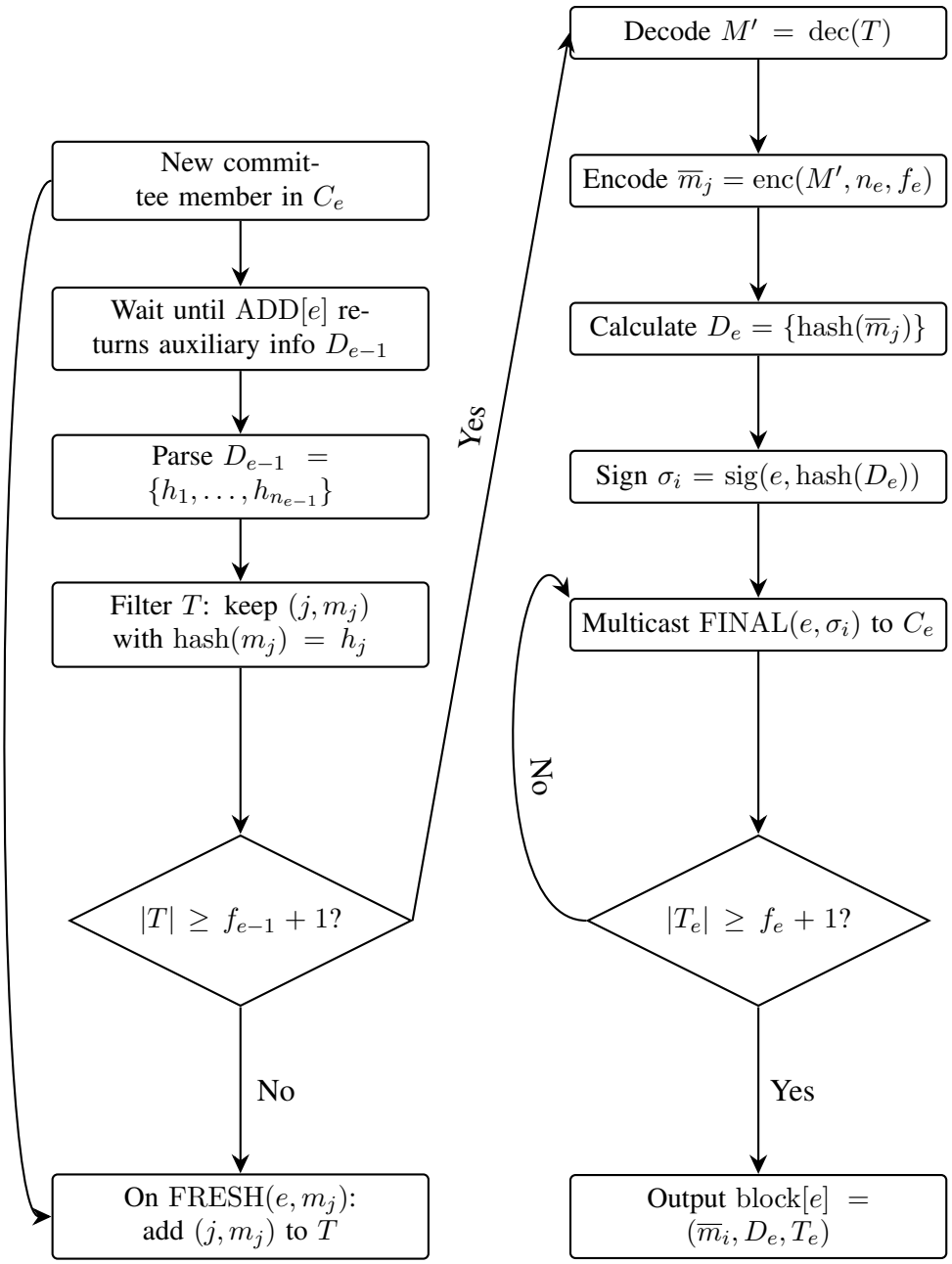

FIGURE 4.2. DPID0 Refresh — New Committee

RECAST$(e, block[e])$ message to $\mathcal{P}_j$. And $\mathcal{P}_j$ will wait for $f_e + 1$ valid RECAST messages from distinct nodes of C_e. Once it has received enough number of RECAST messages, it will decode the fragments into the original data and output the result.

- *Validation function of* DPID0 *at any epoch* e *(Algorithm 2)*: The validation function helps ensure that the fragment is valid and consistent with

Algorithm 4 The DPID0 retrieval, with epoch e for node $\mathcal{P}_i$

Initialization: $T \leftarrow \{\}$
Public parameter: n_e, f_e

if $\mathcal{P}_i$ would like to retrieve the original value **then**
  multicast RETRIEVAL(e) to C_e
upon receiving RETRIEVAL(e) from $\mathcal{P}_j$ and $\mathcal{P}_i \in C_e$ **do**
  if Verify$(i, block[e])$=1 **then**
    send RECAST$(e, block[e])$ to $\mathcal{P}_j$
upon receiving RECAST$(e, block[e])$ from $\mathcal{P}_j (\in C_e)$ **do**
  if Verify$(j, block[e])$=1 **then**
    parse $block[e] := (m_j, \mathbf{D}_e, T_e)$
    $T \leftarrow T \cup \{(j, m_j)\}$
upon $|T| = f_e + 1$ **do**
  $M' \leftarrow$ Dec(T), and output M'

the auxiliary information in $block[e]$. Additionally, the auxiliary information must be endorsed by at least $f_e + 1$ nodes to be considered valid. This guarantees that the fragment is not only consistent with the stored data but also has the necessary endorsements to prove its authenticity.

Security intuition of DPID0**:** Algorithms 1-4 implement all properties of DPID. Intuition is as follows:

- *Termination.* For epoch $e = 1$, the termination follows immediately from the *validity* and *agreement* properties of the RBC protocol. All honest nodes will have the same $\mathbf{D}_e$ and sign the hash of $\mathbf{D}_e$. Consequently, all honest nodes can wait for $f_1 + 1$ signatures for $\mathbf{D}_e$ to form a witness and then output $block[1]$. At the start of epoch 2, according to Algorithm 1, we know that at least $f_1 + 1$ honest nodes have valid $block[1]$. Therefore, every honest node in epoch 2 can wait for $f_1 + 1$ valid $block[1]$ during the refresh phase, ensuring that all honest nodes can output $block[2]$. Similarly, at the start of epoch e, at least $f_{e-1} + 1$ honest nodes have valid $block[e-1]$. Every honest node in epoch e can wait for $f_{e-1} + 1$ valid $block[e-1]$ from the previous committee, ensuring that all honest nodes in the new committee can output $block[e]$.

- *Agreement.* Following the *termination* of DPID, all honest nodes in the new committee can output a $block[e]$ at epoch e. Then, following the validation function, all honest nodes will output a valid $block[e]$.
- *Availability.* If an honest node of the new committee can output a $block[e]$, then, according to the *agreement* of DPID, all honest nodes in the new committee will also be able to output a valid $block[e]$. Therefore, for any node $\mathcal{P}_j$ that wants to retrieve data at any epoch e, it can wait for $f_e + 1$ valid $block[e]$ to eventually reconstruct B'.
- *Correctness.* If an honest node $\mathcal{P}_i$ can output a $block[e]$ containing $\mathbf{D}_e$, then, according to Algorithm 3, at least $f_{e-1} + 1$ nodes must have sent valid FRESH messages to $\mathcal{P}_i$. This implies that at least one honest node in the previous committee C_{e-1} has a valid $block[e-1]$. Based on the validation function, we know that at least one honest node must have signed the auxiliary information $\mathbf{D}_{e-1}$. Now, suppose that another honest node generates a $block'[e]$ with $\mathbf{D}'_e \neq \mathbf{D}_e$. This would imply that at least two honest nodes in C_{e-1} have generated different auxiliary information, and both formed a valid witness for it, i.e., $\mathbf{D}'_{e-1}$ and $\mathbf{D}_{e-1}$, such that $\mathbf{D}'_{e-1} \neq \mathbf{D}_{e-1}$. The recursion then implies that the *agreement* property failed in epoch 1 after initial dispersal, which is clearly impossible according to Algorithm 1. Hence, any two honest nodes must have the same $\mathbf{D}_e$. Following the *availability* property of DPID, any honest client can reconstruct some block B' with the same auxiliary information $\mathbf{D}_e$, ensuring that any two honest clients will eventually reconstruct the same B'. If the dealer was honest and inputted B in epoch 1, then all honest nodes in the new committee will reconstruct the same B, recompute a new $\mathbf{D}_e$ related to B, and hence $B' = B$.
- *Integrity.* If any honest node of C_e can output a value, following the above analysis on the *correctness* of DPID, it also implies that all honest nodes of C_e can reconstruct the same $B' = M'$ (as per line 14 of algorithm 3). Furthermore, we note that all message shares inside the

FRESH message were generated from M' via a deterministic encoding algorithm (as shown in line 15 of algorithm 3), meaning that any honest client in epoch $e-1$ can reconstruct a block $B'' = M'$. Therefore, we have $B' = B''$.

Complexities of DPID0**.** Throughout this paper, when we refer to the communication complexity, we focus on the cost during the Refresh phase. Let ℓ denote the bit-length of the input, and λ represent the size of the security parameter. The communication cost primarily appears in three parts:

- ADD*:* The cost of ADD is $\mathcal{O}(n|m| + n^2\lambda)$. In DPID0, the size of m is $n\lambda$, so the cost of ADD is $\mathcal{O}(n^2\lambda)$.
- FRESH *message:* The size of a FRESH message is $\mathcal{O}(\ell/n + \lambda)$. Hence, the total cost of the FRESH message is $\mathcal{O}(n\ell + n^2\lambda)$.
- FINAL *message:* The size of the FINAL message (which includes the signature) is $\mathcal{O}(\lambda)$. Thus, the total cost of the FINAL message is $\mathcal{O}(n^2\lambda)$.

In summary, the communication complexity of DPID0 is $\mathcal{O}(n\ell + n^2\lambda)$.

CHAPTER 5

DPID1: Asynchronous Dynamic-Committee Proactive Information Dispersal with Better Asymptotic Complexity

---

In DPID0, the communication complexity of Refresh can be up to $\mathcal{O}(n\ell + n^2\lambda)$, but the storage complexity remains $\mathcal{O}(\ell + n^2\lambda)$. Hence, a $\mathcal{O}(n)$ gap exists between the communication and storage complexity. In this section, we would like to reduce the $n\ell$ term of communication complexity such that this term is proportional to the number of nodes.

