Asynchronous Dynamic-committee Proactive Information Dispersal and Its Applications

Alan Fekete and Zhenliang Lu and Qiang Tang
School of Computer Science, The University of Sydney
{alan.fekete,zhenliang.lu,qiang.tang}@sydney.edu.au[*]

ABSTRACT

There is a recent surge of study on dynamic-committee proactive secret sharing (DPSS), in which, not only the shares will be periodically refreshed (proactive secret sharing), but also the parties who hold the shares will also be updated. This has immediate application in blockchain that asking committees to hold a secret, and also decentralized storage with dynamic participants.

Despite substantial attention, DPSS still has high communication complexity, particularly when the secret/data has a large size.

In this article, we initiate the study of dynamic-committee proactive information dispersal (DPID). From notion point of view, we can view DPID as DPSS without confidentiality. We model, and construct DPID schemes with substantially smaller complexity; and to showcase its power, we also provide a general framework to compile DPID to DPSS. Combining with our DPID construction, we obtain the first DPSS with optimal communication when data size is large.

1 INTRODUCTION

With the growth of blockchain technology and decentralized apps [7, 21, 27], which leads an unprecedented need for the deployment of public ledger services for mission-critical applications on the global Internet. These public ledger services were maintained by mutually distrustful and geologically distributed nodes. As a result, Byzantine fault-tolerant (BFT) protocols, such as BFT atomic broadcast, Secure multi-party computation and BFT storage, are receiving renewed attention, as they are the foundational techniques for implementing decentralized infrastructures.

One important application is distributed storage systems, which have attracted much attention. The model of this problem is that the dealer stores some data among a set of nodes (or clients), and where the number of nodes is n, and at most f nodes have Byzantine behaviour. These Byzantine nodes can be controlled by an adversary, the basic requirement of this problem is that if any two clients recover the data, they can recover the same data; additionally, if the dealer is honest, the recovered data is the dealer's stored data. In order to recover the same data, one simple solution is for every node to store the whole data such that the recovery dealer can output the data if $n - f$ nodes send the same data. Clearly, the solution is very impractical. One main reason is that the nodes don't need to store the whole data, especially in a blockchain setting.

As a result, information dispersal is an innovation solution that considers each node to keep a fragment of the data such that a threshold node's fragment can reconstruct the original data. The earliest concept was proposed by Rabin [22] using an information dispersal algorithm (IDA), specifically, the 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 didn't consider the Byzantine nodes, that is, it can't handle this case once some fragments are wrong in the recast phase. Some later works [8] consider the information dispersal with verifiable in the asynchronous network and call it Asynchronous Verifiable Information Dispersal (AVID). In this scenario of information dispersal, each fragment can be verified, that is, during the dispersal phase, each node stores a valid fragment, and in the retrieval phase, a client can discard some invalid fragments when reconstructing the message.

However, as a storage system, especially in distributed cloud services, it is possible for some cloud service companies to go offline or close down, and for others to join; thus, we hope the cloud (store nodes) can change on a regular basis due to practical concerns. Aside from that, it is possible that the adversary will gradually corrupt more nodes.

In this paper, we consider information dispersal in a dynamical committee and a mobile adversary setting. Specifically, we consider that epochs make up a system's lifetime, and for each epoch, a group of selected nodes (called the committee) will store the data fragment. For distinct two epochs, the participating nodes of different epochs could be different, that is, support a dynamical committee. Besides, the adversary can corrupt different nodes in different epochs; eventually, it can corrupt all nodes in a whole lifetime, but it can't corrupt more than one-third of the nodes during each epoch.

Asynchronous Dynamic-committee Proactive Secret Sharing (DPSS) [8, 23, 28–30] is a important primitive for confidential BFT storage. Comparing with DPID, DPSS has additional *secrecy* property, that is: if the adversary corrupts no more than f nodes in a committee of any epoch e, then the adversary learns no information about the secret S during the whole lifetime. DPSS works under the same model as DPID, which also supports dynamic committees and a mobile adversary. The critical DPSS process is Refresh, which occurs when nodes of epoch $e - 1$ transfer the share to nodes of epoch e. However, the state-of-the-art DPSS [29] has a cubic cost in the communication complexity ($O(n^3\ell + n^3\lambda)$) and linear storage complexity ($O(n^2\ell + n^2\lambda)$) in this process (ℓ is the size of the secret, λ is the security parameter and the size of all kinds of signature, and $O(n)$ is the size of every committee), which is due to the need to keep confidential. Clearly, it is unacceptable for confidential BFT storage with the dynamic committee due to the high cost of both communication complexity and storage complexity, hence, a natural question arose here:

[*] The paper is named alphabetically and Zhenliang is the leading author.

Can we asymptotically improve the cost of the DPSS *when the input size is large?*

1.1 Our contributions

We answer the above question affirmatively, and first time present three asynchronous Dynamic-committee Proactive information dispersal protocols, called DPID0, DPID1 and DPID2, respectively. Comparing with DPSS, DPID lacks confidential. Then we are inspired by "Hybrid Encryption" in the setting of cryptography, for any large input data, we use the symmetric-key algorithm, the (distribution) dealer first uses the key to encrypt the plaintext data, then uses DPSS to disperse the key, and DPID to disperse the ciphertext. Let us go into more detail below.

Table 1: Performance metrics of DPID

Protocol	**Complexity**				**Trusted Setup**
	Decode	Msg	Optimistic*	Worst	
DPID0	$\mathcal{O}(1)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n\ell + \lambda n^2)$	$\mathcal{O}(n\ell + \lambda n^2)$	×
DPID1	$\mathcal{O}(1)$	$\mathcal{O}(n^2)$	$\mathcal{O}(\kappa\ell + \kappa\lambda n^2)$	$\mathcal{O}(\kappa\ell + \kappa\lambda n^2)$	✓
DPID2	$\mathcal{O}(e)$	$\mathcal{O}(n^2)$	$\mathcal{O}(\ell + \lambda n^3)$	$\mathcal{O}(\kappa\ell + \lambda n^3)$	✓

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

First DPID **constructions**. In information dispersal, any two clients always can 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 distinct epochs. As a core technology of DPID, we also first time present an Asynchronous complete verifiable information dispersal protocol (ACVID). Comparing with Asynchronous verifiable information dispersal (AVID), the ACVID has one extra property, if some honest node terminates the protocol with some valid fragment, then all honest nodes also can terminate the protocol with some valid fragment.

First, we present DPID0, which is based on ACVID, where the ACVID is no trusted setup. In this paper, when we talk the communication complexity of DPID, we just consider the cost during Refresh. The communication complexity of DPID0 is $\mathcal{O}(n\ell + \lambda n^2)$.

Second, we present DPID1, which is based on DPID0. Essentially, we choose a sub-committee from new committee such that this sub-committee contains one honest node with high probability. In this case, the communication complexity reduce to $\mathcal{O}(\kappa\ell + \kappa\lambda n^2)$, where κ is the size of sub-committee. As the introduction elaborates, the term related to data size is the dominant cost for a storage system. In DPID1, we achieved the cost term about data size independent of n.

Third, we proposed DPID2, where the DPID2 needs to invoke NIZK to guarantee one data can produce two different auxiliary information under different parameters, then when the nodes can make sure these two auxiliary information coming to the same data. In DPID2, the optimal communication complexity is constant when $\ell > \lambda n^3$, however, the worst case of communication complexity is $\mathcal{O}(\kappa\ell + \lambda n^3)$.

DPSS **with large input**. As we briefly elaborate above, the DPID is the core building block of DPSS with large input. As shown in Table 2, the communication complexity has a huge improvement to the state-of-the-art. Besides, in order to obtain the constant complexity in the optimal case, we compromise the time complexity during the recovery phase, that is, for epoch e, it needs to decode e times to obtain the original stored data.

1.2 Technical and Challenge

DPID is the first asynchronous dynamic committee information dispersal. Essentially, DPSS can serve the same purpose. However, if we don't care about the confidential information dispersal, we hope to achieve its low communication complexity rather than high complexity like DPSS, DPID handles several technical challenges in a different way, below is further information about this.

why do we need acvid instead of avid? In fact, we mainly consider AVID in information dispersal, it always can make sure any recovery client output the same data. In our setting, we consider an adversary can control f nodes across both the old and new committees, for a total $2f$ nodes. It is possible that the corrupted nodes in the old and new committees are the same. At the end of AVID, although all honest nodes terminate the protocol, only $n-2f$ honest nodes received the fragment. Unfortunately, it has a devastating disadvantage, as illustrated in Figure 1, at epoch 1, $\mathcal{P}_{f+1}$ to $\mathcal{P}_{2f+1}$ have received the fragment, and any client can recover the same data during epoch 1. However, if the adversary corrupts nodes $\mathcal{P}_{f+1}$ to $\mathcal{P}_{2f}$ and releases $\mathcal{P}_1$ to $\mathcal{P}_f$, in this case, $\mathcal{P}_{2f+1}$ is the only node that has fragment at the start of epoch 2, hence, the data is not integrity and can't be recovered such that the refresh is not meaningful.

