# Shanrang: Fully Asynchronous Proactive Secret Sharing with Dynamic Committees

Yunzhou Yan[1,3], Yu Xia[2,3], and Srinivas Devadas[2]

[1] Tsinghua University yanyz18@mails.tsinghua.edu.cn
[2] MIT CSAIL {yuxia, devadas}@mit.edu
[3] Equal Contribution

**Abstract.** We present Shanrang, the first fully asynchronous proactive secret sharing scheme with dynamic committee support. Even in the worst possible network environment, where messages could have arbitrary latencies, Shanrang allows a dynamic committee to store a secret and periodically refresh the secret shares in a distributed fashion. When the committee changes, both the old committee and the new committee jointly refresh and transfer the shares to the new committee, without revealing the secret to the adversary.
With $n$ parties, Shanrang tolerates $n/4$ Byzantine faults and maintains liveness as long as the messages are delivered. In contrast to prior work, Shanrang makes no assumptions on the network latency. Designing an asynchronous protocol is challenging because it is impossible to distinguish an adversary sending no messages from an honest party whose messages have not arrived yet. We evaluated Shanrang on geographically distributed machines and we found Shanrang achieved 200 seconds for handing off between 2 committees of 41 parties. Shanrang requires $O(\lambda n^3 \log n)$ messages and runs in expected $O(\log n)$ rounds for every handoff. To show Shanrang is robust even in a harsh network environment, we test Shanrang on the Tor network and it shows robust performance.

**Keywords:** Proactive Secret Sharing · Asynchronous Protocols · Distributed Protocols.

## 1 Introduction

Secure key management in open distributed systems has been a long-standing challenge. Loss of private keys might result in catastrophic consequences. For example, the New York Times reports that over hundreds of billions worth of Bitcoin are stored in lost or stranded wallets [1]. What's more, the CEO of Quadriga, one of the largest cryptocurrency exchanges in Canada, reportedly lost the exchange's only private key and, as a consequence, all the assets are no longer retrievable.

People have made many efforts to mitigate this situation. A standalone cold wallet stores the private keys offline in dedicated storage; special hardware is considered a very safe option. However, cold wallets incur significant cost and are usually hard to use. Another popular option is to delegate the private key storage to a distributed key management service. To the best of our knowledge, the closest solution to secure private key delegation in an open blockchain network is CHURP [10], a churn-robust proactive secret sharing scheme. CHURP allows the parties to have a dynamic subset, namely, a committee, hosting a secret. Periodically, the parties elect a new committee, and the new committee together with the old committee performs a handoff protocol to transfer and refresh the secret shares. CHURP tolerates network churns and allows the threshold to change (namely, the threshold is the number of parties required to work together in order to recover the secret).

However, CHURP assumes a synchronous network environment, where the network message latency has a constant upper bound known by the parties. In reality, network latency might diverge greatly from one time period to the next. Although a protocol can always assume a sufficiently large latency bound, when the Byzantine adversaries are present, a large time bound results in a longer waiting timeout, which harms the performance and practicality of the scheme. Besides, network outages and instability are unpredictable. When the network assumption fails, the protocol loses its guarantee on liveness and other properties. For critical services like financial institutes and distributed consensus governance, robustness against network conditions is among the top priorities.

In an asynchronous network, we have no assumptions on the message latencies. The key challenge in designing asynchronous protocols is that it is impossible for a party $P_i$ to distinguish between the following two cases regarding another party $P_j$ when $P_i$ is expecting a message form $P_j$ but $P_i$ has not received it:

– (1) The party $P_j$ sends a message honestly, but it has not yet arrived due to the network latency;
– (2) The party $P_j$ is actually an adversary, and does not send anything.