In the Refresh phase of our DPID0, all committee nodes in C_{e-1} directly multicast their $block[e-1]$ to all new committee nodes in C_e. Essentially, we can select κ nodes from C_e to form a subcommittee, ensuring that at least $\lceil \frac{\kappa}{2} \rceil + 1$ of the nodes are honest with high probability. The subcommittee functions as a middle layer to help relay all messages from the old committee to the new committee. Specifically, any node in the subcommittee performs the same operations as the new committee member described in the previous DPID0 refresh protocol Algorithm 3. However, the node will compute $block'[i] := \{\overline{m_i}, \mathbf{D}_e\}$ for $i \in [n_e]$, and then sends $block'[i]$ to $\mathcal{P}_i \in C_e$. For any honest node $\mathcal{P}_i$ in C_e, if it receives $\kappa/2 + 1$ identical $block'[i]$ messages, it will extract $\mathbf{D}_e$ and execute the same process in lines 17-23 of Algorithm 3. In this way, the communication complexity of DPID1 is $\mathcal{O}(\kappa\ell + \kappa n^2\lambda)$, which improves the dominant term from $n\ell$ to $\kappa\ell$. In contrast, the communication complexity of DPID0 is $\mathcal{O}(n\ell + n^2\lambda)$.

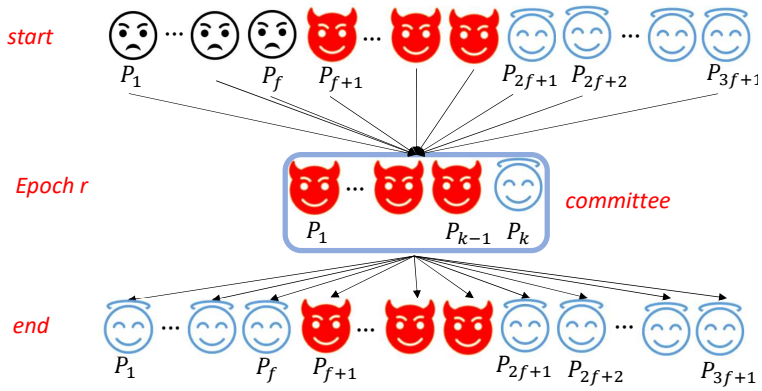

FIGURE 5.1. The very high level of corrupted nodes in DPID1

High-Level Overview. In comparison to DPID0, the main difference of DPID1 is that it considers using the same technology as [**25**] to select an honest majority sub-committee from the new committee, where the size of the sub-committee is κ. Here the sub-committee size κ is proportional to the new committee size of N. And at least $\lceil \frac{\kappa}{2} \rceil + 1$ of the subcommittee is honest except with negligible probability. More precisely, the probability that this condition fails is *negligible* in κ, meaning that for any polynomial $p(\cdot)$ there exists N such that for all $\kappa > N$,

$$\Pr[\text{fewer than } \lceil \kappa/2 \rceil + 1 \text{ honest}] < \frac{1}{p(\kappa)}.$$

This way, we can reduce the $n\ell$-term to $\kappa\ell$-term, achieving a data size term proportional to $\mathcal{O}(n)$.

Constructing the DPID1. The process of DPID1 protocol is the same as the DPID0 except the refresh phase Algorithm 3, the formal description of DPID1 is shown in Algorithm 1, 2, 4 and 5. We describe the Algorithm 5 for a different refresh phase protocol.

Specifically, we consider the old committee members send a valid $block[e-1]$ to these subcommittee members instead of the whole new committee members at the start of the refresh phase. Once the members of the sub-committee retrieve f_{e-1} + 1 fragments that are valid and consistent with

Algorithm 5 Asynchronous DPID1 with epoch $e > 1$: Refresh sub-protocol for party $\mathcal{P}_i$: with subcommittee

Initialization: $T \leftarrow \{\}$; $T_e \leftarrow \{\}$; $flag \leftarrow 0$; Public input: $n_{e-1}, f_{e-1}, n_e, f_e$;

▷ choose a sub committee $Sub.C_e$ from new committee C_e

Invoke Committee Election protocol $\mathsf{CE}(e)$;
wait until $Sub.C_e := \{\mathcal{P}_{j_1}, \mathcal{P}_{j_2}, \cdots, \mathcal{P}_{j_\kappa}\} \leftarrow \mathsf{CE}(e)$
if $\mathcal{P}_i \in C_{e-1}$ **then** ▷ old committee C_{e-1}
  if $\mathsf{Verify}(i, block[e-1]) = 1$ **then**
    parse $block[e-1] := (m_i, \mathbf{D}_{e-1}, T_{e-1})$;
    multicast $\mathsf{FRESH}(e, m_i, D_{e-1})$ to $Sub.C_e$;
if $\mathcal{P}_i \in Sub.C_e$ **then** ▷ sub committee $Sub.C_e$
  upon receiving $\mathsf{FRESH}(e, m_j, D_{e-1})$ from $\mathcal{P}_j (\in C_{e-1})$ for the first time **do**
    $T = T \cup \{(j, m_j)\}$
  wait until f_{e-1} + 1 identical $\mathbf{D}_{e-1}$ have been received from distinct C_e nodes
    then parse $\mathbf{D}_{e-1} := \{h_1, \cdots, h_{n_{e-1}}\}$;
    for any $(j, m_j) \in T$ **do**
      if $\mathcal{H}(m_j) \neq h_j$ **then** discard (j, m_j) from T;
    upon $|T| = f_{e-1} + 1$ **do**
      $M' \leftarrow \mathsf{Dec}(T)$; **let** $\{\overline{m_j}\}_{j \in [n_e]} := \mathsf{Enc}(M', n_e, f_e)$;
      let $\mathbf{D}_e := \{\mathcal{H}(\overline{m_1}), \ldots, \mathcal{H}(\overline{m_{n_e}})\}$;
      send $\mathsf{Relay}(\overline{m_j}, \mathbf{D}_e)$ to $\mathcal{P}_j$ for any $\mathcal{P}_j \in C_e$;
if $\mathcal{P}_i \in C_e$ **then** ▷ new committee C_e
  wait until $\lceil \frac{\kappa}{2} \rceil$+1 identical $(\overline{m_i}, \mathbf{D}_e)$ have been received from distinct subcommittee members
    if $h'_i = \mathcal{H}(\overline{m_i})$ **then** $\sigma_i \leftarrow \mathsf{DS.Sig}(e, \mathcal{H}(\mathbf{D}_e))$;
      multicast $\mathsf{Final}(e, \sigma_i)$ to C_e;
      upon receiving $\mathsf{Final}(e, \sigma_j)$ from $\mathcal{P}_j (\in C_e)$ for the first time **do**
        if $\mathsf{Verify}(e, \mathcal{H}(\mathbf{D}_e), (j, \sigma_j))$ =1 **then** $T_e \leftarrow T_e \cup \{j, \sigma_j\}$;
  upon $|T_e| = f_e + 1$ **do**
    return $block[e] := (\overline{m_i}, \mathbf{D}_e, T_e)$.

the auxiliary information $\mathbf{D}_{e-1}$, then each subcommittee member decodes the fragments to recover the original message and encodes it with new parameter (n_e, f_e), and then sends new fragment m_i along with new auxiliary information $\mathbf{D}_e$ to the corresponding node $\mathcal{P}_i$.

Given that we assume an honest majority within the subcommittee, each node can produce an output block as soon as it receives the same message m_i and the corresponding auxiliary information $\mathcal{D}_e$ from at least $\lceil \frac{\kappa}{2} \rceil + 1$ subcommittee members.

Suppose that some corrupted nodes attempt to deviate by using an alternative dataset M' (with $M' \neq M$) to encode new fragments and generate the

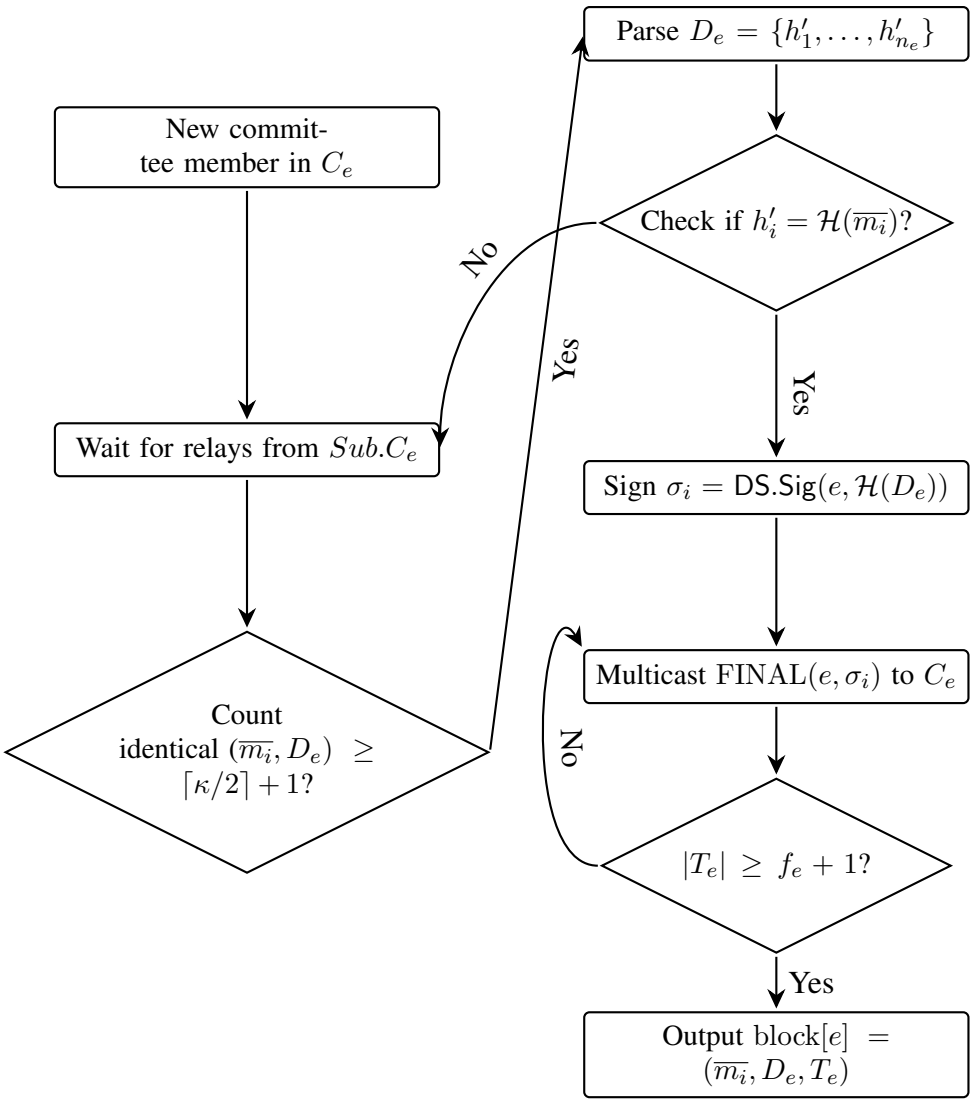

FIGURE 5.2. DPID1 Refresh — New Committee

auxiliary information $\mathcal{D}'_e$. Although an honest node may not be able to independently verify whether a fragment originates from the original data M, it is constrained by the protocol to send out only one message. Consequently, the corrupted nodes cannot collect the necessary $\left\lceil \frac{\kappa}{2} \right\rceil + 1$ shares for M' to convince a new node to accept a fragment that does not correspond to the original dataset.