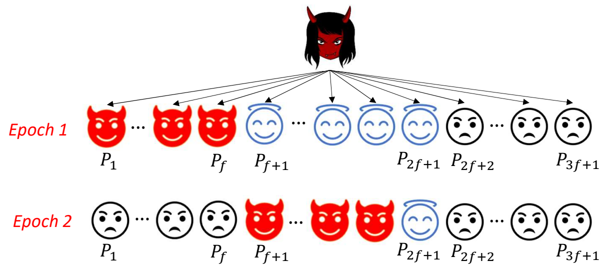

Figure 1: The very high level of corrupted nodes in AVID**: the new committee has fragments at the end of epoch 1 and the old committee has fragments at the start of epoch 2.**

Hence, all honest nodes must have the fragment once terminating in order to defend against the mobile adversary. Unlike Reliable Broadcast (RBC), each node only outputs a portion of the data in information dispersal, but in RBC, every node outputs the entire data. In reality, letting the dealer first call the dispersal black-box subprotocol and then, if some honest nodes terminate without fragment, these nodes execute a recast black-box subprotocol, is an inefficient approach to achieve the ACVID. For example, in the ideal situation: all honest nodes received fragment once terminating, besides, all honest nodes also output a portion of auxiliary (proof) information (which can verify whether any fragment is valid or not) in the state-of-the-art AVID [3, 4, 11]. However, these state-of-the-art protocols' design is unsuitable for our setting, the main reason is all their protocol use an online error correcting code to recover the auxiliary information, but which needs at least receiving $n-f+r$ messages to tolerate r bad messages. As previously described,

Table 2: Comparison for performance metrics of DPSS

Protocol	Network	Tolerance	Dynamic	Complexity			Trusted Setup
				Decode	Optimistic*	Worst	
Herzberg et al. [17]	Sync	$n/2$	×	$O(1)$	$O(n^3\ell + \lambda n^3)$	$O(n^3\ell + \lambda n^3)$	×
Desmedt et al. [12]	Sync	$n/2$	✓	$O(1)$	$O(n^2\ell + \lambda n^2)$	–§	×
Wong et al. [26]	Sync	$n/2$	✓	$O(1)$	$O(\exp(n\ell + \lambda n))$	$O(\exp(n\ell + \lambda n))$	×
Baron et al. [5]	Sync	$(1/2 - \varepsilon)n$	✓	$O(1)$	$O(n^3\ell + \lambda n^3)$	$O(n^3\ell + \lambda n^3)$	×
CHURP [19]	Sync	$n/2$	✓	$O(1)$	$O(n^2\ell + \lambda n^2)$	$O(n^3\ell + \lambda n^3)$	✓
Goyal et al. [13]	Sync	$n/2$	✓	$O(1)$	$O(n^2\ell + \lambda n^2)$	$O(n^3\ell + \lambda n^3)$	✓
Schultz-MPSS [24]	Partial Sync	$n/3$	✓	$O(1)$	$O(n^4\ell + \lambda n^4)$	$O(n^4\ell + \lambda n^4)$	×
COBRA [25]	Partial Sync	$n/3$	✓	$O(1)$	$O(n^3\ell + \lambda n^3)$	$O(n^4\ell + \lambda n^4)$	×
Cachin et al. [8]	Async	$n/3$	×	$O(1)$	$O(n^4\ell + \lambda n^4)$	$O(n^4\ell + \lambda n^4)$	✓
Zhou et al. [30]	Async	$n/3$	✓	$O(1)$	$O(\exp(n\ell + \lambda n))$	$O(\exp(n\ell + \lambda n))$	×
Shanrang [28]	Async	$n/4$	✓	$O(1)$	$O(n^3 \log n\ell + \lambda n^3 \log n)$	$O(n^4\ell + \lambda n^4)$	✓
Y-VSS [23]	Async	$n/2$	✓	$O(1)$	$O(n^4\ell + \lambda n^4)$	$O(n^4\ell + \lambda n^4)$	×
Long Live [29]†	Async	$n/3$	✓	$O(1)$	$O(n^3\ell + \lambda n^3)$	$O(n^3\ell + \lambda n^3)$	×‡
DPID0+[29]	Async	$n/3$	✓	$O(1)$	$O(n\ell + \lambda n^3)$	$O(n\ell + \lambda n^3)$	×
DPID1+[29]	Async	$n/3$	✓	$O(1)$	$O(\kappa\ell + \lambda n^3)$	$O(\kappa\ell + \lambda n^3)$	✓
DPID2+[29]	Async	$n/3$	✓	$O(e)$	$O(\ell + \lambda n^3)$	$O(\kappa\ell + \lambda n^3)$	✓

*optimistic case means all nodes are honest and all messages have no delay. §Desmedt et al. [12] doesn't support Byzantine setting. † the state-of-the-art DPSS.
‡ Long Live [29] has the same communication complexity performance either way, if it is no trusted setup, then it has to rely on a high threshold setting.

just $n - 2f$ honest nodes have the correct fragment of auxiliary information at the start of the new epoch, so it is hard to recover the auxiliary information to discard some invalid message such that it is impossible to recover the original store data, let alone the Refresh.

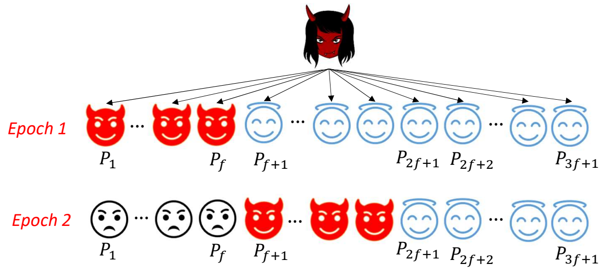

Figure 2: The very high level of corrupted nodes in ACVID

DPID. With the ACVID at hand, we can make sure at the start of any epoch, the old committee still has at least $n - 2f$ honest nodes that have a fragment and whole auxiliary information. First, we consider the node of the old committee sends fragments to all nodes of the new committee, the node of the new committee collects enough messages from distinct nodes of the old committee, then these nodes recompute the fragment and some auxiliary information to store. The auxiliary information can verify the fragment, but we need to make sure the auxiliary information can't be forged, otherwise, the nodes of the new committee received some wrong auxiliary information and then use some wrong fragments to recover a wrong data, which will influence the security. Besides, we also need the auxiliary information that can directly verify any fragments instead of recasting them and then verifying fragments to discard some invalid fragments. Noted that at most $O(n)$ fragments from $O(n)$ different members, so the size of the auxiliary information is $O(n\lambda)$. Hence, we add a witness for the auxiliary information by signature, in this case, any node of the old committee sends a message to new committee members, then the message needs to contain fragment, auxiliary information, and a witness.

DPID0. As a basic version of DPID, which is extremely simple, it doesn't need a trusted setup and any type of consistent view, such as common coin, 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 includes many signatures of the old committee member signed, which size is equal to the auxiliary information, both are $O(n\lambda)$. hence, the size of the message from an old committee member is $O(\ell/f + n\lambda)$, so the communication complexity of Refresh is $O(n\ell + n^2\lambda)$. Besides, we noted that "yoso" security [23] (that is, 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.

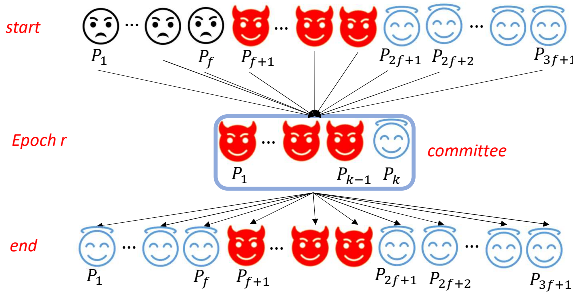

Figure 3: The very high-level of corrupted nodes in DPID1 and DPID2.

DPID1. The communication complexity of basic version is $O(n\ell + n^2\lambda)$. In DPID1, as figure 3, we choose a subcommittee with size κ

from the new committee such that there exists one honest node in the subcommittee with high probability, in this case, the communication complexity of DPID1 is independent with n when $\ell > n^2\lambda$, the communication complexity of DPID1 is $O(\kappa\ell + \kappa\lambda n^2)$.

DPID2. In order to further reduce the communication complexity, we use NIZK against the malicious nodes sending junk messages. In DPID2, each node of the old committee dispersal the fragment to the whole new committee instead of multicast the whole fragment, every node of the new committee just receives a piece of the fragment, in this case, we need to use MVBA to make sure all nodes of new committee have a consistent view to decide whose piece will be store. Suppose the old committee members $\mathcal{P}'_1$, $\mathcal{P}'_2$ and $\mathcal{P}'_3$'s piece need to store, that is for any $\mathcal{P}_i$ of new committee, $\mathcal{P}_i$ will take $m_{1,i}||m_{2,i}||m_{3,i}$ as new fragment to store, and use $\bigoplus\{\mathcal{H}(j, m_{j,i})\}_{j\in[f_{e-1}+1]}$ to verify $\mathcal{P}_j$'s fragment, $\{\bigoplus\{\mathcal{H}(j, m_{j,i})\}_{j\in[f_{e-1}+1]}\}_{i\in[n_e]}$ as the new auxiliary information. We use $\{\mathcal{H}(j, m_{j,i})\}$ to against malicious nodes to do any other order combinations for these pieces. However, if some client would like to recover the original data in epoch e, then he needs to do e times recovery.

1.3 Related work

The information dispesarl either start from AVID, or start from Verifiable Secret Sharing (VSS) if considering confidential information dispersal. However, all these works do not consider a mobile adversary, where the mobile adversary can eventually corrupt all ndoes in a long-lived system. In this case, the adversary can collect the required number of shares to reconstruct the secret in VSS, then confidentiality can't be guaranteed. In AVID, the adversary can erasure/revise the store fragments such that any clients can't recover the stored data.

Hence, some new scheme was presented to protect against such adversaries. One very similar works are in the form of Proactive (Verifiable) Secret Sharing (PSS or PVSS) [15], Nodes in a PSS (or PVSS) protocol periodically re-randomize their shares while keeping the secret key the same. As a result of the re-randomization, all shares previously compromised by the adversary lose their value. However, PSS just considers refreshing shares within the same committee, it doesn't support mobile adversaries and dynamic committees.

To overcome these challenges, previous works propose a generalization 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, at the same time, the adversary learns nothing about the secret during this process. However, some prior works either incur a high communication cost to reshare a secret [8, 23–26, 28–30], or assume the network is synchronous or partial synchronous [5, 12, 13, 17, 19, 24–26], or consider suboptimal fault tolerance [5, 28]. In this paper, we consider asynchronous DPSS with optimal tolerance, the state-of-the-art Asynchronous DPSS has communication complexity up to $O(n^3\ell + n^3\lambda)$. Many confidential BFT-storage applications directly invoke DPSS as the underlying scheme, clearly, this way is very inefficient, especially for data-storing applications itself, as the data size is large.

A recent interesting work of DPSS [23] (called Y-VSS) can tolerate up to 1/2 corrupted nodes in an asynchronous network, this work needs to assume the dealer is honest and two committees do not overlap. This work is not surprising under this assumption. We also can easily revise our work to achieve 1/2 tolerance under the same assumption. However, if we consider two committees that can intersect, as illustrated in Figure 4, 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 data instead of some piece of data, otherwise, the recovery will fail.