Waiting for such a message is dangerous in an asynchronous protocol because the second case can simply stop $P_i$ from making progress. Whenever we let parties *broadcast* to all the parties in the protocol, in the next step we can at most let an honest party expect $n - t$ messages. Otherwise, it might get stuck due to an adversarial behavior. With this rule in mind, we utilize several building blocks from prior works, including asynchronous common subsets, asynchronous binary agreement, and asynchronous byzantine broadcasting to maintain liveness.

In this work, we present Shanrang, the first asynchronous proactive secret sharing protocol with dynamic committees and Byzantine-fault tolerance support. Compared to CHURP, Shanrang assumes absolutely no assumptions on the network latency. Shanrang makes progress as long as the messages are delivered. This makes Shanrang able to survive the worst possible network environment. In Section 6, we evaluate Shanrang on the Tor relay network because the Tor network has considerably large variance of latency. Besides the strong robustness property, Shanrang is also efficient and Byzantine fault tolerant. Overall, Shanrang provides a secure scheme to delegate secrets like private keys to a dynamically changing committee. The paper makes the following contributions:

- We present Shanrang, the first asynchronous proactive secret sharing protocol with dynamic committees and Byzantine-fault tolerance support.
- We prove formally the properties of Shanrang and analyze the asymptotic complexity of running time and communication cost.
- We perform comprehensive evaluations of Shanrang on geographically distributed machines on Amazon EC2. We also conduct experiments on the Tor relay network, where the connections are unstable and the latency varies greatly.

We first present the background, related works, and building blocks in Section 2. We discuss a new primitive, the multi-value asynchronous common subset protocol in Section 3. Then, we describe the Shanrang protocol in details in Section 4. Next, we formally define the properties and prove them in Section 5. We describe our evaluations in Section 6. Lastly, we conclude our work in Section 7. We also include discussion on an extension of the KZG [9] commitment in Appendix.

## 2 Background

*Secret sharing protocols* [15] allow a number of parties to jointly share a secret to $n$ parties such that an adversary has to compromise at least $t$ parties to recover the secret, where $n$ and $t$ are parameters. Secret sharing is a critical building block in a number of secure distributed protocols like key management and access control. Secret sharing provides a way to avoid a single point of failure in terms of secret handling. However, in the real world, adversaries are usually able to corrupt more parties if given enough time. People model such adversaries as *mobile adversaries*. With this kind of adversaries, sooner or later, any simple secret sharing scheme will leak the secret due to enough corruptions made by the adversaries. *Proactive Secret Sharing* [8] refreshes the secret with new randomness periodically to defend it against *mobile adversaries* [17], adversaries that can compromise more parties with more time. Mobile adversaries are not bounded by the total number of parties they can compromise, but rather the rate of compromising. This provides a way for distributed secret handling in long-running systems.

With the outgrowth of blockchains and cryptocurrency systems, permissionless networks pose several new challenges for distributed protocols. In contrast to *permissioned networks*, which assumes a public key infrastructure for the parties involved and the parties cannot join or leave freely, *permissionless networks* allow nodes to join and leave at any time, and network churns could happen. The parties handling the secret might also become unavailable over time. Therefore, in such a context the protocol needs to support dynamic committee changes. Mobile Proactive Secret Sharing (MPSS) allows dynamic change of the parties holding the secret shares. Additionally, MPSS allows the threshold to change when the shares transfer to the new group. CHURP [10] presents a churn-tolerant proactive secret sharing protocol with the support of dynamic committees. However, CHURP assumes synchronous networks and is prone to unstable network latency.

### 2.1 Related Works

Herzberg et al. proposed the notion of Proactive Secret Sharing [8] (PSS) for the first time, where the secret is periodically renewed and the information learned by the adversary between the refreshment is useless.

Asynchronous Verifiable Secret Sharing [4] by Cachin et al. achieves verifiable secret sharing in asynchronous networks. The authors also include an asynchronous proactive secret sharing protocol based on their verifiable secret sharing protocol. VPSS [13] is another early attempt on verifiable proactive secret sharing scheme. VPSS is based on Shamir's secret sharing. Similarly, Asynchronous Proactive Secret Sharing [17] (APSS) is another early construction of a proactive secret sharing for asynchronous networks. The protocol allows dynamic committees, but the secret shares are exponentially large in terms of the size of the committee.