Thus, the protocol ensures that even if corrupted nodes try to introduce alternative data, they will fail to gather sufficient consensus from the honest

majority, preventing any incorrect fragments from being accepted. The detailed protocol proceeds as follows:

- Refresh after epoch 1: (Algorithm 5). First, the new committee members run the committee election (CE) protocol to agree on the subcommittee for the new epoch. After reaching a consensus on the subcommittee $Sub.C_e$, they send refresh messages to all nodes in both old and new committees. Concurrently, the old committee members continue to listen to the committee-election protocol messages. Once the old committee members receive the output $Sub.C_e$ from the committee-election subprotocol, then each node $\mathcal{P}_i$ of the old committee first checks whether the local $block[e-1] := \{m_i, \mathbf{D}_{e-1}, T_{e-1}\}$ is valid or not. If the local block is valid, it will invoke multicast and send its stored fragment and auxiliary information to all elected nodes of subcommittee $Sub.C_e$ using $\mathsf{FRESH}(e, m_i, D_{e-1})$. See the flowchart 5.2 for further details.
- All nodes in $Sub.C_e$ first wait for $f_{e-1}+1$ identical D_{e-1} received from the old committee, then filter out all invalid fragments based on this auxiliary information. Once receiving $f_{e-1}+1$ valid fragments, each member decodes data M' and recomputes new auxiliary information $\mathbf{D}_e$ for the new committee. Then all honest nodes in the subcommittee will send $\mathsf{Relay}(\overline{m_j}, \mathbf{D}_e)$ to each corresponding node $\mathcal{P}_j$ of new committee C_e.
- For any honest node $\mathcal{P}_i$ of the new committee, once receiving $\lceil \frac{\kappa}{2} \rceil + 1$ identical $\mathsf{Relay}(\overline{m_i}, \mathbf{D}_e)$ message from distinct members of subcommittee $Sub.C_e$, it verifies the new message fragment $\overline{m_i}$ inside Relay message by $\mathcal{H}(\overline{m_i}) = \mathbf{D}_e[i]$. Finally, each node signs and broadcasts $\mathcal{H}(\mathbf{D}_e)$ to other nodes in the new committee, similarly to the DPID0 refresh protocol. Once f_e+1 valid signatures on $\mathcal{H}(\mathbf{D}_e)$ have been received, these signatures constitute the witness T_e, allowing the node to output the final block $(\overline{m_i}, \mathbf{D}_e, T_e)$ for epoch e.

Security intuition of DPID1**:** The security intuition of DPID1 is similar to DPID0. The core point is that the honest majority of the elected subcommittee can make sure $\mathbf{D}_e$ is unique, and the two different auxiliary information $\mathbf{D}_{e-1}$ and $\mathbf{D}_e$ are generated by the same input data M. And when the $\mathbf{D}_e$ is fixed and agreed by all honest nodes in the new committee, every single fragment of epoch e would be easy to verify. Algorithms 1, 2, 4 and 5 implement all properties of DPID. Intuition is as follows:

- *Termination.* For epoch $e = 1$, the termination follows immediately from the *validity* and *agreement* properties of the RBC protocol. All honest nodes will have the same $\mathbf{D}_e$ and sign the hash of $\mathbf{D}_e$. Consequently, all honest nodes can wait for $f_1 + 1$ signatures for $\mathbf{D}_e$ to form a witness and then output $block[1]$. At the beginning of epoch 2, as specified in Algorithm 1, it is established that a minimum of $f_1 + 1$ honest nodes possess a valid block$[1]$. Consequently, each honest node within the subcommittee, elected by nodes in epoch 2, can await $f_1 + 1$ valid instances of block$[1]$ during the refresh phase. This ensures that the honest majority of subcommittee nodes are capable of successfully reconstructing and re-encoding the message. Subsequently, these nodes can transmit the relay message to each designated node in the epoch 2 committee. As a result, all honest nodes in the epoch 2 committee will eventually receive at least $\lceil \frac{\kappa}{2} \rceil + 1$ valid message fragments along with D_2, enabling them to output block$[2]$. Similarly, at the start of epoch e, at least $f_{e-1} + 1$ honest nodes have valid $block[e-1]$. Every honest node in epoch e can wait for $\lceil \frac{\kappa}{2} \rceil + 1$ valid message fragment and D_e from the honest nodes in the elected subcommittee of this epoch, ensuring that all honest nodes in the new committee can output $block[e]$.
- *Agreement.* Following the *termination* of DPID, all honest nodes in the new committee can output a $block[e]$ at epoch e. Then, following the validation function, all honest nodes will output a valid $block[e]$.

- *Availability.* If an honest node of the new committee can output a $block[e]$, then, according to the *agreement* of DPID, all honest nodes in the new committee will also be able to output a valid $block[e]$. Therefore, for any node $\mathcal{P}_j$ that wants to retrieve data at any epoch e, it can wait for $f_e + 1$ valid $block[e]$ to eventually reconstruct B'.
- *Correctness.* If an honest node $\mathcal{P}_i$ can output a $block[e]$ containing $\mathbf{D}_e$, then, according to Algorithm 5, at least $\lceil \frac{\kappa}{2} \rceil + 1$ nodes inside the elected subcommittee of epoch e must have sent the same valid $Relay$ with D_e. Now, suppose that another honest node generates a $block'[e]$ with $\mathbf{D}'_e \neq \mathbf{D}_e$. This would imply that at least one honest node in the subcommittee has generated different auxiliary information such that $\mathbf{D}'_e \neq \mathbf{D}_e$, which is clearly impossible as the honest node will only generate one valid D_e. Hence, any two honest nodes must have the same $\mathbf{D}_e$. Following the *availability* property of DPID, any honest client can reconstruct some block B' with the same auxiliary information $\mathbf{D}_e$, ensuring that any two honest clients will eventually reconstruct the same B'. If the dealer was honest and inputted B in epoch 1, then all honest nodes in the new committee will reconstruct the same B, recompute a new $\mathbf{D}_e$ related to B, and hence $B' = B$.
- *Integrity.* If any honest node of C_e can output a value, following the above analysis on the *correctness* of DPID, it also implies that all honest nodes of C_e can reconstruct the same $B' = M'$ (as per line 19 of algorithm 5). Furthermore, we note that all message shares inside the FRESH message were generated from M' via a deterministic encoding algorithm (as shown in line 15 of algorithm 5), meaning that any honest client in epoch $e - 1$ can reconstruct a block $B'' = M'$. Therefore, we have $B' = B''$.

Complexities of DPID1**:** The DPID1's communication complexity is asymptotically $\mathcal{O}(\kappa\ell + \kappa\lambda n^2)$, the cost mainly appears in four parts:

- Committee Election*:* The cost of CE(e) is $\mathcal{O}(n^2\lambda)$.
- FRESH *message:* The size of FRESH message is $\mathcal{O}(\ell/n+n\lambda)$, the total cost of FRESH message is $\mathcal{O}(\kappa\ell + \kappa n^2\lambda)$.
- Relay *message:* The size of Relay is $\mathcal{O}(\ell/n + n\lambda)$, the total cost of Relay message is $\mathcal{O}(\kappa\ell + \kappa n^2\lambda)$.
- Final *message:* The size of Final (signature) is $\mathcal{O}(\lambda)$, the total cost of Final message is $\mathcal{O}(n^2\lambda)$.

In sum, the DPID1's communication complexity is asymptotically $\mathcal{O}(\kappa\ell + \kappa\lambda n^2)$. The total storage cost of DPID1 is the same as DPID0, it is also $\mathcal{O}(\ell + n^2\lambda)$.

CHAPTER 6

DPID2: Asynchronous Dynamic-Committee Proactive Information Dispersal with Optimal Communication in Optimistic Cases

---

6.1 Weak Set Cover

In this section, we first introduce the core component for DPID2 construction, namely the *weak core set* (WSetCover)[1]. This critical component addresses the following challenge: Given any message M, we generate $N = 3F + 1$ fragments using an encoding algorithm such that any $F + 1$ fragments are sufficient to reconstruct M. The task then becomes how to properly distribute these $N = 3F + 1$ distinct fragments among $n = 3f + 1$ nodes, in a way that ensures any $f + 1$ nodes collectively hold at least $F + 1$ distinct fragments. This distribution method guarantees that any set of $f + 1$ nodes can always recover the original message M using $F + 1$ distinct fragments, while maintaining a constant number of fragments per node.

Essentially, the first requirement of the above problem can be categorised as a specific instance of the set cover problem [**6**]. Formally, the definition of the set cover is as follows:

[1] It is to avoid the use of heavy generic NIZK but still enable cheap verifications, in the design of DPID2.