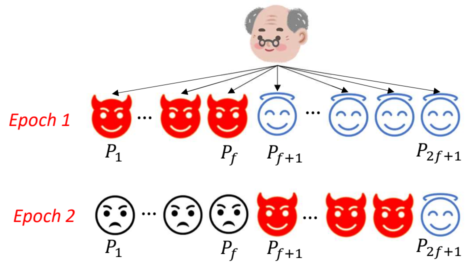

Figure 4: The very high-level of corrupted nodes in AVID or ACVID

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 nodes are malicious. (1), if the committees do not overlap: in this case, the dealer first invokes reliable broadcast (RBC) to boradcast data, it make sure all honest nodes received the same data; then in the next epoch, all honest nodes invoke Asynchronous Data Dissemination (ADD) [10] 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 of data in this way, and key refresh with Y-VSS [23], then the communication complexity is $O(n\ell + n^4\lambda)$ instead of $O(n^4\ell + n^4\lambda)$ [23]. (2), if the committees have an overlap: in this case, the dealer first invokes RBC to boardcast data M which can make sure all honest nodes received the same data M, then all honest nodes do a (threshold) signature of $\mathcal{H}(M)$ 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, it also can make sure all honest nodes have the same data. If we do BFT storage service without confidentiality in this way, then the communication complexity is $O(n^2\ell + n^3\lambda)$. With confidentiality, then the communication complexity is $O(n^2\ell + n^4\lambda)$.

2 MODELS AND GOALS

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

Setup. We consider a fully-connected distributed system composed by a universe of clients/nodes $\Pi := \{\mathcal{P}_1, \mathcal{P}_2, \cdots\}$ and a dealer $\mathcal{P}_d$, all nodes (including dealer) have a unique identity. The time of whole system is divided into fixed intervals of predetermined length called epochs. In each epoch, a group of nodes was specified, and the group nodes called the 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. Besides, suppose current epoch is e, we also need all nodes (non-committee members and committee members) known these public informations. 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 powerful active adversary who can corrupt nodes at any time. Once a node is corrupted by the adversary, it is assumed to be corrupted until the end of the current epoch. There are up to f_e corrupted nodes ($3f_e + 1 \leq n_e$) during epoch e, and these faulty nodes are fully controlled by a probabilistic polynomial-time bounded adversary [9, 20], i.e., the adversary can get all the faulty nodes' initial internal states and also can let these nodes deviate from the protocol arbitrarily. Note that during the refresh between epochs $e-1$ and e, committee C_{e-1} and C_e may intersect, 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.

This paper assumes that the adversary does not actively attack the system across multiple epochs due to the impossibility result [2]. We follow the same epoch definition [24, 29] to constrain the power of the adversary to overcome the impossibility. Additionally, we observe that the impossibility can be addressed [8] by assuming channels that ensure a message can be received in an epoch if and only if it has been broadcast 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 asynchronous network, the adversary has the ability to arbitrarily delay and reorder messages, 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.

Notations. We define ℓ as the size of data (or input value) and λ as the cryptographic security parameter that captures the size of the (threshold) signature and the hash value throughout the paper. We use $\mathcal{H}$ to denote a collision-resistant hash function. For an integer $x > 0$, let $[x]$ denote the set $\{1, 2, \cdots, x\}$. 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. For the sake of convenience, $\Pi[\mathsf{ID}]$ represents protocol Π with a session identifier ID, and $y \longleftarrow \Pi[\mathsf{ID}](x)$ means to invoke $\Pi[\mathsf{ID}]$ on input x and wait for its output y. Besides this, $x||y$ denotes concatenating two strings x and y. Throughout the paper, we always consider $O(n_e) = O(n)$ and $n_e = 3f_e + 1$ for any epoch e. Namely, our protocol is optimally resilient for any epoch and $O(n_{e-1}) = O(n_e) = O(n)$.

2.1 Design goals

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, any client always can recover a data if he can received $f_e + 1$ valid fragments in current epoch e, and the data is the same even if recasting in a different epoch. Formally, the DPID satisfies the following properties with all but negligible probability:

for any node of new committee C_e in epoch e,

- *Termination.* If DPID is initiated by a correct dealer, then all honest nodes eventually output a value.
- *Agreement.* If an honest node output 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 some block B' once he received 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 block B'. If the dealer was honest and input B, then $B' = B$.
- *Integrity.* If any honest node of C_e (C_{e-1}) can output a value, then any honest clients eventually can reconstructs a block B' (B'') during epoch e ($e-1$), then $B' = B''$.

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 always can recover a confidential data if he can received $f_e + 1$ valid messages in current epoch e, the confidential data remains the same even when recasting in a different epoch. Besides that, the adversary learns no information about the data except by recasting it. Formally, the DPSS satisfies the following properties with all but negligible probability:

for any node of new committee C_e in epoch e,

- *Termination.* If DPSS is initiated by a correct dealer, then all honest nodes eventually output a valid secret share.
- *Agreement.* If an honest node output a valid secret share, then all honest nodes eventually output a valid secret share.
- *Availability.* If an honest node can output a secret share, then any honest client eventually can reconstructs some secret S' once he received 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 any honest node of C_e (C_{e-1}) can output a value, then any honest clients eventually can reconstruct a block S' (S'') during epoch e ($e-1$), then $S' = S''$.
- *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.

3 PRELIMINARIES

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

Reliable broadcast (RBC): there is a protocol running among a set of n nodes [6], where a node called sender whose aim is to broadcast a value to all the other nodes. Formally, an 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.

Multi-valued validated Byzantine agreement (MVBA**)** [1, 9] is a Byzantine agreement with external validity, such that the output value satisfies a universal predicate function Q that is known to all of them. Except with negligible probability, the MVBA (with predicate Q) satisfies the following properties:

- *Termination.* If all honest nodes input a value satisfying the predicate Q, then each honest node would output a value;
- *External-Validity.* If an honest node outputs a value v, then $Q(v) = 1$;
- *Agreement.* All honest nodes output the same value.

Asynchronous Data Dissemination (ADD**)** [10] is a protocol, it can make sure if 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 $\perp$, then it can guarantee that all honest nodes eventually output M. In this paper, we consider that at least $f_{e-1} + 1$ honest nodes have common message M and other honest nodes have $\perp$ in the old committee C_{e-1}, the ADD can guarantee that all honest nodes of new committee C_e eventually output M.

The ADD satisfies the following properties except with negligible probability:

- *Termination.* 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 input $\perp$, then each honest node of new committee would output a value;
- *Validity.* If one honest node of new committee output M, then M is the input of some honest node of 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.

Remark that we noted that the ADD [10] just make sure all other honest nodes within the same committee have the same value. Throughout the paper, when any node of the old committee invokes ADD, either (1) we ask every message will send to all nodes (including all new committee members), but all new committee members just keep listening and simulate themselves as a mute honest node in the old committee; or (2) we do some minor revise in [10], and Algorithm 10 is the revised ADD, it can help us better understand the ADD protocol work in the dynamic setting, and the revised ADD satisfies "yoso" security.

$(1, \kappa, \epsilon)$-Committee election (CE**) [16]:** A CE protocol is executed among n nodes. If at least $f + 1$ honest nodes participate, the protocol can output a κ-sized committee C such that at least one of C is honest nodes. Specifically, a protocol is said to be $(1, \kappa, \epsilon)$-committee election, if it satisfies the following properties except with negligible probability:

- *Termination.* If $f + 1$ honest nodes participate the protocol, then all honest nodes output C;
- *Agreement.* all honest nodes output the same committee C;
- *Validity.* If any honest node outputs C, (i) $|C| = \kappa$, (ii) the probability of every node $P_i \in C$ is same, and (iii) C contains at least one honest node with at least probability $1 - \epsilon$;
- *Unpredictability.* The probability of the adversary to predict the returned committee is at most $1/\binom{n}{\kappa}$ unless at least one honest node has invorked the protocol.

Remark that a $(1, \kappa, \epsilon)$-CE can be constructed directly from a threshold coin-tossing, which can be readily derived from threshold signatures. Note that the CE message complexity is $O(n^2)$ and the communication complexity is $O(\lambda n^2)$ in expectation.

Non-interactive zero knowledge proofs (NIZK**) [14]** allow one party (the prover) prove validity of some statement to another party (the verifier), nothing other than the statement's validity is revealed, and no interaction between the prover and the verifier is required.

Let $R := \{x, w\}$ be an efficiently computable binary relation, where x is the statement and w is the witness. A proof system for relation R consists of a key generation algorithm KeyGen, a prover P and a verifier V. The key generation algorithm produces a CRS string σ. The prover inputs (σ, x, w) and produces a proof π. The verifier inputs (σ, x, π) and outputs 1 if the proof is valid and 0 otherwise. More formally, an NIZK has the following guarantees:

- *prove*: $\pi \leftarrow P(\sigma, x, w)$ for $(x, w) \in R$
- *verify*: $V(\sigma, x, \pi) = 1$ if $\pi \leftarrow P(\sigma, x, w)$ for $(x, w) \in R$

Remark that for simplicity, throughout this paper, we omit the CRS string σ when there is no ambiguity.

4 DPID0: ASYNCHRONOUS DYNAMIC-COMMITTEE PROACTIVE INFORMATION DISPERSAL WITHOUT TRUSTED SETUP

In this section, we present our first basic DPID, which is called DPID0. It doesn't need any trusted setup, just relying on only public key infrastructure (PKI). Besides, our DPID0 also doesn't rely on a consistency view.

High level overview. The core of our design is to present a new information dispersal protocol. As briefly elaborated in the Introduction (Figure 1), the reason that the previous information dispersal just guarantees $f + 1$ honest nodes have correct fragments at the end of the dispersal phase, although all honest nodes can terminate with the same auxiliary (or proof) information in this phase, this auxiliary information (sometimes also called proof information) can verify every fragment whether valid or not. During Refresh, because 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 input participates in the Refresh phase, while other honest nodes participate in Refresh with no input value.