In some use cases, we want to have a dynamic group of nodes holding the secret share. The reader might ask why not simply launch a new instance of APSS and homomorphically add them to local shares. The answer is in the threshold. If we assume at most $t$ parties in a committee might be adversarial, when both of the old committee and the new committee are present, there might be $2t$ adversarial nodes involved. The tolerance needs to double during the handoff.

Mobile Proactive Secret Sharing [14] (MPSS) is based on [8] and allows the committee to evolve along with time. MPSS uses PBFT [6] to reach a consensus on the old committee shares and incurs an expensive message complexity of $O(n^4)$. Therefore, MPSS is subject to the constraints of PBFT that the network has to be partially synchronous. In Shanrang, we use *asynchronous common subset* protocols to reach consensus on the interpolation points on the old committee, which is not only fully asynchronous, but also more efficient in terms of message complexity.

CHURP [10], is our counterpart in the synchronous setting. CHURP achieves churn-robust proactive secret sharing with dynamic committees. It assumes the network to be synchronous, namely, a known upper bound of message delivery latency. Meanwhile, Shanrang makes no assumption on the message delivery. The liveness of Shanrang guarantees that as long as messages deliver, the protocol makes progress.

HBACSS [16] is a fully asynchronous complete secret sharing protocol. Complete secret sharing provides an additional property that all honest parties are guaranteed to have correct shares. This is useful for MPC applications.

Figure 1 shows a preliminary comparison of related works and Shanrang. Among the protocols listed, Shanrang is the only one that supports proactive properties, asynchronous networks, and mobile committees, and without incurring an exponentially large message complexity.

| | **Shanrang** | PSS | MPSS | AVSS[*] | APSS | VPSS | Churp | HBACSS |
|---|---|---|---|---|---|---|---|---|
| Proactive | **Y** | Y | Y | Y | Y | Y | Y | N |
| Network | **Async** | Sync | Partial-Sync | Async[**] | Async | Async[**] | Sync | Async |
| Mobile Committee | **Y** | N | Y | N | Y | N | Y | N |
| Message Complexity | $O(\lambda n^3 \log n)$ | $O(n^2)$ | $O(n^4)$ | $O(n^4)$ | $\exp(n)$ | $O(n^2)$ | $O(n^2)$ optimal $O(n^2)$ on-chain pessimistic | $O(n)$ |
| Threshold | $t < n/4$ | $t < n/2$ | $t < n/3$ | $t < n/3$ | $t < n/3$ | $t < n/3$ | $t < n/2$ | $t < n/3$ |

**Fig. 1.** Comparisons of related work

$*$ We refer to the proactive secret sharing protocol presented in the AVSS paper [4].

$**$ Adversaries have to send messages in their models.

## 2.2 Building Blocks

In this section, we discuss several building blocks that Shanrang relies on, namely, from bottom up, the asynchronous binary agreement protocol, the asynchronous common subset protocol, and the asynchronous atomic broadcast protocol. These protocols provide essential functionalities for agreeing on a random polynomial to refresh the secrets in an asynchronous network. We also need the polynomial commitment protocol to make sure the polynomials are well-formed.

*Asynchronous Binary Agreement (ABA)* is an asynchronous protocol that allows a number of parties to reach a consensus on a binary value. It is a popular primitive in distributed consensus protocols like the *multi-value agreement* protocol.

Informally, at the beginning of the protocol, every party $P_i$ has an input $x_i \in \{0, 1\}$. Finally, every honest party outputs the same bit $b$ and $b \in \{x_i\}$. An asynchronous binary agreement protocol satisfies the following properties.