DEFINITION 1 (Set Cover [**6**]). *Given a family $S = \{S_1, S_2, \cdots, S_n\}$ of subsets of a ground set $C = \{c_1, c_2, \cdots, c_N\}$, we assume that $\cup_{S_i \in S} S_i = C$, a set cover of C is a sub-family $S' \subseteq S$ such that $\cup_{S_i \in S'} S_i = C$.*

In our setting, we require that $C = \mathsf{Enc}(M, N, F+1)$ and $\bigcup_{S_i \in S'} S_i = C'$, where $|C'| = F + 1$, and the size of each $|S_i|$ is constant. Compared to the formally defined set cover problem, we refer to our problem as a weak set cover problem. Specifically, we consider the following scenario: Given a family $S = \{S_1, S_2, \ldots, S_n\}$ of subsets of a ground set $C = \{c_1, c_2, \ldots, c_N\}$, where $\bigcup_{S_i \in S} S_i = C$ and $|S_i| = \mathcal{O}(1)$ for any $i \in [n]$, then C' is defined as any sub-family $S' \subseteq S$ such that $\bigcup_{S_i \in S'} S_i = C' \subseteq C$, where $|C'| = F + 1$ and $|S'| = f + 1$. Formally, we define this requirement as follows:

DEFINITION 2 (Weak Set Cover). *In a weak set cover protocol (*$\mathsf{WSetCover}$*) among n nodes, if a designated sender $\mathcal{P}_s$ inputs a message M, the protocol satisfies the following properties:*

- Termination. *If the sender is honest, then there are at most N distinct fragments distributed among honest nodes, and every honest node $\mathcal{P}_i$ can output a valid fragment set S_i.*
- Availability. *Given any $f + 1$ valid fragment sets S_i, the union of these sets contains at least $F + 1$ distinct values, enabling anyone to reconstruct the original message M.*
- Efficiency. *The size of each fragment set S_i is $\mathcal{O}(|M|/n)$.*

The details of $\mathsf{WSetCover}$**.** The formal description of the protocol is provided in Algorithm 6. Suppose that we have N fragments m_1 to m_N, where $N = n \cdot k_1 + k_2$, with $0 \leq k_2 < n$.

For the fragments m_1 to $m_{n \cdot k_1}$, the basic idea is to distribute these $n \cdot k_1$ fragments to n nodes in a round-robin fashion. The first fragment is given

to the first node, the second fragment to the second node, and so on, until the n-th fragment is given to the n-th node. Afterwards, the $(n+1)$-th fragment is given to the first node again, and this pattern continues until all fragments from m_1 to $m_{n \cdot k_1}$ have been distributed. Hence, each node receives k_1 distinct fragments.

If $k_2 > 0$, let $n = k_2 \cdot k_3 + k_4$. Throughout this paper, we assume $k_4 = 0$ (for $k_4 > 0$, we leave it as future work). For the remaining k_2 fragments $m_{n \cdot k_1 + 1}$ to m_N, we assign fragment $m_{n \cdot k_1 + 1}$ to the first node, fragment $m_{n \cdot k_1 + 2}$ to the second node, and so on, until fragment $m_{n \cdot k_1 + k_2}$ is assigned to the k_2-th node. Afterwards, fragment $m_{n \cdot k_1 + 1}$ will be given to the (k_2+1)-th node, and this round-robin pattern continues until every node receives one fragment. This way, we can ensure that every node receives $k_1 + 1$ fragments. Our assignment strategy satisfies the weak set cover property if $N = \mathcal{O}(n)$. Besides, we also add one function VerifyWCS, which can be used to verify whether the received S_i is valid.

Security Intuition of WSetCover**.** We brief the security intuitions of the weak set cover protocol in the following:

- *Terminatation.* If the sender is honest, according to Algorithm 6, the sender invokes the Enc to generate N distinct fragments and distributes these fragments to n nodes. Hence, all honest nodes $\mathcal{P}_i$ can successfully output a valid fragment set S_i.
- *Availability.* For any $f+1$ honest nodes:
  (1) If $k_2 = 0$, the $f+1$ nodes collectively have $(f+1) \cdot k_1$ fragments. Since: $(f+1) \cdot k_1 > \left(\frac{n-1}{3} + 1\right) \cdot k_1 = \left(\frac{n+2}{3}\right) \cdot k_1 > \frac{n \cdot k_1}{3} = \frac{N}{3}$, the availability is satisfied.
  (2) If $k_2 > 0$, for the remaining k_2 fragments $m_{n \cdot k_1 + 1}$ to m_N, there are at most $\frac{f+1}{k_3}$ fragments among these k_2 fragments that are held by the $f+1$ nodes. Therefore, the total number of fragments these

Algorithm 6 The weak set cover (WSetCover): for each node $\mathcal{P}_i$

Initialization: $S_i \leftarrow \{\}$; $H[i] \leftarrow \{\}$; $H'[i] \leftarrow \{\}$
Public parameter: N, F, n, f
let $N = n \cdot k_1 + k_2$, where $0 \leq k_2 < n \leq N$
if $\mathcal{P}_i$ is the sender of $\mathcal{P}_s$ and with M as input **then**
  $\{m_j\}_{j\in[N]} \leftarrow \mathsf{Enc}(M, N, F+1)$
  $\mathbf{D} \leftarrow \{\mathcal{H}(m_1), \ldots, \mathcal{H}(m_N)\}$
  for any $j \leq n \cdot k_1$ **do**
    for any $i \leq n$ **do**
      if $i \mod n = j \mod n$ **then**
        $S_i \leftarrow S_i \cup (j, m_j)$
  if $k_2 > 0$ **then**
    let $n = k_2 \cdot k_3 + k_4$ ▷ in this paper, we assume $k_4 = 0$
    for any $n \cdot k_1 < j \leq N$ **do**
      for any $i \leq n$ **do**
        if $i \mod k_2 = j \mod n$ **then**
          $S_i \leftarrow S_i \cup (j, m_j)$
  for $i \in [n]$ **do**
    sent $\mathsf{WSC}(S_i)$ to $\mathcal{P}_i$
upon receiving $\mathsf{WSC}(S_i)$ from $\mathcal{P}_s$ **do**
  output S_i

function $\mathsf{VerifyWCS}(i, n, N, \mathbf{D}, S_i)$:
  let $N = n \cdot k_1 + k_2$, where $0 \leq k_2 < n \leq N$
  for any $(j, m_j) \in S_i$ **do**
    $H'[i] \leftarrow H'[i] \cup \mathcal{H}(m_j)$
  for any $j \leq n \cdot k_1$ **do**
    for any $i \leq n$ **do**
      if $i \mod n = j \mod n$ **then**
        $H[i] \leftarrow H[i] \cup \mathbf{D}[j]$
  if $k_2 > 0$ **then**
    let $n = k_2 \cdot k_3 + k_4$ ▷ $k_4 = 0$
    for any $j > n \cdot k_1$ **do**
      for any $i \leq n$ **do**
        if $i \mod k_2 = j \mod n$ **then**
          $H[i] \leftarrow H[i] \cup \mathbf{D}[j]$
  if $H[i] = H'[i]$ **then** return 1
  else return 0

nodes possess is: $(f+1) \cdot k_1 + \frac{f+1}{k_3}$. Substituting $\frac{f+1}{k_3} = \frac{k_2}{\frac{n}{f+1}}$, we have: $(f+1) \cdot k_1 + \frac{k_2}{\frac{n}{f+1}} > \left(\frac{n+2}{3}\right) \cdot k_1 + \frac{k_2}{3} > \frac{n \cdot k_1 + k_2}{3} = \frac{N}{3}$.

Hence, in any case, the $f+1$ nodes can always collectively hold at least $F+1$ fragments.

- *Efficiency.* Given that $N = \mathcal{O}(n)$, we have $k_1 = \mathcal{O}(1)$. As a result, each node receives at most $k_1 + 1 = \mathcal{O}(1)$ fragments. Considering that the size of each fragment is $\mathcal{O}(|M|/n)$, the size of each fragment set

S_i is therefore $\mathcal{O}(|M|/n)$, which ensures that the number of fragments received by each node is independent of $\mathcal{O}(n)$.

6.2 Concrete Implementation of DPID2

For information dispersal, it is established that the lower bound on communication complexity for storing data (or files) is $\Omega(\ell + n^2)$, and the lower bound on storage cost is $\Omega(\ell)$, where ℓ denotes the data size [**2**]. In DPID0 and DPID1, when $\ell > n^2\lambda$, the storage complexity remains $\mathcal{O}(\ell)$; however, the communication complexity during the refresh phase becomes $\mathcal{O}(n\ell)$ or $\mathcal{O}(\kappa\ell)$. This naturally raises the question of whether a new protocol can be designed with optimal communication complexity $\mathcal{O}(\ell)$, or one that outperforms DPID1 in communication efficiency when the data size is sufficiently large.

This section considers the optimistic case where all nodes are honest and the network is synchronous. We introduce DPID2, which achieves optimal communication complexity under these conditions while maintaining the same security guarantees and worst-case costs as DPID0 and DPID1.

Overview of DPID2**.** Recall the DPID0 and DPID1 protocols: in the Refresh phase, all old nodes either multicast their fragments directly to all new nodes (as in DPID0), or multicast their fragments to a sub-committee, where sub-committee members subsequently reply with the corresponding fragments to the new nodes (as in DPID1). Considering the condition of the optimistic case, we further optimise DPID1 by reducing the subcommittee size to 1, meaning only one node acts as the intermediary. Based on this idea, DPID2 operates as follows: all old nodes in C_{e-1} send their $block[e-1]$ to a designated node $\mathcal{P}_e \in C_e$. Node $\mathcal{P}_e$ performs the same role as a subcommittee member in DPID1. If all nodes receive their corresponding information within a fixed time, they output $block[e]$.

Otherwise, if any node fails to receive the output by the end of this period, it will send a complaint message to the network. Upon receiving such a complaint, any honest node will forward it to all other nodes, prompting all nodes to fall back to the DPID1 protocol. The node will then switch to the DPID1 protocol to handle the situation. This section focuses solely on the optimistic case and ignores the fallback processes. For completeness, we also provide the dealer dispersal in epoch 1 (Algorithm 9) and the retrieval process at any epoch (Algorithm 10). It is worth noting that although Byzantine fault injection attacks do not compromise system integrity, they can trigger a fallback to the DPID1 protocol, resulting in degraded performance.

In DPID, where the sizes of the new and old committees may differ, we encounter a critical challenge: a malicious node $\mathcal{P}_e$ could compromise the system's security by sending inconsistent fragments to the new committee nodes, such as inputting different data instead of original data when invoking Enc. To address this issue, it becomes crucial to verify the correctness of the newly generated fragments.

One potential solution is to employ NIZK or SNARK proofs to ensure the fragments derived from the original data. When the leader tries to encode the original message into new message fragments, it needs to generate a valid NIZK proof to prove that the latest messages originated from the exact same message as in the previous epoch. However, as we mentioned earlier, this approach could be inefficient. To achieve the $\mathcal{O}(\ell)$ communication term, the size of the fragments each node receives must be at most $\mathcal{O}(\ell/n)$.

To address this issue, we consider a fixed total number of fragments $N = 3F + 1$, where the original message is encoded into N fragments and an associated fixed auxiliary information $\mathbf{D}$ is generated. Additionally, once the

fragments are encoded in the first epoch, both the number of distinct fragments and the fragments themselves remain unchanged throughout the subsequent epochs. At the end of each epoch, we ensure that each honest node only needs to receive a subset of the N original fragments, where the size of this subset remains constant ($\mathcal{O}(1)$). This design reduces the total number of messages in the intermediary phase from κn to n, thereby achieving optimal communication complexity in the optimistic case (when all nodes are honest and the network is synchronous). In the presence of dishonest nodes within the committee, the communication complexity matches that of DPID1.