A possible solution is first to execute the dispersal subprotocol of AVID, if some honest node found he has not received the fragments once the dispersal phase terminates, then he will invoke the retrieval subprotocol of AVID, it is possible that it found the dealer send a junk data. For example, the regenerated fragments obtained from decoding the recovered data are inconsistent with the auxiliary information. In this case, the nodes don't know what fragment he will store beside $\perp$. However, $\perp$ is hard to have the validity, otherwise, the corrupt nodes always send the ⊥ to the recast (retrieval) nodes, such that the recast node always uses ⊥ to decode which will bring the agreement issues. Let us considers the next case at the end of epoch $e-1$: if $f+1$ honest nodes store valid fragments, but these fragments just can reconstruct junk data, other honest nodes and corrupted nodes store ⊥; then at the start of epoch e, just one honest node stores a valid fragment, and all other honest nodes store the ⊥, however, a new committee member just waits for $f+1$ valid fragments, note that ⊥ considers being invalid message, hence, the Refresh process will get stuck.

Another possible solution is if some honest node has not received the corresponding fragment and retrieved junk data via $f+1$ valid fragments, then the honest node stores the $f+1$ valid fragments instead of outputting a ⊥. In this way, at least $f+1$ honest nodes terminate with output one fragments, f honest nodes terminate with output $f+1$ fragments. Note that the size of $f+1$ fragments is equal to the size of the data. Then the communication complexity will be up to $O(n^2\ell)$ during Refresh (details see algorithm 4), where ℓ is the size of the data.

In this paper, we consider all honest nodes instant invoke the retrieval subprotocol once the dispersal subprotocol terminates. Because the agreement of dispersal, we can make sure all honest nodes can output the same data from the retrieval subprotocol, then all honest nodes recompute themselves fragment and auxiliary information as the output if the data is not the ⊥, otherwise, all honest nodes set both the fragment and auxiliary information with ⊥. In either case, all honest nodes have the same auxiliary information; however, 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.

We note that instantly invokes retrieval subprotocol once the dispersal terminates, it can be thought the process of RBC protocol. Hence, in our basic version protocol, we consider all honest nodes waiting for the output of RBC, then compute the auxiliary information $\mathbf{D}$, and sign the auxiliary information to form a witness to prove the $\mathbf{D}$ is unique.

Algorithm 1 The DPID0 with epoch $e=1$: ACVID dispersal subprotocol with sender $\mathcal{P}_s$ for party $\mathcal{P}_i$: without trusted setup

```
Initialization: Let RBC[$e$] refer to one instance of the reliable broadcast protocol,
where $\mathcal{P}_s$ is the sender of RBC[$e$]; $T \leftarrow \{\}$;
if $\mathcal{P}_i = \mathcal{P}_s$ and received input $M$ then
    as the sender to invoke RBC[$e$] with input $M$;
upon receiving output $M$ from RBC[$e$] do
    if $M \neq \perp$ then
        let $\{m_j\}_{j\in[n_e]} :=$ Enc($M, n_e, f_e$);
        let $\mathbf{D}_e := \{\mathcal{H}(m_j)\}_{j\in[n_e]} = \{\mathcal{H}(m_1), \ldots, \mathcal{H}(m_{ne})\}$;
        $\sigma_i \leftarrow$ Signature($e, \mathcal{H}(\mathbf{D}_e)$); multicast Final($e, \sigma_i$) to $C_e$;
        upon receiving Final($e, \sigma_j$) from $\mathcal{P}_j(\in C_e)$ for the first time do
            if Verify($e, \mathcal{H}(\mathbf{D}_e), (j, \sigma_j)$) =1 then $T_e \leftarrow T_e \cup \{j, \sigma_j\}$;
        upon $|T_e| = f+1$ do
            return $block[e] := (m_i, \mathbf{D}_e, T_e)$.
```

Algorithm 2 The DPID0 retrieval subprotocol, with epoch e for party $\mathcal{P}_i$

```
Initialization: $T \leftarrow \{\}$;
if $\mathcal{P}_i$ would like to retrieval 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 $\mathcal{P}_j(\in C_e)$ for the first time 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($T$); return $M'$.
```

Algorithm 3 Validation function of DPID0, with epoch e

```
function Verify($i, block[e]$):
    if $block[e] \neq \emptyset$ then parse $block[e] := (m_i, \mathbf{D}_e, T_e)$;
        if $|T_e| = f+1$ and $\mathcal{H}(m_i) = \mathbf{D}_e[i]$ then
            if Verify($e, \mathcal{H}(\mathbf{D}_e), (j, \sigma_j)$) = 1 for any $(j, \sigma_j) \in T_e$ then
                return 1;
    else return 0.
```

Algorithm 4 Asynchronous DPID0 with epoch $e > 1$: Refresh subprotocol for party $\mathcal{P}_i$: without trusted setup

```
Initialization: $T \leftarrow \{\}$; $T_e \leftarrow \{\}$; Public input: $n_{e-1}, f_{e-1}, n_e, f_e$;
if $\mathcal{P}_i \in C_{e-1}$ then                                        ▷ old committee $C_{e-1}$
    if Verify($i, block[e-1]$) = 1 then
        parse $block[e-1] := (m_i, \mathbf{D}_{e-1}, T_{e-1})$;
        ADD[$e$]($\mathbf{D}_{e-1}$); multicast Fresh($e, m_i$) to $C_e$;
if $\mathcal{P}_i \in C_e$ then                                            ▷ new committee $C_e$
    upon receiving Fresh($e, m_j$) from $\mathcal{P}_j(\in C_{e-1})$ for the first time do
        $T = T \cup \{(j, m_j)\}$
    wait until ADD[$e$] return $\mathbf{D}_{e-1}$ 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
            $M' \leftarrow$ Dec($T$); let $\{\overline{m_j}\}_{j\in[n_e]} :=$ Enc($M', n_e, f_e$);
            let $\mathbf{D}_e := \{\mathcal{H}(\overline{m_1}), \ldots, \mathcal{H}(\overline{m_{ne}})\}$;
            $\sigma_i \leftarrow$ Signature($e, \mathcal{H}(\mathbf{D}_e)$); multicast Final($e, \sigma_i$) to $C_e$;
            upon receiving Final($e, \sigma_j$) from $\mathcal{P}_j(\in C_e)$ for the first time do
                if 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 $block[e] := (\overline{m_i}, \mathbf{D}_e, T_e)$.
```

Constructing the DPID0. Now we describe our protocol. The formal description of the protocol is shown in Algorithm 1,2,3 and 4. The DPID0 protocol mainly consists of two phases, namely, dealer dispersal data (for epoch 1) and refresh phase (for epoch > 1). Specificity, the dealer dispersal his data into the committee of epoch 1 through ACVID (Algorithm 1), and then the old committee refreshes himself fragment to the new committee through DPID (Algorithm 4) in any epoch $e > 1$. The detailed protocol proceeds as follows:

- *dealer dispersal in epoch 1*: (Algorithm 1 of ACVID). Upon the dealer $\mathcal{P}_s$ receives its input data M, it activates RBC[e] as the sender and with M as input; any honest node of the new committee once receiving the output of RBC, then computing the fragments via the encoding algorithm to generate the $\mathbf{D}$ as the auxiliary information. Thanks to the agreement and termination of RBC and the deterministic algorithm of encoding, all honest nodes will have the same $\mathbf{D}$. After generating $\mathbf{D}$, the honest node signs the hash of $\mathbf{D}$, then waits for $f+1$ signatures for the same $\mathbf{D}$ from distinct nodes as a witness, finally forming a block as the output.

  Note that, we need to varify the stored fragment, otherwise, the adversary may arbitrarily take some fragment in these nodes' local storage disk which will hate the security.

For example, $M1 \neq M2, n = 4$ and $f = 1, \{m_1, m_2, m_3, m_4\} \leftarrow \mathsf{Dec}(M, 4, 1)$ and $\{m'_1, m'_2, m'_3, m'_4\} \leftarrow \mathsf{Dec}(M, 4, 1)$, $D := \mathcal{H}(m_1)_{i\in[4]}$ and $D' := \mathcal{H}(m'_1)_{i\in[4]}$. Consdiering $\mathcal{P}_4$ is malicous nodes during epoch $e-1$ and taking $(m'_4, \mathbf{D}')$ as the output of ACVID. Suppose $C_e := \{\mathcal{P}'_1, \mathcal{P}'_2, \mathcal{P}_3, \mathcal{P}_4\}$ and $\mathcal{P}_4 \in C_e$ is an honest node at the epoch e, that is the adversary releases the control for $\mathcal{P}_4$, and $\mathcal{P}_3 \in C_e$ is the malicious node. If there has no witness for auxiliary information $\mathbf{D}$, then $\mathcal{P}_4$ will send $\{m'_4, \mathbf{D}'\}$ to all nodes of C_e, at the same time $\mathcal{P}_3$ sends $\{m'_3, \mathbf{D}'\}$ to all nodes of C_e, the adversary dealy all other messages of nodes in C_{e-1}, then at the end of epoch e, $\mathcal{P}'_1$ will output $\{m'_1, \mathbf{D}'\}$ and $\mathcal{P}'_2$ will output $\{m'_2, \mathbf{D}'\}$, $\mathcal{P}_3$ will output $\{m'_3, \mathbf{D}'\}$, however, this case will violate the security because any client can retrieval two different value in epoch $e-1$ and e.

- Refresh after epoch 1: (Algorithm 4). 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 yes, then invokes ADD with $\mathbf{D}_{e-1}$ as input, and sends fragment to all nodes of C_e via Fresh message type. All nodes of C_e wait for the output of ADD$[e]$, then filter some invalid fragments from C_{e-1}. Once receiving $f_{e-1}+1$ valid fragments, then generating data M' and recomputing $\mathbf{D}_e$, after that, all honest nodes sign the hash of $\mathbf{D}_e$, once forming a witness, then all nodes of C_e output.
- retrieval at any epoch: (Algorithm 2). When a node/client $\mathcal{P}_j$ would like to retrieval data at epoch e, he will multicast retrieval(e) to C_e; for any honest nodes of C_e, once he received a retrieval(e) from $\mathcal{P}_j$, if he local $block[e]$ is valid, then he will send RECAST$(e, block[e])$ to $\mathcal{P}_j$; $\mathcal{P}_j$ waits for $f_e + 1$ valid RECAST messages from distinct nodes, he will then decode these fragments into data and output the data.
- Validation function of DPID0 at any epoch: (Algorithm 3). the validation function needs to check the fragment is consistence with the auxiliary information in $block[e]$, and the auxiliary information was endorsed by at least $f_e + 1$ nodes.