- **Agreement** If a honest party outputs a bit $b$, and another honest party outputs a bit $b'$, then $b = b'$.
- **Liveness** Eventually, every honest party outputs a bit $b$.
- **Soundness** If any honest party outputs a bit $b$, there exists a party index $i$ such that the input $x_i = b$.

In Shanrang, we instantiate the asynchronous binary agreement with a Byzantine fault tolerating protocol from Mostefaoui et al. [12]. This ABA protocol runs in expected $O(\log n)$ asynchronous rounds (we roughly measure the asynchronous rounds by the number of distributed random coins).

*Asynchronous Common Subset (ACS) [2]* In an asynchronous common subset protocol, parties can propose a single bit (either 0 or 1) as the input to the protocol. If enough parties have proposed, the honest parties eventually will agree on a common subset of the parties who have proposed 1. More formally,

- **Agreement** If a honest party outputs $V$, eventually every honest party outputs $V$.

– **Liveness** If at least $n-t$ honest nodes proposed, every honest party outputs a vector $V$.
– **Soundness** If any honest party outputs $V$, then $V$ includes at least $n-2t$ honest parties, and $|V| \geq n-t$.

*Asynchronous Atomic Broadcast [11]* is an asynchronous protocol that allows $n$ parties to reach consensus on a sequence of outputs. If a value is input to enough honest parties, it will be output by every honest party. Formally,

– **Agreement** If any honest party outputs a value $v$, then every honest party outputs $v$
– **Total Order** For every two honest parties $P_i$ and $P_j$, either the output of $P_i$ is a prefix of the output of $P_j$, or the output of $P_j$ is a prefix of the output of $P_i$.
– **Censorship Resilience** If a value $v$ is input to $n-t$ honest parties, every honest party outputs $v$ eventually.

*Polynomial Commitment* is a useful primitive in cryptography that binds a prover to a fixed polynomial and allows a verifier to efficiently check an evaluation of the polynomial at a particular position.

In Shanrang, we instantiate the polynomial commitment scheme with the Kate, Zaverucha and Goldberg (KZG) scheme [9].

## 3 New tool: Asynchronous Multi-Variate Common Subset

Before we dive into the description of the Shanrang protocol, we define a new primitive that we will be using in the main Shanrang protocol. This protocol is in fact constructed in the Honeybadger BFT protocol [11], but the authors called it an asynchronous common subset protocol. We want to distinguish this from the strict definition of ACS in [2].

*Asynchronous Multi-Variate Common Subset (AMVCS)* allows a number of parties to efficiently agree on a subset of a number of parties and their proposals. AMVCS is closely-related to *asynchronous common subset* [2] and Asynchronous Atomic Broadcast [11].

Different from an ACS protocol, AMVCS allows a party to propose an arbitrary message rather than a single bit value. After enough honest parties input their messages, all honest parties eventually will reach a consensus on a subset of the proposals.

We further distinguish the conceptually two different groups of parties in the ACS protocol, the *input group* and the *participating group*. The input group are those parties who both provide inputs to the protocol and participate in deciding the common subset. In contrast, the participating parties do not input to the protocol but only participate.

Compared to the Asynchronous Atomic Broadcast protocol, an Asynchronous Multi-Value Common Subset protocol does not care about the sequential order of the outputs as it only outputs a set. However, AMVCS puts a (in some sense) stronger constraint on the common subset that, every honest party sees the same subset, while an atomic broadcast protocol only requires a non-trivial prefix of all the output sequences to be the same. Moreover, not all parties propose values. The input group and participating group are decoupled. AMVCS does not have the notion of resilience tolerance as all the parties might propose different values.

More formally, AMVCS protocols satisfy the following properties.

- **Agreement** If a honest party outputs $V$, eventually every honest party outputs $V$.
- **Liveness** If at least $n-t$ honest nodes proposed, every honest party outputs a vector $V \subseteq M$, where $M = \{m_1, \ldots, m_{\geq n-t}\}$ is the set of values proposed by the parties (including adversaries who have proposed).
- **Soundness** If any honest party outputs $V$, then $V$ includes at least the proposals from $n-2t$ honest parties, and $|V| \geq n-t$.