Consider the following example to illustrate the design of DPID2. A message is encoded into $N = 12$ fixed fragments and distributed among the nodes based on the committee size. When the committee size is $n = 4$, each node receives three fragments (e.g., m_1, m_2, m_3 for the first node). If the committee size increases to $n = 6$, each node receives two fragments (e.g., m_1, m_2 for the first node). The total number of fragments remains constant ($N = 12$), ensuring scalability and maintaining optimal communication complexity as the committee size varies.

To transfer the data to the new committee from the old committee without blowing up the communication complexity, we first let every honest node of the old committee send its fragment and auxiliary information to an elected node in the new committee. That elected node will then decode these fragments and get the original message. The next step is to deterministically encode the message into the same N fragments as in epoch 1 and send fragments alongside auxiliary information to designated nodes in the new committee.

Construction Details of DPID2**.** Now we describe our protocol DPID2 for the case where $N \geq n_e$, where $e > 1$. The formal description of the

Algorithm 7 The optimistic case of DPID2 Refresh, with epoch $e > 1$ for node $\mathcal{P}_i$

Initialization: $T_1 \leftarrow \{\}$; $T_2 \leftarrow \{\}$; $T_e \leftarrow \{\}$
Public parameter: $N, F, n_{e-1}, f_{e-1}, n_e, f_e$
Let $s \leftarrow (e \mod n) + 1$

if $\mathcal{P}_i \in C_{e-1}$ **then** ▷ old committee C_{e-1}
  if $\mathsf{Verify}(i, block[e-1]) = 1$ **then**
    parse $block[e-1] := (S_i, \mathbf{D}, T_{e-1})$
    $\mathsf{ADD}[e](\mathbf{D})$
    multicast $\mathsf{FRESH}(e, S_i)$ to $\mathcal{P}_s (\in C_e)$
if $\mathcal{P}_i = \mathcal{P}_s$ and $\mathcal{P}_i \in C_e$ **then** ▷ intermediary
  upon receiving $\mathsf{FRESH}(e, S_j)$ from $\mathcal{P}_j (\in C_{e-1})$ **do**
    $T_1 \leftarrow T_1 \cup (j, S_j)$
  wait until $\mathsf{ADD}[e]$ return $\mathbf{D}$
    for any $(j, S_j) \in T_1$ **do**
      if $\mathsf{VerifyWCS}(j, n_{e-1}, N, \mathbf{D}, S_j) = 1$ **then**
        $T_2 \leftarrow T_2 \cup S_j$
    upon $|T_2| \geq F + 1$ **do**
      $M' \leftarrow \mathsf{Dec}(T)$
      invoke WSetCover with M' and public parameter N, F, n_e, f_e
if $\mathcal{P}_i \in C_e$ **then** ▷ new committee C_e
  upon receiving output S_i from WSetCover **do**
    wait until $\mathsf{ADD}[e]$ return $\mathbf{D}$
      if $\mathsf{VerifyWCS}(i, n_e, N, \mathbf{D}, S_i) = 1$ **then**
        $\sigma_i \leftarrow \mathsf{DS.Sig}(e, \mathcal{H}(\mathbf{D}))$
        multicast $\mathsf{FINAL}(e, \sigma_i)$ to C_e
        upon receiving $\mathsf{FINAL}(e, \sigma_j)$ from $\mathcal{P}_j \in C_e$ **do**
          if $\mathsf{DS.Vrf}(e, \mathcal{H}(\mathbf{D}), (j, \sigma_j)) = 1$ **then**
            $T_e \leftarrow T_e \cup \{(j, \sigma_j)\}$
  upon $|T_e| = f_e + 1$ **do**
    $block[e] \leftarrow (S_i, \mathbf{D}, T_e)$, and output $block[e]$

Algorithm 8 Validation function of DPID2, with epoch e

Public parameter: N, F, n_e, f_e
function $\mathsf{Verify}(i, block[e])$
  if $block[e] \neq \emptyset$ **then** parse $block[e] := (S_i, \mathbf{D}, T_e)$
    if $|T_e| = f_e + 1$ and $\mathsf{VerifyWCS}(i, n_e, N, \mathbf{D}, S_i) = 1$ **then**
      if $\mathsf{DS.Vrf}(e, \mathcal{H}(\mathbf{D}), (j, \sigma_j)) = 1$ for any $(j, \sigma_j) \in T_e$ **then**
        return 1
  return 0

protocol is provided in Algorithms 7, 8, 9 and 10. For the Refresh phase (Algorithm 7), the details proceed as follows:

- Phase 1: For all honest nodes $\mathcal{P}_i$ in the old committee: each node first verifies the validity of its local $block[e-1]$. If valid, the node invokes

Algorithm 9 The DPID2 dispersal (ACVID), with epoch $e = 1$ and sender $\mathcal{P}_s$ for node $\mathcal{P}_i$

Initialisation: Let RBC$[e]$ refer to one instance of the reliable broadcast protocol, where $\mathcal{P}_s$ is the sender of RBC$[e]$; $T_e \leftarrow \{\}$
Public parameter: N, F, n_1, f_1

1 **if** $\mathcal{P}_i = \mathcal{P}_s$ and received input M **then** ▷ for the sender
2 invoke RBC$[e]$ with input M
3 invoke WSetCover with input M and public parameter N, F, n_1, f_1
4 **upon** receiving output M from RBC$[e]$ **do**
5 **if** $M \neq \perp$ **then**
6 $\{m_j\}_{j\in[N]} \leftarrow \mathsf{Enc}(M, N, F)$
7 $\mathbf{D} \leftarrow \{\mathcal{H}(m_1), \ldots, \mathcal{H}(m_N)\}$
8 **upon** receiving output S_i from WSetCover **do**
9 **if** VerifyWCS$(i, n_e, N, \mathbf{D}, S_i) = 1$ **then**
10 $\sigma_i \leftarrow \mathsf{DS.Sig}(e, \mathcal{H}(\mathbf{D}))$
11 **multicast** FINAL(e, σ_i) to C_e
12 **upon** receiving FINAL(e, σ_j) from $\mathcal{P}_j \in C_e$ **do**
13 **if** DS.Vrf$(e, \mathcal{H}(\mathbf{D}), (j, \sigma_j)) = 1$ **then**
14 $T_e \leftarrow T_e \cup \{(j, \sigma_j)\}$
15 **upon** $|T_e| = f_e + 1$ **do**
16 $block[e] \leftarrow (S_i, \mathbf{D}, T_e)$, and output $block[e]$

ADD to share the auxiliary information **D** with the new committee and sends its fragment set to the new leader $\mathcal{P}_s$ via FRESH message.

- Phase 2: For the leader of the new committee: he waits for ADD to output the auxiliary information **D**, which is then used to filter out invalid FRESH messages. If the leader receives at least $f_{e-1} + 1$ valid FRESH messages, ensuring $T_2 \geq F + 1$, the leader decodes a message M' and uses it as input to the WSetCover protocol to distribute the N fragments of M' to the n_e nodes.
- Phase 3: For all honest nodes in the new committee: once ADD outputs the auxiliary information **D** and WSetCover returns the fragment set S_i, each node verifies whether the leader $\mathcal{P}_s$ has provided the correct input to WSetCover. If the verification is successful, the node signs $\mathcal{H}(\mathbf{D})$ and multicasts the signature. Finally, upon receiving $f_e + 1$ valid signatures on $\mathcal{H}(\mathbf{D})$ to form T_e, the node outputs $block[e] = (S_i, \mathbf{D}, T_e)$ for epoch e.

Security Intuition of DPID2**:** The security intuition of DPID2 is similar to that of DPID1. The key idea is to use fixed encoded fragments across all

Algorithm 10 The DPID2 retrieval, with epoch e for node $\mathcal{P}_i$

Initialization: $T \leftarrow \{\}$
Public parameter: N, F, n_e, f_e

if $\mathcal{P}_i$ would like to retrieve the original value **then**
  multicast RETRIEVAL(e) to C_e;
upon receiving RETRIEVAL(e) from $\mathcal{P}_j$ and $\mathcal{P}_i \in C_e$ **do**
  if Verify$(i, block[e])$=1 **then**
    send RECAST$(e, block[e])$ to $\mathcal{P}_j$;
upon receiving RECAST$(e, block[e])$ from $\mathcal{P}_j (\in C_e)$ **do**
  if Verify$(j, block[e])$=1 **then**
    parse $block[e] := (S_i, \mathbf{D}, T_e)$
    $T \leftarrow T \cup S_i$
upon $|T| \geq F + 1$ **do**
  $M' \leftarrow$ Dec(T), and output M'

epochs, ensuring the auxiliary information **D** remains unchanged. The only difference lies in the number of fragments held by honest nodes, which may vary depending on the committee size. The WSetCover protocol that we introduced in the previous section guarantees the availability of the original message, ensuring that it can always be reconstructed.

- *Termination.* For epoch $e = 1$, the termination follows immediately from the *validity* and *agreement* properties of the RBC protocol. All honest nodes will have the same $\mathbf{D}_e$ and sign the hash of $\mathbf{D}_e$. Consequently, all honest nodes can wait for $f_1 + 1$ signatures for $\mathbf{D}_e$ to form a witness and then output $block[1]$.

  At the start of epoch 2, according to Algorithm 1, we know that at least $f_1 + 1$ honest nodes have valid $block[1]$. Therefore, the selected leader in epoch 2 can wait for $f_1 + 1$ valid $block[1]$ during the refresh phase. And according to the WSetCover protocol, the epoch 2 leader can expect to receive $F + 1$ valid message fragments from valid $block[1]$ to decode the original message. Then the leader can encode the message again using the same public parameters N, F and auxiliary information D. Finally, all honest nodes in epoch 2 can wait for valid fragments from the leader or fallback to DPID1 to output valid $block[2]$.

Similarly, at the start of epoch e, at least $f_{e-1}+1$ honest nodes have valid $block[e-1]$. The epoch leader in epoch e can wait for $f_{e-1}+1$ valid $block[e-1]$ from the previous committee to get $F+1$ message fragments and recover the original message. And then the leaders in all epochs can always encode the original message again using the exact same public parameters N, F and auxiliary information D, ensuring that all honest nodes in the new committee can output a valid $block[e]$.

- *Agreement.* Following the *termination* of DPID, all honest nodes in the new committee can output a $block[e]$ at epoch e. Then, following the validation function, all honest nodes will output a valid $block[e]$.