Security intuition of DPID0**:** The Algorithm 1-4 implements all properties of DPID. Its securities can be intuitively understood as follows:

- *Termination.* For epoch $e = 1$, the termination immediately follows the *validity* and *agreement* of RBC, all honest nodes will have the same $\mathbf{D}_e$, and do signature for the hash of $\mathbf{D}_e$, hence, all honest nodes can wait for $f+1$ signatures for $\mathbf{D}_e$ and output $block[1]$. At the start of epoch 2, according to the Algorithm 4, we know at least $f_1 + 1$ honest nodes have valid $block[1]$, every honest node of epoch 2 can wait for $f_1 + 1$ valid $block[1]$, hence, all honest nodes can output a $block[2]$. Similarly, at the start of epoch e, at least $f_{e-1} + 1$ honest nodes have valid $block[e-1]$, every honest node of epoch e can wait for $f_{e-1} + 1$ valid $block[e-1]$. Hence, all honest nodes of the new committee can output a $block[e]$.
- *Agreement.* Following the *ternimation* of DPID, then all honest nodes of the new committee can output a $block[e]$ at the epoch e , then following the validation function, all honest nodes output a valid $block[e]$.
- *Availability.* If an honest node of the new committee can output a $block[e]$, according to *agreement*, then all honest nodes of the new committee also can output a valid $block[e]$. So the availability is trival to obtain.
- *Correctness.* If an honest node can output a $block[e]$, then following the algorithm 4, the honest nodes can receive $f_{e-1} + 1$ valid $block[e-1]$, and these blocks contain same auxiliary information $\mathbf{D}_{e-1}$. From the validation function, we know at least one honest node signatures the auxiliary information. Suppose another honest node generates a $block'[e]$ with different $\mathbf{D}_e$, that means at least two different honest nodes generate two different $\mathbf{D}_{e-1}$ and $\mathbf{D}'_{e-1}$, then recursion implies that the agreement failed in epoch 1. It clearly is impossbile according to algorithm 1. Hence, any two honest nodes have the same $\mathbf{D}_e$. Then following the *availability* of DPID, any honest client can reconstruct some block B' with the same auxiliary information $\mathbf{D}_e$, which can make sure any two honest clients eventually reconstruct the same block B'. If the dealer was honest and input B, then any honest nodes of the new committee can reconstruct the same B, then recompute a new D_e that is related to B. hence, $B' = B$.
- *Integrity.* If any honest node of C_e can output a value, following the *correctness* of DPID, it means all honest nodes of C_e have the same $\mathbf{D}_e$. Then it also implies that all honest nodes of C_e recontrust a same $B' = M'$ (line 12 of algorithm 4). Besides, we also note that the C_e was generated from M' via a deterministic encode algorithm (line 13 of algorithm 4), hence, any honest clients can reconstructs a block $B'' = M'$. So $B' = B''$.

Complexities of DPID0**.** When we talk about complxity, we just consider the cost of during Refresh. Assuming ℓ is the bit-length of input and λ is the size of the security parameter. The cost mainly appears in three parts:

- ADD*:* The cost of ADD (Algorithm 10) is $O(n|m| + n^2\lambda)$. In DPID0, the size of m is $n\lambda$, then, the cost of ADD is $O(n^2\lambda)$.
- Fresh *message:* The size of Fresh message is $O(\ell/n + \lambda)$, hence, the total cost of Fresh message is $O(n\ell + n^2\lambda)$.
- Final *message:* The size of Final (signature) is $O(\lambda)$, so the total cost of Final message is $O(n^2\lambda)$.

In sum, the DPID0's communication complexity is asymptotically $O(n\ell + n^2\lambda)$. Besides, the size of *block* is $O(\ell/n + n\lambda)$, so the total storage cost is $O(\ell + n^2\lambda)$.

5 DPID1: ASYNCHRONOUS DYNAMIC-COMMITTEE PROACTIVE INFORMATION DISPERSAL WITH BETTER ASYMPTOTIC COMPLEXITY

In DPID0, the communication complexity of Refresh up to $O(n\ell + n^2\lambda)$, but the storage complexity just is $O(\ell + n^2\lambda)$, hence, there exists a $O(n)$ gap between communication and storage complexity. In this section, we would like to reduce the $n\ell$-term of communication complexity such that this term is independent of the number of nodes.

High level overview. In comparison to DPID0, the main difference of DPID1 is that it considers using the same technology as [16] to select a sub-committee from the new committee, where the size of the sub-committee is κ, and at least one node of the sub-committee is honest except with negligible probability. In this way, we can reduce the $n\ell$-term to $\kappa\ell$-term, achieving a data size term that is independent of $O(n)$.