To instantiate a AMVCS protocol, we use a similar approach in Honeybadger BFT [11]. The protocol consists of two parts: a Reliable Broadcast protocol (RBC) and an asynchronous common subset protocol. In the beginning, all the parties in the input group broadcast the value with RBC to all the participating parties. All the participating parties set up the ACS protocol. Note that, every input party is also a participating party. Whenever a party receives the output $m_i$ of an input party $P_i$ from the RBC protocol, it enters a bit 1 to the $i$-th channel of the ACS protocol if it has not done so. Finally, when the ACS outputs a common subset $s \subseteq [n]$, the participating party inputs a bit 0 to all the rest channels (i.e., whose index is not in the subset $s$), and outputs $\{m_i | i \in s\}$ as the output subset of AMVCS.

## 4 Protocol

In this section, we present the details of Shanrang protocol.

### 4.1 Model and Assumptions

In Shanrang, we assume point-to-point communication channels between parties where the message latency is decided by the adversary.

We follow the same epoch-based proactive secret sharing model with CHURP [10]. As with CHURP, Shanrang is also a long-running cryptographic protocol. The timeline is divided into epochs of variable length. At the beginning of each epoch, Shanrang proactivize the secret by running a *handoff* protocol that transfers the secret to a new commitee and refreshes the secret shares at the same time. When the *handoff* protocol finishes, the new committee holds the secret and can participate in any secure queries regarding the secret (e.g., an MPC protocol).

More formally, for epoch number $e > 0$, we denote $C^{(e)}$ as the committee of epoch $e$ (how to elect such a committee in a robust way is orthogonal to our work). Shanrang puts no restrictions between any adjacent committees. They can be totally the same, or completely disjoint. We follow the same adversary model as CHURP except that in Shanrang, we additionally allow the adversary to determine the message latency. We limit the adversary to only control up to $t^{(e)}$ parties in the committee $C^{(e)}$. While controlling a party, the adversary can also observe the memory of that party and send arbitrary messages on behalf of that party. For simplicity, we assume $|C^{(e)}| = n$ and $t^{(e)} = t$ for all $e > 0$. And we denote the $i$-th party in the $e$-th committee as $P_i^{(e)}$, holding a secret share $s_i^{(e)}$. In theory, Shanrang can support committees of different sizes and different corruption thresholds. Namely, the threshold can change in different epochs.

At the beginning of the protocol, we assume the first parties hold shares about a secret $s$. In each handoff protocol, the old committee $C^{(e-1)}$ and the new committee $C^{(e)}$ together secretly use the old shares $\{s_i^{(e-1)}\}$ as private inputs, calculate the new shares $\{s_i^{(e)}\}$, and dispatch them to the correct party $P_i^{(e)}$ with the invariant that, upon requests, any $t+1$ parties can recover the secret $s$.

Same as CHURP, Shanrang is also churn-tolerant. A proportion of parties can leave the committee at any time, but new parties can only join the new committee before a handoff protocol. We denote $C_{alive}^{(e-1)}$ as the alive parties in the old committee. We also assume that each party in the committee has a public key known to all the parties in the whole system. This is to make sure that the point-to-point communication is authenticated. We also assume each pair of parties has setup a symmetric key (e.g. via D-H exchange). Alternatively, we can always use the public keys to encrypt the messages but we assume symmetric encryptions are much faster.

In terms of the adversary tolerance budget, Shanrang can only tolerate 1/3 of the previous alive parties, and 1/4 of the parties in the new committee. Namely, $t \leq \min(\lfloor |C_{alive}^{(e-1)}|/3 \rfloor, \lfloor |C^{(e)}|/4 \rfloor)$.