- *Availability.* If an honest node of the new committee can output a $block[e]$, then, according to the *agreement* of DPID, all honest nodes in the new committee will also be able to output a valid $block[e]$. Therefore, for any node $\mathcal{P}_j$ that wants to retrieve data at any epoch e, it can wait for f_e+1 valid $block[e]$. And based on the WSetCover protocol, the node can expect to get at least $F+1$ valid message fragments from the blocks received to reconstruct B' eventually.

- *Correctness.* If an honest node $\mathcal{P}_i$ can output a $block[e]$ containing D, then, according to Algorithm 7, at least f_e+1 nodes inside the new committee of epoch e must have sent the same valid $Final$ message with D. Now, suppose that another honest node generates a $block'[e]$ with $D' \neq D$. This would imply that at least one honest node in the old committee has generated different auxiliary information such that $D' \neq D$, which is clearly impossible as the honest node will only generate one valid D and any different D' will be detected immediately and fallback to DPID1. Hence, any two honest nodes must have the same D.

  Following the *availability* property of DPID, any honest client can reconstruct some block B' with the same auxiliary information **D**, ensuring that any two honest clients will eventually reconstruct the same B'.

If the dealer was honest and inputted B in epoch 1, then all honest nodes in the new committee will reconstruct the same B, carrying the same D related to B in all epochs, and hence $B' = B$.

- *Integrity.* If any honest node of C_e can output a value, following the above analysis on the *correctness* of DPID, it also implies that all honest nodes of C_e can reconstruct the same $B' = M'$ (as per line 14 of algorithm 7). Furthermore, we note that all message shares inside the FRESH message were generated from M' via a deterministic encoding algorithm (as shown in line 3 of algorithm 9), meaning that any honest client in epoch $e-1$ can reconstruct a block $B'' = M'$. Therefore, we have $B' = B''$.

Complexities of DPID2**:** From WSetCover, we know the size of S_i is $\mathcal{O}(\ell/n)$, where ℓ is the input size. The communication complexity of DPID2 in the optimistic case is asymptotically $\mathcal{O}(\ell + n^2\lambda)$, and the cost is mainly distributed across the following five parts:

- ADD*:* The cost of ADD is $\mathcal{O}(n|m| + n^2\lambda)$. In DPID2, the size of m is $n\lambda$, so the cost of ADD becomes $\mathcal{O}(n^2\lambda)$.
- FRESH *message:* The size of each FRESH message is $\mathcal{O}(\ell/n + \lambda)$, leading to a total cost of $\mathcal{O}(\ell + n\lambda)$.
- Relay *phase (*WSetCover*):* In WSetCover, the sender distributes S_i to $\mathcal{P}_i$ for all $i \in [n_e]$. Hence, the total cost of WSetCover messages is $\mathcal{O}(\ell + n\lambda)$.
- FINAL *message:* The size of the FINAL message (signature) is $\mathcal{O}(\lambda)$. Therefore, the total cost for FINAL messages is $\mathcal{O}(n^2\lambda)$.

In summary: (1) Optimistic case: All honest nodes successfully output $block[e]$, resulting in a communication complexity of $\mathcal{O}(\ell + n^2\lambda)$. (2) Worst case: If some nodes fail to output, all nodes revert to executing DPID1, where the communication complexity becomes $\mathcal{O}(\kappa\ell + \kappa n^2\lambda)$.

General Case of DPID2**:** If $N < n_e$, we handle this by ensuring that $n_e/3^k \leq N < n_e/3^{k-1}$, where k is an integer. Specifically, we divide the n_e nodes evenly into 3^k groups. For each group, we distribute the N fragments among its members by invoking DPID2 within the group. Since $N = \mathcal{O}(n_e)$, it follows that $k = \mathcal{O}(1)$. At the start of the next epoch, we can ensure that at least one group can reconstruct the original message. This is guaranteed because, within each group, at least one-third of the nodes are honest. Thus, the protocol maintains correctness and ensures the original message can be recast in subsequent epochs.

CHAPTER 7

Application to Dynamic-Committee Proactive Secret Sharing

Dynamic-committee Proactive secret share (DPSS) has made significant progress, and its important application is to provide confidentiality in BFT SMR or BFT storage. However, the prior works consider directly applying DPSS to store the data such that the cost of Refresh up to $\mathcal{O}(n^3\ell + n^3\lambda)$ [**43**, **46**]. Clearly, these works are sub-optimal for large data.

In this section, we give a simple and generic compiler from DPID to DPSS and plug in our DPID by any state-of-the-art DPSS to get a better DPSS protocol. The asymptotic performance of the resulting protocol is summarised in Table 1.1. In each epoch, all nodes execute the DPID and DPSS protocols in parallel: the input to DPID is the encrypted data, while the input to DPSS is the corresponding encryption key. This approach significantly reduces the overall communication cost.

In our approach, we utilise hybrid encryption to optimise the performance of DPSS, mainly for handling large data sizes. The main principle behind hybrid encryption is to combine the strengths of DPSS and DPID: secure but heavy (in our case, DPSS) for key sharing and key committing symmetric encryption for large-sized input data (in our case, DPID). This vital characteristic makes it ideal for scenarios where large volumes of data must be securely dispersed and refreshed across multiple epochs without incurring significant communication or computation costs.

To elaborate, the DPSS protocol is employed to share and refresh a small, fixed-size secret key securely, typically 32 bytes in length. The use of DPSS to share only the small secret key, rather than the full input data, drastically reduces the $\mathcal{O}(n^3\ell + n^3\lambda)$ complexity associated with directly applying DPSS to large data. The whole process typically takes a few seconds, regardless of the input data size. This way, the asymptotic costs are reduced, as the communication overhead associated with data refresh operations is minimised since only the key needs to be refreshed. Symmetric encryption is advantageous for encrypting large amounts of data, and the size of the resulting ciphertext is nearly identical to the original plaintext. It is ideal for scenarios where large volumes of data must be securely dispersed and refreshed across multiple epochs.

The generated key is then used as the key for AES-GCM, which is known for its efficiency in both encryption and authentication. More importantly, it is key committing [**23**] that will ensure that the value after decryption remains consistent, to guarantee the correctness property of DPSS, as discussed in the Introduction. The 32-byte key is protected by the secrecy property of DPSS, ensuring its confidentiality even in the presence of Byzantine failures or active adversaries. Meanwhile, the data itself is protected by the symmetric encryption scheme, where any tampering with the ciphertext can be detected due to the authentication properties of the chosen encryption mode, and the adversaries learn no information about the plaintext based on the ciphertext. Specifically, if there is an inconsistency between the key and ciphertext, the decryption would simply return $\perp$. Any two distinct clients will always extract the same value when receiving the same secret key and encrypted data from DPSS and DPID, respectively. For simplicity, we omit the key committing process in Algorithm 11.

DPSS further ensures confidentiality under the mobile adversary model, where the adversary may corrupt different subsets of nodes across epochs.

This is achieved by periodic Refresh operations. Even if the adversary compromises up to t parties in one epoch, the refreshed shares in the next epoch are statistically independent of the previous compromised nodes. Hence, long-term secrecy of the initial secret key is maintained, provided that at most t nodes are corrupted in any single epoch. This rolling protection mechanism makes DPSS particularly well-suited for systems requiring strong confidentiality.

Finally, while the current algorithm construction assumes classical hardness assumptions, post-quantum considerations are becoming increasingly important. Replacing AES-GCM with a post-quantum secure symmetric cipher incurs negligible cost. More critically, the DPSS layer relies on public-key cryptographic primitives for verifiable secret sharing, which can be instantiated with lattice-based commitments or zero-knowledge proofs resistant to quantum attacks. This substitution would preserve the compiler framework while slightly increasing the concrete communication costs. And the relative efficiency advantage of our approach over direct DPSS remains even in a post-quantum setting.

In the future, an attractive application domain is blockchain or distributed ledger systems. Here, DPSS can serve as a robust building block for confidentiality in state machine replication, threshold signing, or on-chain storage. In these environments, dynamic nodes may join or leave so that our dynamic-committee DPSS integrates naturally. By combining hybrid encryption with DPSS, large state snapshots can be encrypted and dispersed among committee members, while only small keys require proactive refreshing. This reduces on-chain overhead, amortises costs over epochs, and ensures forward secrecy of the ledger's confidential data.

Algorithm 11 Generic Compiler from DPID to DPSS for party $\mathcal{P}_i$

Initialisation : Let DPID refer to one instance of the dynamic-committee proactive information dispersal protocol, and DPSS represent an instance of the dynamic-committee proactive secret share. Let M denote the plaintext data and sk the secret key.

if $e = 1$ and $\mathcal{P}_i = \mathcal{P}_\mathsf{s}$ **and** received inputs M and sk **then**
  $M' \leftarrow Encrypt(M, sk)$
  invoke DPID with input M' and DPSS with input sk
if $e > 1$ and $\mathcal{P}_i \in C_{e-1}$ **then**
  parse $block[e-1] := (block_{dpid}[e-1], block_{dpss}[e-1])$
  invoke DPID with input $block_{dpid}[e-1]$
  invoke DPSS with input $block_{dpss}[e-1]$
upon receiving output $block_{dpid}[e]$ from DPID$[e]$ and $block_{dpss}[e]$ from DPSS$[e]$ **do**
  $block[e] \leftarrow (block_{dpid}[e], block_{dpss}[e])$; and output $block[e]$
upon receiving Retrieval[e] **do**
  $M' \leftarrow$ invoke $\mathsf{DPID}_{retrieval}[e]$
  $K \leftarrow$ invoke $\mathsf{DPSS}_{retrieval}[e]$
  $M \leftarrow Decrypt(M', sk)$ and output M

CHAPTER 8

Implementation and Evaluation

Implementation Details. We implemented our DPID protocols, DPSS0 and DPSS2, in Python, along with Longlive DPSS [**46**]. For encryption, we used AES-GCM from the PyCryptodome library. The DPSS component of the secret key in our hybrid construction was based on the Low-threshold DPSS implementation from Longlive DPSS. All three algorithms were deployed within a Docker image and evaluated on multiple distinct EC2 t2.micro AWS instances. Each instance hosted a single node to emulate a realistic network-latency environment. In the Refresh phase, we considered $n_e = n_{e-1} = n$.