Algorithm 5 Asynchronous DPID1 with epoch $e > 1$: Refresh subprotocol 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 CE($e$);
wait until $Sub.C_e := \{\mathcal{P}_{j_1}, \mathcal{P}_{j_2}, \cdots, \mathcal{P}_{j_\kappa}\} \leftarrow$ CE($e$)
if $\mathcal{P}_i \in C_{e-1}$ then                                  ▷ old committee $C_{e-1}$
    if Verify($i$, $block[e-1]$) = 1 then
        parse $block[e-1] := (m_i, \mathbf{D}_{e-1}, T_{e-1})$;
        ADD[$e$]($\mathbf{D}_{e-1}$); multicast Fresh($e$, $m_i$) to $Sub.C_e$;
if $\mathcal{P}_i \in Sub.C_e$ then                                ▷ sub committee $Sub.C_e$
    upon receiving Fresh($e$, $m_j$) from $\mathcal{P}_j (\in C_{e-1})$ for the first time do
        $T = T \cup \{(j, m_j)\}$
    wait until ADD[$e$] return $\mathbf{D}_{e-1}$ 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
            $M' \leftarrow$ Dec($T$); let $\{\overline{m_j}\}_{j \in [n_e]} :=$ Enc($M'$, $n_e$, $f_e$);
            let $\mathbf{D}_e := \{\mathcal{H}(\overline{m_1}), \ldots, \mathcal{H}(\overline{m_{n_e}})\}$;
            $\pi \leftarrow$ NIZK(($\mathbf{D}_{e-1}$, $n_{e-1}$, $f_{e-1}$, $\mathbf{D}_e$, $n_e$, $f_e$), $M'$);     ▷ see equation 1
            send Relay($\overline{m_j}$, $\mathbf{D}_e$, $\pi$) 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 ADD[$e$] return $\mathbf{D}_{e-1}$;
        upon receiving Relay($\overline{m_i}$, $\mathbf{D}_e$, $\pi$) from $\mathcal{P}_j (\in Sub.C_e)$ for the first time do
            parse $\mathbf{D}_e := \{h'_1, \cdots, h'_{ne}\}$;
            if NIZKVerify(($\mathbf{D}_{e-1}$, $n_{e-1}$, $f_{e-1}$, $\mathbf{D}_e$, $n_e$, $f_e$), $\pi$) = 1 then
                if $h'_i = \mathcal{H}(\overline{m_i})$ then $\sigma_i \leftarrow$ Signature($e$, $\mathcal{H}(\mathbf{D}_e)$);
                    multicast Final($e$, $\sigma_i$) to $C_e$; $flag \leftarrow 1$;
                    upon receiving Final($e$, $\sigma_j$) from $\mathcal{P}_j (\in C_e)$ for the first time do
                        if Verify($e$, $\mathcal{H}(\mathbf{D}_e)$, $(j, \sigma_j)$) =1 then $T_e \leftarrow T_e \cup \{j, \sigma_j\}$;
    upon $|T_e| = f_e + 1$ and $flag = 1$ do
        return $block[e] := (\overline{m_i}, \mathbf{D}_e, T_e)$.
```

Constructing the DPID1. The process of DPID1 protocol is the same as the DPID0 except the Algorithm 4, the formal description of DPID1 is shown in Algorithm 1,2,3,5. Here, we just describe the Algorithm 5. Specifically, we consider the old committee members just send a valid $block[e-1]$ to these subcommittee members instead of sending the $block[e-1]$ to the whole new committee members. Once the members of the subcommittee retrieve data that is consistent with the auxiliary information $\mathbf{D}_{e-1}$, then he encodes the data with new parameter (n_e, f_e), then he sends new fragment m_i along with new auxiliary information $\mathbf{D}_e$ to the corresponding node $\mathcal{P}_i$.

A potential question is some corrupted nodes of the subcommittee could use some other data M' which is not equal M to encode, and then generate the auxiliary information $\mathbf{D}'_e$ of M'. In this case, it is hard for the honest node to verify the fragment whether belongs to the original data M. Hence, we consider using NIZK to generate proof π to prove the new auxiliary information $\mathbf{D}_e$ and the old auxiliary information $\mathbf{D}_{e-1}$ are consistent for the same M. The detailed protocol proceeds as follows:

- Refresh after epoch 1: (Algorithm 5). First, the new committee members run the committee-election protocol and send any messages to all members (nodes) of the old committee and new committee, but the old committee members keep listening to all messages of the committee-election protocol. Once the old committee members receive the output $Sub.C_e$ from the committee-election subprotocol, then each node $\mathcal{P}_i$ of old committee first checks whether the local $block[e-1] := \{m_i, \mathbf{D}_{e-1}, T_{e-1}\}$ is valid or not, if yes, then invokes ADD[e] with $\mathbf{D}_{e-1}$ as input, and sends fragment to all nodes of subcommittee $Sub.C_e$ via Fresh(e, m_i).

  All nodes of $Sub.C_e$ wait for the output of ADD[e], then filter some invalid fragments from C_{e-1}. Once receiving $f_{e-1} + 1$ valid fragments, then generating data M' and recomputing $\mathbf{D}_e$. Besides, the nodes also use NIZK to generate a proof to prove $\mathbf{D}_{e-1}$ is the auxiliary information of M' under f_{e-1} and n_{e-1} and $\mathbf{D}_e$ is the auxiliary information of M' under f_e and n_e. Then the node sends Relay($\overline{m_j}, \mathbf{D}_e, \pi$) to correponding node $\mathcal{P}_j$ of C_e.

  For any honest node $\mathcal{P}_i$ of the new committee, once receiving a Relay($\overline{m_i}, \mathbf{D}_2, \pi$) message from the member of sub-committee $Sub.C_e$, and also he receives the output $\mathbf{D}_{e-1}$ from ADD[e], then verifies the Relay message is valid or not via NIZKVerify(($\mathbf{D}_{e-1}, n_{e-1}, f_{e-1}, \mathbf{D}_e, n_e, f_e$), π) and $\mathcal{H}(\overline{m_i}) = \mathbf{D}_e[i]$. If yes, then he will sign the hash of $\mathbf{D}_e$ to form a witness. If any node in C_e has the witness and has signed for $\mathbf{D}_e$, then output.

Security intuition of DPID1**:** The security intuition of DPID1 is similar with DPID0, the core point is the NIZK can make sure the two different auxiliary informations $\mathbf{D}_{e-1}$ and $\mathbf{D}_e$ were geneted by a same data M. Due to ADD, all honest nodes of the new committee have the auxiliary information $\mathbf{D}_{e-1}$, so $\mathbf{D}_e$ is unique. That is:

$$\begin{aligned} \pi \leftarrow \mathsf{NIZK}\{((\mathbf{D}_{e-1}, n_{e-1}, f_{e-1}, \mathbf{D}_e, n_e, f_e), M') :& \\ \{\overline{m_j}\}_{j \in [n_e]} &= \mathsf{Enc}(M', n_e, f_e) \\ \wedge\ \mathbf{D}_e &= \{\mathcal{H}(\overline{m_1}), \ldots, \mathcal{H}(\overline{m_{n_e}})\} \\ \wedge\ \{m_j\}_{j \in [n_{e-1}]} &= \mathsf{Enc}(M', n_{e-1}, f_{e-1}) \\ \wedge\ \mathbf{D}_{e-1} &= \{\mathcal{H}(m_1), \ldots, \mathcal{H}(m_{n_{e-1}})\}\}. \end{aligned} \tag{1}$$

When the $\mathbf{D}_e$ fixed, then any fragments of epoch e are also easy to verify.

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

- *committee election:* The cost of CE(e) is $O(n^2\lambda)$.
- *ADD:* The input size of ADD is $n\lambda$, then the cost of ADD is $O(n^2\lambda)$.
- Fresh *message:* The size of Fresh message is $O(\ell/n + \lambda)$, the total cost of Fresh message is $O(\kappa\ell + \kappa n\lambda)$.
- Relay *message:* The size of Relay is $O(\ell/n + n\lambda)$, the total cost of Relay message is $O(\kappa\ell + \kappa n^2\lambda)$.
- Final *message:* The size of Final (signature) is $O(\lambda)$, the total cost of Final message is $O(n^2\lambda)$.

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

6 DPID2: ASYNCHRONOUS DYNAMIC-COMMITTEE PROACTIVE INFORMATION DISPERSAL WITH OPTIMAL CASE

For information dispersal, we noted that the lower bound of information dispersal on the communication complexity is $\Omega(\ell + n^2)$ for storing a data (or file) [3] in dispersal phase, and the lower bound on storage cost is $\Omega(\ell)$, where the size of the data is ℓ. In DPID0 and DPID1, when $\ell > n^2\lambda$, the storage complexity just is $O(\ell)$, however, the communication complexity of refresh is $O(n\ell)$ or $O(\kappa\ell)$. Hence, can we design a new protocol such that it has optimal communication complexity $O(\ell)$ or a better communication performance than the DPID1 when the data (or file) size is enough large?

In this section, we consider an optimistic case, that is all nodes are honest and all messages have no delay. In this case, we presented DPID2, which achieved optimal communication complexity in the optimistic case while maintaining the same communication complexity as DPID1 in the other cases.

High level overview. Considering the optimistic case, we note that every honest node just stores a fragment at the end of epoch, the size of fragment is $O(\ell/n)$. Hence, in order to transfer the data to new committee from old committee, and at the same time keeping the communication complexity doesn't blow-up, we first choose to let every honest node of the old committee encode with himself fragment to generate new fragments during the refresh phase, and send the corresponding fragment to the corresponding node. Because the old committee could contain some corrupted nodes, every honest node of new committee just can receive $n_{e-1} - f_{e-1}$ new fragments from distinct nodes. However, we can't make sure these fragments from the same set of old committee members, hence, we need to use a consensus protocol MVBA to make sure these fragments from the same set of old committee members. Otherwise, if we let every honest node directly stores these new fragments as a new fragment, the availabilty will be failed. For example, old committee and new committee both have four nodes. $\mathcal{P}_i(\in C_{e-1})$ do $\{m_{i,1}, m_{i,2}, m_{i,3}, m_{i,4}\} \leftarrow \mathsf{Dec}(m_i, 4, 1)$. If $\mathcal{P}'_1 \in C_e$ received $\{m_{1,1}, m_{3,1}\}$ and $\mathcal{P}'_2 \in C_e$ received $\{m_{2,1}, m_{2,2}\}$, $\mathcal{P}'_1$ and $\mathcal{P}'_2$ both are honest at the start of epoch $e+1$, then the retrieval subprotocol will fail in the epoch $e+1$, becasue although $\{m_{1,1}, m_{2,1}\}$ can use to decode m_1, no other $m_i (i \neq 1)$ can be recasted, but when M can be recasted, it needs any two distinct m_i.

Once the MVBA output a set, if all the corresponding fragments were received for all nodes of the new committee, then they would directly generate a witness for auxiliary information. Otherwise, they will select a sub-committee $Sub.C_e$ to help all honest nodes of the new committee receive the related fragments.

In this way, we achieve the optimal communication complexity under the optimistic case, and the same communication complexity as the DPID1 in the worst case.

Constructing the DPID2. Essentially, every honest node does information dispersal with himself fragment as the input instead of sending the whole fragment to the nodes of the new committee (or subcommittee) during the Refresh phase of DPID2. A challenge is how to against the corrupted node use some other fragments to do information dispersal instead of the correct fragment. In DPID1, we use the data M to generate auxiliary information $\mathbf{D}_{e-1}$ under n_{e-1}, f_{e-1} and deterministic encode algorithm, and auxiliary information $\mathbf{D}_e$ under n_e, f_e and deterministic encode algorithm, we use the NIZK to bind $\mathbf{D}_{e-1}$ and $\mathbf{D}_e$. However, in DPID2, we use the data m_i to generate auxiliary information $\mathbf{d}_i$ under n_e, f_e, f_{e-2} and a deterministic algorithm, the data m_i can generate value $\mathbf{D}_{e-1}[i]$ under f_{e-2}, vector T_{e-1} and another deterministic algorithm, we use the NIZK to bind $\mathbf{d}_i$ and $\mathbf{D}_{e-1}[i]$ under n_e, f_e, f_{e-2} and T_{e-1}.

Algorithm 6 The DPID2 with epoch $e = 1$: ACVID dispersal subprotocol with sender $\mathcal{P}_s$ for party $\mathcal{P}_i$

```
Initialization: Let RBC[$e$] refer to one instance of the reliable broadcast protocol,
where $\mathcal{P}_s$ is the sender of RBC[$e$]; $T \leftarrow \{\}$;
if $\mathcal{P}_i = \mathcal{P}_s$ and received input $M$ then
    as the sender to invoke RBC[$e$] with input $M$;
upon receiving output $M$ from RBC[$e$] do
    if $M \neq \perp$ then
        let $\{m_j\}_{j\in[n_e]} := \mathsf{Enc}(M, n_e, f_e)$; $T_e = \{0\}$;
        $\mathbf{D}_e := \{\mathcal{H}(0, m_1), \ldots, \mathcal{H}(0, m_{n_1})\}$;
        $\sigma_i \leftarrow \mathsf{SigShare}_{2f_e+1}(e, 0, \mathcal{H}(\mathbf{D}_e), \mathcal{H}(T_e))$; multicast Final($e$, $\sigma_i$) to $C_e$;
        upon receiving Final($e$, $\sigma_j$) from $\mathcal{P}_j(\in C_e)$ for the first time do
            if $\mathsf{VerShare}_{2f_e+1}(e, 0, \mathcal{H}(\mathbf{D}_e), \mathcal{H}(T_e), (j, \sigma_j))$ =1 then
                $T \leftarrow T \cup \{j, \sigma_j\}$;
        upon $|T| = 2f_e + 1$ do $\Sigma \leftarrow \mathsf{Combine}_{2f_e+1}(e, 0, \mathcal{H}(\mathbf{D}_e), \mathcal{H}(T_e), T)$;
            return $block[e] := (0, m_i, \mathbf{D}_e, T_e, \Sigma)$.
```

Algorithm 7 The DPID2 retrieval subprotocol, with epoch e for party $\mathcal{P}_i$

```
Initialization: $T' \leftarrow \{\}$;
if $\mathcal{P}_i$ would like to retrieval 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(ID, $block[e]$) from $\mathcal{P}_j(\in C_e)$ for the first time do
    if Verify($j$, $block[e]$)=1 then
        parse $block[e] := (f_{e-1}, m_i, \mathbf{D}_e, T_e, \Sigma)$; $T' \leftarrow T' \cup \{j, m_j\}$;
upon $|T'| = f + 1$ do $M' \leftarrow \mathsf{DEC}(f_{e-1}, f_e, T')$;
    return $M'$;

function DEC($f_{e-1}$, $f_e$, $T$):
    if $f_{e-1} = 0$ then return Dec($T$)
    for any $\{i, m_i\} \in T$ do
        parse $m_i := m_{1,i} \parallel \cdots \parallel m_{f_{e-1}+1,i}$; $T_k \leftarrow T_k \cup \{i, m_{k,i}\}$
    if $T_k = f_e + 1$ then
        $m_k \parallel f_{e-2} \leftarrow \mathsf{Dec}(T_k)$; $T' \leftarrow T' \cup \{k, m_k\}$
    if $|T'| = f_{e-1} + 1$ then return DEC($f_{e-2}$, $f_{e-1}$, $T'$)
```

Algorithm 8 Validation function of DPID2, with epoch e

```
function Verify($i$, $block[e]$):
    if $block[e] \neq \emptyset$ then parse $block[e] := (f_{e-1}, m_i, \mathbf{D}_e, T_e, \Sigma)$;
        if $\mathsf{ThldVerify}_{2f_e+1}(e, f_{e-1}, \mathcal{H}(\mathbf{D}_e), \mathcal{H}(T_e), \Sigma) = 1$ then
            parse $m_i := m'_1 \parallel \cdots \parallel m'_{f_{e-1}+1}$; $T_e := \{j_1, \cdots, j_{f_{e-1}+1}\}$;
            if $\mathbf{D}_e[j] = \bigoplus\{\mathcal{H}(j_k, m'_k)\}_{k\in[f_{e-1}+1]}$ then return 1;
    else return 0.
```

The new fragment m_i of $\mathcal{P}_i$ was formed by the concatenation operator in $f_{e-1}+1$ new fragments, that is $m_i = m_{j_1,i} \parallel \cdots \parallel m_{j_{f_{e-1}+1},i}$; because $\mathbf{d}_{j_k} := \{\mathcal{H}(j_k, m_{j_k,i})\}_{i\in[n_e]}$ has been received by all honest nodes, in order to verify the validity of m_i, we let $\mathbf{D}_e[i] = \bigoplus\{\mathcal{H}(j_k, m_{j_k,i})\}_{k\in[f_{e-1}+1]}$ to make sure m_i is the unique corresponding element.

Now we describe our protocol. The formal description of the protocol is shown in Algorithm 6,7,8 and 9. The Algorithm 6,7 and

Algorithm 9 Asynchronous DPID2 with epoch $e > 1$: Refresh subprotocol for party $\mathcal{P}_i$: with subcommittee and with NIZK