Latency Comparison. We evaluate the latency of one Refresh phase under the optimistic case where all nodes are honest. Experimental results demonstrate that our DPSS protocols significantly outperform the LongLive DPSS scheme, as illustrated in Figures 8.1 and 8.2. This latency improvement directly translates to stronger scalability. As the number of participants or the reconstruction threshold grows, the communication savings in our design ensure that performance degradation remains modest. Owing to the communication savings inherent in our design, this performance gap is expected to widen further under realistic network conditions with non-negligible delays. Also, since our DPSS protocols minimise both per-node bandwidth requirements and the number of interactive rounds, they sustain higher refresh throughput even under moderate churn.

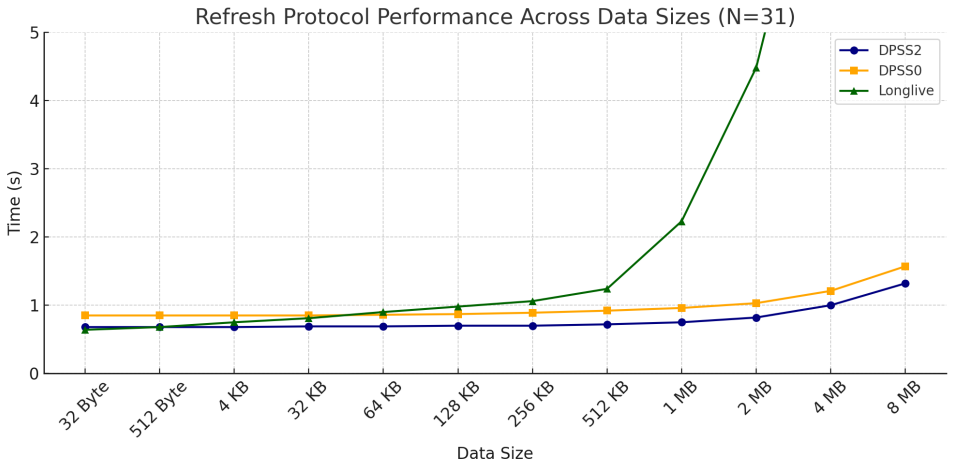

FIGURE 8.1. Latency comparison in the optimistic case

Case 1: Performance analysis with increasing data size and fixed n.

The performance evaluation of LongLive DPSS and its variants gives some interesting insights into how they handle computational efficiency with different data block sizes. The original LongLive DPSS exhibits a sharp latency rise, rising from under 1 s for payloads up to 64 KB to more than 2 s at 1 MB and eventually exceeding the 5 s mark by 8 MB (see Figures 8.1). This dramatic inflection reflects the cubic growth in per-block processing cost inherent to LongLive DPSS. By contrast, both DPID0 and DPID2 grow much more gracefully. Over the entire 32 Byte to 8 MB range, DPID2's latency increases only from roughly 0.68 s to 1.32 s, such modest slopes indicate near-linear scaling with data size, suggesting that their optimised encoding and share-distribution routines impose predominantly $O(n)$ overhead per block rather than the $O(n^3)$ behaviour that plagues LongLive DPSS. Moreover, the tight clustering of the DPID0 and DPID2 curves up to 512 KB, where all three schemes perform comparably, highlights that, for small-to-medium blocks, network and I/O latency dominate. Only beyond this threshold do algorithmic differences emerge. In sum, under fixed n, DPID2 and DPID0 offer substantially better computational efficiency on large data blocks, whereas LongLive DPSS remains viable only for relatively small payloads.

This analysis points out how important optimisations are for computational systems. While the original LongLive DPSS is aggressive in its scaling, DPID0 and DPID2 bring in refinements that make performance more manageable. The differences between these implementations suggest that careful design choices and optimisations can have a big impact, especially for systems that need reliable performance across varying block sizes.

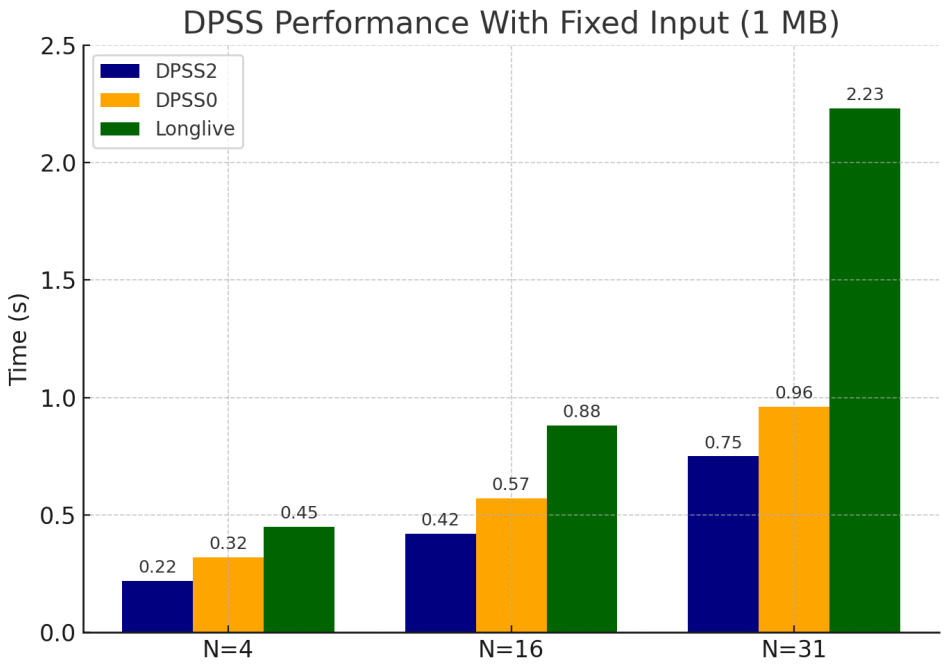

FIGURE 8.2. Latency comparison with fixed input $\ell = 1$ MB

Case 2: Performance analysis with fixed data size and increasing n.

The Refresh phase performance results for a fixed 1MB payload (Figure 8.2) align closely with the theoretical communication complexities. As n grows from 4 to 16 to 31, LongLive DPSS's latency climbs steeply from 0.45 s to 0.88 s to 2.23 s, reflecting its $O(n^3\ell + \lambda n^3)$ behaviour. By comparison, DPID0 rises moderately from 0.32 s to 0.57 s to 0.96 s, consistent with its $O(n\ell + \lambda n^3)$ complexity, and DPID2 shows the gentlest increase, from 0.18 s to 0.42 s to 0.75 s, in line with its dominant $O(\ell)$ term (the λn^3 component remains insignificant at these scales). Thus, under fixed data size, DPID2

exhibits the greatest resilience to increasing n, DPID0 scales acceptably, and LongLive DPSS incurs sharply rising overheads as the network grows.

These findings suggest that the scalability of LongLive DPSS is significantly limited as n increases. In comparison, DPSS0 and DPSS2 exhibit more consistent scalability with a fixed data size. Furthermore, it is important to note that the benefits of our communication complexity reduction may not be fully realised in a local setting. When deployed in a distributed environment, our DPSS0 and DPSS2 are expected to exhibit even greater performance, demonstrating their scalability and efficiency on a larger scale.

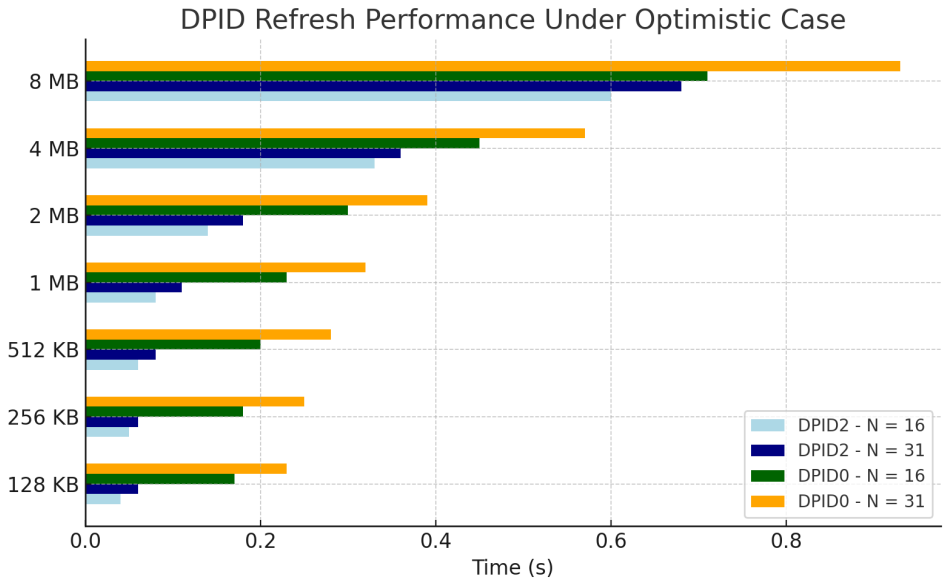

FIGURE 8.3. Performance of DPID Refresh under the optimistic case

Performance of our Refresh protocols with large data size:

Figure 8.3 plots the optimistic case refresh latency of DPID2 and DPID0 for block sizes ranging from 128 KB to 8 MB with a committee size of $N = 16$ and $N = 31$. Both curves exhibit an approximately linear increase in latency as the payload grows, but with markedly different slopes. For DPID2, refresh time rises from about 0.04 s to 0.45 s when $N = 16$ (and

from 0.06 s to 0.68 s when $N = 31$), reflecting the dominant $O(\ell)$ term in its $O(\ell + \lambda n^3)$ complexity. In contrast, DPID0 increases more steeply from 0.17 s to 0.71 s at $N = 16$. From 0.23 s to 0.93 s, DPID0's performance at $N = 31$ appears to be consistent with its square complexity.

Moreover, the ratio of DPID0 latency at $N = 31$ versus $N = 16$ remains roughly constant 1.5 times across all block sizes, directly demonstrating the linear dependency on n predicted by theory. By contrast, DPID2's n-dependent term λn^3 remains negligible throughout, so its latency curve for $N = 31$ is only marginally above that for $N = 16$. These trends confirm that our DPID2 refresh protocol minimises communication overhead for large payloads, while DPID0 incurs a moderate n-scaled penalty but still far outperforms schemes with full cubic complexity.