```
    Initialization: $flag \leftarrow \emptyset$; $flag1 \leftarrow \emptyset$; $T \leftarrow \{\}$; $T_s \leftarrow \{\}$; $T_{share} \leftarrow \{\}$;
    $s \leftarrow 0$; Public input: $n_{e-1}, f_{e-1}, n_e, f_e$;
 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 $block[e-1] := (f_{e-2}, m_i, \mathbf{D}_{e-1}, T_{e-1}, \Sigma)$;
 4:     ADD$[e](f_{e-2}, \mathbf{D}_{e-1}, T_{e-1})$;
 5:     $\{m_{i,j}\}_{j\in[n_e]} :=$ Enc$(m_i \parallel f_{e-2}, n_e, f_e)$; $\mathbf{d}_i := \{\mathcal{H}(i, m_{i,j})\}_{j\in[n_e]}$;
 6:     $\pi_i \leftarrow$ NIZK$((f_{e-2}, n_e, f_e, \mathbf{D}_{e-1}[i], T_{e-1}, \mathbf{d}_i), m_i)$;      ▷ see equation 2
 7:     for each $\mathcal{P}_j \in C_e$ do send Fresh$(e, \mathbf{d}_i, \pi_i, m_{i,j})$ to $\mathcal{P}_j$;
                                                                      ▷ new committee $C_e$
 8: if $\mathcal{P}_i \in C_e$ then wait ADD$[e]$ return $(f_{e-2}, \mathbf{D}_{e-1}, T_{e-1})$;
 9:   upon receiving Fresh$(e, \mathbf{d}_j, \pi_j, m_{j,i})$ from $\mathcal{P}_j \in C_{e-1}$ for the first time do
10:     if NIZKVerify$((f_{e-2}, n_e, f_e, \mathbf{D}_{e-1}[j], T_{e-1}, \mathbf{d}_j), \pi_j) = 1$ then
11:       if $\mathcal{H}(j, m_{j,i}) = \mathbf{d}_j[i]$ then
12:         $\sigma_{j,i} \leftarrow$ SigShare$_{2f_e+1}(e, j, \mathcal{H}(\mathbf{d}_j))$; $T_s \leftarrow T_s \cup \{j\}$;
13:         multicast Vote$(j, \mathcal{H}(\mathbf{d}_j), \sigma_{j,i})$ to $C_e$;
14:       upon receiving Vote$(j, \mathcal{H}(\mathbf{d}_j), \sigma_{j,k})$ from $\mathcal{P}_k \in C_e$ for the first time do
15:         if VerShare$_{2f_e+1}(e, j, \mathcal{H}(\mathbf{d}_j), (k, \sigma_{j,k}))$ =1 then $T_j \leftarrow T_j \cup$
    $\{k, \sigma_{j,k}\}$;
16:           upon $|T_j| = 2f_e + 1$ do $\sigma_j \leftarrow$ Combine$_{2f_e+1}(e, j, \mathcal{H}(\mathbf{d}_j), T_j)$;
17:             $T \leftarrow T \cup \{j, \mathbf{d}_j, \sigma_j\}$;
18:   upon $|T| = f_{e-1} + 1$ do propose $T$ for the MVBA$[e]$
19:   upon the MVBA$[e]$ return $W = \{(j_k, \mathbf{d}_{j_k}, \sigma_{j_k})\}_{k\in[f_{e-1}+1]}$ do
20:     let $\mathbf{D}_e \leftarrow \{\bigoplus\{\mathbf{d}_{j_k}[r]\}_{k\in[f_{e-1}+1]}\}_{r\in[n_e]}$; $T_e = \{j_k\}_{k\in[f_{e-1}+1]}$;
21:     if $j_k \notin T_s$ for any $k \in [f_{e-1} + 1]$ then multicast CALLHELP$(e)$;
22:     else $flag \leftarrow 1$;
                              ▷ choose subcommittee $Sub.C_e$ from new committee $C_e$
23:   upon receiving CALLHELP$(e)$ from $\mathcal{P}_j \in C_e$ for the first time do
24:     Invoke Committee Election protocol CE$(e)$;
25:     wait until $Sub.C_e := \{\mathcal{P}_{z_1}, \mathcal{P}_{z_2}, \cdots, \mathcal{P}_{z_\kappa}\} \leftarrow$ CE$(e)$
26:     for any $k \in [f_1 + 1]$: $j_k \in T_s$ and $j_k \in T_e$ do
27:       if $\mathcal{H}(j_k, m_{j_k,i}) = \mathbf{d}_{j_k}[i]$ then $T_{data} \leftarrow T_{data} \cup \{j_k, m_{j_k,i}\}$;
28:     send HELP$(T_{data})$ to every $\mathcal{P}_j \in Sub.C_e$;
29:   if $\mathcal{P}_i \in Sub.C_e$ then                                       ▷ relay
30:     upon receiving HELP$(T')$ from $\mathcal{P}_j \in C_e$ for the first time do
31:       for any $\{j_k, m_{j_k,j}\} \in T'$ and $j_k \in T_e$ and $m_{j_k,j} \neq \perp$ do
32:         if $\mathbf{d}_{j_k}[j] = \mathcal{H}(j_k, m_{j_k,j})$ then $T_3(j_k) = T_3(j_k) \cup \{(j, m_{j_k,j})\}$;
33:     for any $j_k \in T_e$: upon $|T_3(j_k)| = f_e + 1$ do $m_{j_k} \parallel f_{e-2} \leftarrow$ Dec$(T_3(j_k))$;
34:       $\{m_{j_k,j}\}_{j\in[n_e]} :=$ Enc$(m_{j_k} \parallel f_{e-2}, n_e, f_e)$; $s = s + 1$;
35:     upon $s = |T_e| = f_{e-1} + 1$ do for any $j \in [n_e]$;
36:       let $value_j \leftarrow m_{j_1,j} \parallel \cdots \parallel m_{j_{f_{e-1}+1},j}$; send Store$(value_j)$ to $\mathcal{P}_j$;
                                                                      ▷ add proof
37:   upon $flag = 1$ or receiving Store$(value_i)$ from $\mathcal{P}_j \in Sub.C_e$ do
38:     let $value_i := m_{j_1,i} \parallel \cdots \parallel m_{j_{f_{e-1}+1},i}$;
39:     if $\mathbf{D}_e[i] = \bigoplus\{\mathcal{H}(j_k, m_{j_k,i})\}_{k\in[f_{e-1}+1]}$ then
40:       $m_i \leftarrow value_i$; $\sigma_i \leftarrow$ SigShare$_{2f_e+1}(e, f_{e-1}, \mathcal{H}(\mathbf{D}_e), \mathcal{H}(T_e))$;
41:       multicast Final$(e, \sigma_i)$; $flag1 \leftarrow 1$;
42:   upon receiving Final$(e, \sigma_j)$ from $\mathcal{P}_j \in C_e$ for the first time do
43:     if VerShare$_{2f_e+1}(e, f_{e-1}, \mathcal{H}(\mathbf{D}_e), \mathcal{H}(T_e), (j, \sigma_j))$ =1 then
44:       $T_{share} \leftarrow T_{share} \cup \{j, \sigma_j\}$;
45:   upon $|T_{share}| = 2f_e + 1$ do
46:     $\Sigma \leftarrow$ Combine$_{2f_e+1}(e, f_{e-1}, \mathcal{H}(\mathbf{D}_e), \mathcal{H}(T_e), T_{share})$;
47:     wait $flag1 = 1$ return $block[e] := (f_{e-1}, m_i, \mathbf{D}_e, T_e, \Sigma)$.
```

8 of DPID2 are similar with the Algorithm 1,2 and 3 of DPID0. As a result, we only describe the detailed proceeds of Algorithm 9.

- Phase 1: For any node $\mathcal{P}_i$ in the old committee: first, node $\mathcal{P}_i$ checks its local *block*$[e-1]$, if it is valid, then invokes ADD$[e]$ with $(f_{e-2}, \mathbf{D}_{e-1}, T_{e-1})$ as input, and then computing the fragments with input $(m_i \parallel f_{e-2}, n_e, f_e)$ by the encoding algorithm to output $\{m_{i,j}\}_{j\in[n_e]}$. Then $\mathcal{P}_i$ generates an auxiliary information $\mathbf{d}_i$ for $\{m_{i,j}\}_{j\in[n_e]}$, at the same time, $\mathcal{P}_i$ invokes NIZK to bind $\mathbf{d}_i$ and $\mathbf{D}_{e-1}[i]$, then sends Fresh$(e, \mathbf{d}_i, \pi_i, m_{i,j})$ to $\mathcal{P}_j (\in C_e)$.
- Phase 2: For any node $\mathcal{P}_i$ of the new committee: first, waiting for ADD$[e]$ to return $(f_{e-2}, \mathbf{D}_{e-1}, T_{e-1})$; then if he receives any valid Fresh$(e, \mathbf{d}_j, \pi_j, m_{j,i})$ message from the old committee member $\mathcal{P}_j$, then do a threshold signature for $(e, j, \mathcal{H}(\mathbf{d}_j))$; and multicast a Vote$(j, \mathcal{H}(\mathbf{d}_j), \sigma_{j,i})$ message to new committee to announce that he receives a valid fragment from old committee member $\mathcal{P}_j$. If at least $2f_e + 1$ honest nodes announced for $\mathcal{P}_j$, then adds $\{j, \mathbf{d}_j, \sigma_j\}$ into T. Once the size of T is $f_{e-1} + 1$, then invokes MVBA$[e]$ with input T.
- Phase 3: Once the MVBA$[e]$ returns $\{(j_k, \mathbf{d}_{j_k}, \sigma_{j_k})\}_{k\in[f_{e-1}+1]}$. First, all honest nodes use the $\{\mathbf{d}_{j_k})\}_{k\in[f_{e-1}+1]}$ to prodcue $\mathbf{D}_e$. Then, for any node $\mathcal{P}_i$ of the new committee, if $\mathcal{P}_i$ receives all fragments from the correponding old committee members, $\mathcal{P}_i$ will directly skip to line 37 to do threshold signature to produce a witness Σ.

  If some nodes find they have some fragments that have not been received, then they will multicast a CALLHELP(e) to new committee; if some honest node of new committee receives a CALLHELP(e), then he will invoke committee election subprotocol to select a sub-committee from the whole new committee such that the subcommittee contains at least one honest node. After that, all honest nodes will send all the related fragments that they have received to the subcommittee members. The subcommittee members will retrieval the orignal data, and do encode on it with some publich parameter, then send the corresponding fragment $value_i$ to $\mathcal{P}_i (\in C_e)$. When $\mathcal{P}_i$ receives the correct fragment, $\mathcal{P}_i$ will do threshold signature to produce a witness Σ.

Security intuition of DPID2**:** The security intuition of DPID2 is similar to DPID1, the main point is we use the NIZK to bind $\mathbf{d}_i$ and $\mathbf{D}_{e-1}[i]$ under f_{e-2}, n_e, f_e, and T_{e-1} to guarantee $\mathbf{d}_i$ and $\mathbf{D}_{e-1}[i]$ is generated by a same message m_i, that is:

$$\begin{aligned} \pi_i \leftarrow \text{NIZK}(((f_{e-2}, n_e, f_e, \mathbf{D}_{e-1}[i], T_{e-1}, \mathbf{d}_i), m_i) : & \\ m_i = m'_1 \parallel \cdots \parallel m'_{f_{e-2}+1} & \\ \wedge\ T_{e-1} = \{j_1, \cdots, j_{f_{e-2}+1}\} & \\ \wedge\ \mathbf{D}_{e-1}[i] = \bigoplus\{\mathcal{H}(j_k, m'_k)\}_{k\in[f_{e-2}+1]} & \\ \wedge\ \{m_{i,j}\}_{j\in[n_e]} = \text{Enc}(m_i \parallel f_{e-2}, n_e, f_e) & \\ \wedge\ \mathbf{d}_i = \{\mathcal{H}(i, m_{i,j})\}_{j\in[n_e]}). & \end{aligned} \tag{2}$$

Complexities of DPID2**:** The DPID1's communication complexity is asymptotically $O(\kappa\ell + n^2\lambda)$, the cost mainly appears in five parts:

- ADD*:* The input size of ADD is $n\lambda$, then the cost of ADD is $O(n^2\lambda)$.
- Fresh *message:* The size of Fresh message is $O(\lambda)$, the total cost of Fresh message is $O(\ell + n^3\lambda)$.
- Vote *message:* The size of Vote message is $O(\ell/n^2 + n\lambda)$, the total cost of Vote message is $O(n^3\lambda)$.
- MVBA*:* The cost of MVBA [18] is $O(n|m| + n^2\lambda)$. In DPID2, the size of m is $n^2\lambda$, then, the cost of MVBA is $O(n^3\lambda)$.
- *committee election:* The cost of CE(e) is $O(n^2\lambda)$.
- Help *message:* The size of Help is $O(\ell/n + n\lambda)$, the total cost of Help message is $O(\kappa\ell + \kappa n^2\lambda)$.
- Store *message:* The size of Store is $O(\ell/n)$, the total cost of Store message is $O(\kappa\ell)$.

- Final *message:* The size of Final (signature) is $O(\lambda)$, the total cost of Final message is $O(n^2\lambda)$.

In sum, (1). for the optimistic case, it doesn't contain *committee election*, send Help and Store messages, so the communication complexity of DPID2 is $O(\ell + n^3\lambda)$. (2). for the worst case, it contains all parts, so the communication complexity of DPID2 is $O(\kappa\ell + n^3\lambda)$.

7 APPLICATION TO DYNAMIC-COMMITTEE PROACTIVE SECRET SHARE

Dynamic-committee Proactive secret share (DPSS) has made great 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 $O(n^3\ell + n^3\lambda)$ [25, 29]. Clearly, these works are suboptimal for the data itself is large. Especially, the size of stored data itself is large for the data storage system.

In this section, we can instantiate our DPID by any state-of-the-art DPSS to get a better DPSS protocol, the asymptotic performance of the new DPSS was shown in table 2. For each epoch, we consider all nodes will parallel execute two protocols: DPID and DPSS, where the input of DPID is encrypted data, and the input of DPSS is the encryption key. In this way, we can hugely reduce communication costs.

REFERENCES

[1] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. 2019. Asymptotically optimal validated asynchronous byzantine agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. 337–346.

[2] Andreea B Alexandru, Erica Blum, Jonathan Katz, and Julian Loss. 2023. 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*. Springer, 681–710.

[3] Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. 2022. Asynchronous Verifiable Information Dispersal with Near-Optimal Communication. *Cryptology ePrint Archive* (2022).

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

[5] Joshua Baron, Karim El Defrawy, Joshua Lampkins, and Rafail Ostrovsky. 2016. 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*. Springer, 23–41.

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

[7] Vitalik Buterin et al. 2014. A next-generation smart contract and decentralized application platform. *white paper* 3, 37 (2014).

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

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

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

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

[12] Yvo Desmedt and Sushil Jajodia. 1997. *Redistributing secret shares to new access structures and its applications.* Technical Report. Citeseer.

[13] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. 2022. 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*. Springer, 252–282.

[14] Jens Groth. 2006. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *Advances in Cryptology–ASIACRYPT 2006: 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006. Proceedings 12*. Springer, 444–459.

[15] Christoph U Günther, Sourav Das, and Lefteris Kokoris-Kogias. 2022. Practical Asynchronous Proactive Secret Sharing and Key Refresh. *Cryptology ePrint Archive* (2022).

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

[17] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. 1995. 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*. Springer, 339–352.

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

[19] Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. 2019. CHURP: dynamic-committee proactive secret sharing. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2369–2386.

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

[21] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

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

[23] Matthieu Rambaud and Antoine Urban. 2022. Proactive Secret Sharing over Asynchronous Channels under Honest Majority (with Ephemeral Roles): Refreshing Without a Consistent View on Shares. *Cryptology ePrint Archive* (2022).

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

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

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

[27] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.

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

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

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

8 APPENDIX

Asynchronous Data Dissemination (ADD**)**: the ADD in Algorithm 10 is the minor revise of [10], the revised ADD has the same communication performance as [10], where the message complexity is $O(n^2)$ and the communication complexity is $O(n|m| + n^2\lambda)$ in expectation assuming the size of input is $|m|$.

Algorithm 10 The ADD with epoch e for party $\mathcal{P}_i$

```
Public input: $n_{e-1}$, $f_{e-1}$, $C_{e-1}$, $n_e$, $f_e$, $C_e$;
input $M_i$: either $M_i = M$ or $M_i = \perp$;
                                                        ▷ old committee $C_{e-1}$
if $\mathcal{P}_i \in C_{e-1}$ and $M_i \neq \perp$ then
    Let $M' = \{m_1, \cdots, m_{n_e}\} := \mathsf{RSEnc}(M_i, n_e, f_e)$;
    send $\mathrm{Disp}(m_j)$ to $\mathcal{P}_j \in C_e$ for every $j \in [n_e]$;    ▷ dispersal
                                                        ▷ new committee $C_e$
if $\mathcal{P}_i \in C_e$ then
    upon receiving $f_{e-1} + 1$ identical $\mathrm{Disp}(m_i)$ from distinct nodes of $C_{e-1}$ do
        $m'_i \leftarrow m_i$;
        multicast $\mathrm{Reconst}(m'_i)$ to $C_e$;
    Let $T := \{\}$;                                    ▷ reconstruction
    upon receiving $\mathrm{Reconst}(m'_j)$ from $\mathcal{P}_j \in C_e$ for the first time do
        $T \leftarrow T \cup (j, m'_j)$;
    for $0 \le r \le f_e$ do                            ▷ Online Error Correction
        wait $|T| \ge 2f_e + r + 1$
            let $p_r(\cdot) = \mathsf{RSDec}(f_e, r, T)$;
            if $2f_e + 1$ elements $(j, m'_j) \in T$ satisfy $p_r(j) = m'_j$ then
                let the coefficients of $p_r(\cdot)$ as $M$;
                return $M$.
```