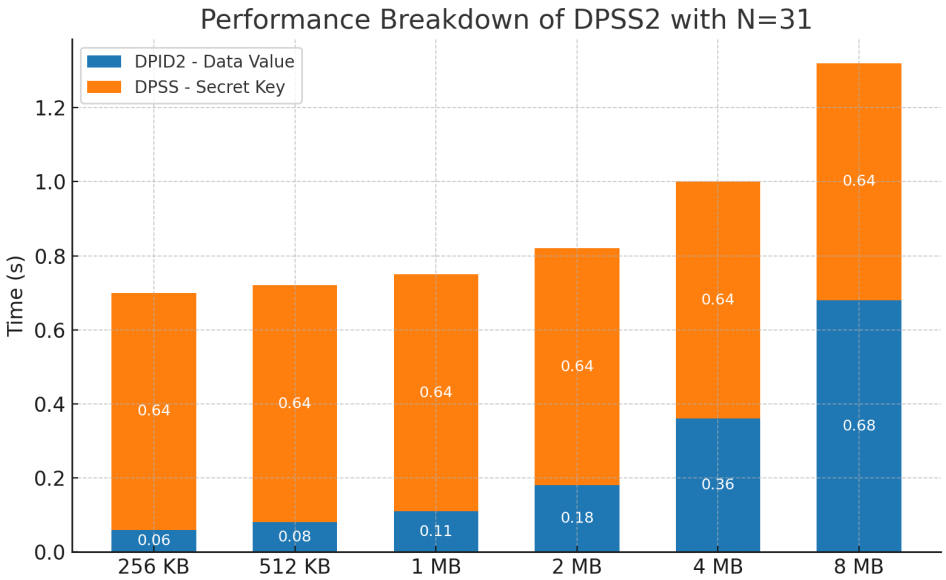

FIGURE 8.4. Performance breakdown of DPSS2 when $N = 31$

Overhead of Longlive DPSS scheme over our DPID: Finally, we test the overhead of Longlive DPSS protocol compared to DPID2, where the main overhead arises from the LongLive DPSS instance during compilation. Here, DPID2 and Longlive DPSS both include one round of dispersal and refresh. As shown in Fig.8.4, DPID2 exhibits a gradual increase with

data size, while the Longlive DPSS overhead mainly stems from secret key sharing.

CHAPTER 9

Conclusion

9.1 Summary of Contributions

This thesis introduced three dispersal-proof-in-dispersal (DPID) variants and a compiled distributed proactive secret-sharing (DPSS) protocol that together achieve near-optimal communication in the optimistic case while keeping recovery latency practical. Beyond the core protocols, we delivered a full security model, a reference implementation, and an empirical evaluation that highlights how careful committee selection and erasure-coding can cut honest-party bandwidth by an order of magnitude compared with classical schemes.

Key takeaways are:

- *Protocol Efficiency.* DPSS2 shows that $O(\ell)$ optimistic-path communication is attainable without exotic cryptography.
- *Practical Resilience.* The compiled DPSS tolerates up to $n/3$ Byzantine nodes and offers automatic recovery under moderate node churn.
- *Implementation Realism.* A 31-node test-net on AWS confirms theory as data refresh takes seconds for megabyte-scale secrets in a real-world network, and complaints are rare in benign epochs.

9.2 Future Outlook

- *Balanced storage via richer cover sets.* The current *weak* set-cover rule keeps the maths tidy by setting $k_4 = 0$. Dropping that shortcut would let us juggle fragments more intelligently—handing out extra pieces to beefier servers, reshuffling data when nodes churn, and squeezing more resilience out of the same hardware. Greedy heuristics, lightweight ILP solvers or even a handful of well-placed *if-else* rules could capture most of the gain without drowning in theory.
- *Capping worst-case bandwidth.* In the current design, a single Byzantine node can trigger a $\kappa\ell$-sized burst of evidence, which is tolerable for prototypes but prohibitive at scale. We propose to bound the pessimistic communication to $O(\ell)$ even under adversarial conditions, either by employing a one-shot dispersal with a constant-size aggregate signature or by proving such a bound impossible. In either case, the result will tighten the protocol specification.
- *Empirical evaluation via live deployment.* Integrating the DPID2-enhanced DPSS into a Filecoin plug-in or a Kubernetes operator will yield essential performance data: real-world egress bandwidth, recovery latency under churn, and the operational overhead of periodic committee refresh.

9.3 Closing Thoughts

Secure and usable decentralised storage has matured from an academic curiosity into a practical requirement for open blockchains, privacy-preserving cloud services and edge computing. By showing that proactive secret sharing can be both *secure* and *usable*, this thesis narrows the gap between elegant cryptography and production-grade systems. The roadmap outlined above, which includes lighter-weight zero-knowledge techniques, more expressive cover sets, stronger adversarial resilience and real-world deployments, traces a credible trajectory from research prototype to an enterprise-grade platform. The remaining milestones are implementation-intensive yet

conceptually straightforward. Once achieved, decentralised storage could become as commonplace and dependable as the replication layers that underpin modern data centres.

References

[1] Andreea B Alexandru, Erica Blum, Jonathan Katz, and Julian Loss. State machine replication under changing network conditions. In *Advances in Cryptology–ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part I*, pages 681–710. Springer, 2023.

[2] Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. Asynchronous verifiable information dispersal with near-optimal communication. *Cryptology ePrint Archive*, 2022.

[3] Nicolas Alhaddad, Sisi Duan, Mayank Varia, and Haibin Zhang. Practical and improved byzantine reliable broadcast and asynchronous verifiable information dispersal from hash functions. *Cryptology ePrint Archive*, 2022.

[4] Microsoft Azure. Introduction to azure storage. Online, 2024. Accessed: 2024-12-26.

[5] Joshua Baron, Karim El Defrawy, Joshua Lampkins, and Rafail Ostrovsky. Communication-optimal proactive secret sharing for dynamic groups. In *Applied Cryptography and Network Security: 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers*, pages 23–41. Springer, 2016.

[6] Cristina Bazgan, Jérôme Monnot, Vangelis Th Paschos, and Fabrice Serrière. On the differential approximation of min set cover. *Theoretical Computer Science*, 332(1-3):497–513, 2005.

[7] Amos Beimel. Secret-sharing schemes: A survey. *International Conference on Coding and Cryptology*, pages 11–46, 2011.

[8] Richard E Blahut. *Theory and practice of error control codes*. Addison-Wesley, 1983.

[9] G.R. Blakley. Safeguarding cryptographic keys. In *Proc. AFIPS 1979 National Computer Conference*, pages 313–317, 1979.

[10] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[11] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 88–97, 2002.

[12] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*, pages 524–541. Springer, 2001.

[13] Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. *IEEE Symposium on Reliable Distributed Systems*, 2005.

[14] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 42–51, 1993.

[15] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. *Foundations of Computer Science*, pages 383–395, 1985.

[16] Google Cloud. Object storage for companies of all sizes. Online, 2024. Accessed: 2024-12-26.

[17] Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2721, 2021.

[18] Sourav Das, Zhuolun Xiang, and Ling Ren. Near-optimal balanced reliable broadcast and asynchronous verifiable information dispersal. *Cryptology ePrint Archive*, 2022.

[19] Yvo Desmedt and Sushil Jajodia. Redistributing secret shares to new access structures and its applications. Technical report, Citeseer, 1997.

[20] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. *Foundations of Computer Science*, pages 427–438, 1987.

[21] International Organization for Standardization. ISO/IEC 27001:2013 Information Security Management. Online, 2013. Accessed: 2025-06-25.

[22] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. In

Public-Key Cryptography–PKC 2022: 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8–11, 2022, Proceedings, Part I, pages 252–282. Springer, 2022.

[23] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message franking via committing authenticated encryption. In *Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part III 37*, pages 66–97. Springer, 2017.

[24] Christoph U Günther, Sourav Das, and Lefteris Kokoris-Kogias. Practical asynchronous proactive secret sharing and key refresh. *Cryptology ePrint Archive*, 2022.

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

[26] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Advances in Cryptology—CRYPT0'95: 15th Annual International Cryptology Conference Santa Barbara, California, USA, August 27–31, 1995 Proceedings 15*, pages 339–352. Springer, 1995.

[27] Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 72(9):56–64, 1987.

[28] Protocol Labs. Interplanetary file system (ipfs). Online, 2015. Accessed: 2025-06-25.

[29] Protocol Labs. Filecoin: A decentralized storage network. Online, 2020. Accessed: 2025-06-25.

[30] Storj Labs. Storj: Distributed cloud storage. Online, 2018. Accessed: 2025-06-25.

[31] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[32] Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. Churp: dynamic-committee proactive secret sharing. In *Proceedings of the 2019 ACM SIGSAC*

Conference on Computer and Communications Security, pages 2369–2386, 2019.

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

[34] European Parliament and Council of the European Union. Regulation (eu) 2016/679 (general data protection regulation). Online, 2016. Accessed: 2025-06-25.

[35] Michael O Rabin. The information dispersal algorithm and its applications. In *Sequences*, pages 406–419. Springer, 1990.

[36] Matthieu Rambaud and Antoine Urban. Proactive secret sharing over asynchronous channels under honest majority (with ephemeral roles): Refreshing without a consistent view on shares. *Cryptology ePrint Archive*, 2022.

[37] David A Schultz, Barbara Liskov, and Moses Liskov. Mobile proactive secret sharing. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 458–458, 2008.

[38] Amazon Web Services. Summary of the amazon s3 service disruption in the northern virginia (us-east-1) region. Online, 2017. Accessed: 2025-06-25.

[39] Amazon Web Services. Aws simple cloud storage. Online, 2024. Accessed: 2024-12-26.

[40] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[41] Ars Technica. Amazon admits that employees review "small sample" of alexa audio. Online, 2019. Accessed: 2025-06-25.

[42] Stefano Tessaro and Chenzhi Zhu. Threshold and multi-signature schemes from linear hash functions. In *Advances in Cryptology – EUROCRYPT 2023*, volume 14008 of *Lecture Notes in Computer Science*, pages 628–658. Springer, 2023.

[43] Robin Vassantlal, Eduardo Alchieri, Bernardo Ferreira, and Alysson Bessani. Cobra: Dynamic proactive secret sharing for confidential bft services. In *2022 IEEE symposium on security and privacy (SP)*, pages 1335–1353. IEEE, 2022.

[44] Theodore M Wong, Chenxi Wang, and Jeannette M Wing. Verifiable secret redistribution for archive systems. In *First International IEEE Security in Storage Workshop, 2002. Proceedings.*, pages 94–105. IEEE, 2002.

[45] Yunzhou Yan, Yu Xia, and Srinivas Devadas. Shanrang: Fully asynchronous proactive secret sharing with dynamic committees. *Cryptology ePrint Archive*, 2022.

[46] Thomas Yurek, Zhuolun Xiang, Yu Xia, and Andrew Miller. Long live the honey badger: Robust asynchronous dpss and its applications. *Cryptology ePrint Archive*, 2022.

[47] Lidong Zhou, Fred B Schneider, and Robbert Van Renesse. Apss: Proactive secret sharing in asynchronous systems. *ACM transactions on information and system security (TISSEC)*, 8(3):259–286, 2005.