## 4.2 Inherited Techniques from CHURP

Same as CHURP, Shanrang uses bivariate polynomials $B(x, y)$ s.t. $B(0, 0) = s$ to generate the shares. Each share is a univariate polynomial, either $B(x, i)$ or $B(i, y)$ where $i$ is the index of the party.

During a handoff phase, the parties refresh the shares by jointly sampling a new bivariate polynomial $Q(x, y)$ s.t. $Q(0, 0) = 0$. This is equivalent to re-sharing a secret of 0. Each party adds the share of $Q$ to its previous share of $B$. It is easy to see that the new shares correspond to a new polynomial $B'(x, y) = B(x, y) + Q(x, y)$, and therefore, $B'(0, 0) = B(0, 0) = s$.

Shanrang also inherits the dimension switching technique from CHURP that, by using a degree $(t, 2t)$ polynomial $B(\cdot, \cdot)$, the parties can switch to a higher threshold $2t$ by switching the sharing dimension of $B$, namely, a share change from $B(i, y)$ to $B(x, i)$. This is important because both committees are present during a handover, and the adversary can control up to $2t$ parties at this moment.

Like CHURP, Shanrang also assumes a pessimistic path. But we trigger the pessimistic path only when we found an active adversarial behavior. Shanrang handles network events like high latency surges well in the optimistic case. We omit the description of the pessimistic path in this work since if triggered, it is identical to that in CHURP.

## 4.3 Overview

In this subsection we give an overview of Shanrang protocol. During the handoff phase, the parties first perform *dimension reduction* to switch from threshold $t$ to threshold $2t$. Then, the parties perform the *proactivation* to refresh the shares by jointly sampling a new 0-shared polynomial $Q(x, y)$. The parties in the new committee also need to perform verification to make sure that the shares from the old committee are valid. Lastly, the parties end the handoff phase and calculate the full shares to switch back to $t$-threshold.

## 4.4 Notation

Before we start describing Shanrang, we list our denotations in Table 1. We use $C^{(e-1)}$ and $C^{(e)}$ to represent the previous committee and the new committee. For simplicity, we denote the set of the parties in $C^{(e-1)}$ and $C^{(e)}$ as $c_i$ and $c_i'$.

We denote the public keys and private keys for the $i$-th party in the new committee as $PK_i$ and $SK_i$. We will also use a symmetric encryption scheme between two parties in the old committee and the new committee. We denote the key as $K_{i,j}$, the encryption function as $E_K(x)$ and the decryption function as $D_K(x)$. $C_{alive}^{(e-1)}$ denotes the alive parties in the old committee before the handoff. We use $B$ and $B'$ to indicate the old and new bivariate polynomial of degree $(t, 2t)$.

We denote the polynomial commitment of $B(x, j)$ as $C_{B(x,j)}$, and the corresponding witness as $W_{B(x,j)}$. We denote the 0-share proactivization polynomial as $Q$.

We inherit the optimization of CHURP of the 0-sharing and use the same denotation of the subset of nodes participating in the handoff as $U$. However, we have made necessary changes to tolerate the adversary in the asynchronous network and use AMVCS to determine which subsets of $U$ to interpolate the new polynomials. We denote these subsets as $U_1$ and $U_2$. Besides the above, we also assume a collision-resistant hash function $H(x)$. We use $\lambda_j$ to represent the Lagrange coefficients.

## 4.5 Algorithm

Figure 1 shows the outline of Shanrang's handoff protocol. We assume $C_{B(x,j)}$, the polynomial commitments of the previous shares, are public. At the beginning

**Table 1.** notation

| Notation | Description |
|---|---|
| $C^{(e-1)}$, $C^{(e)}$ | Old/New committee |
| $c_i$, $c'_i$ | Members of the old/new committee |
| $PK_i$, $SK_i$ | Public/Private keys of the new committee |
| $K_{i,j}$ | symmetric key between two members generated by D-H key exchange |
| $E_K(x)$, $D_K(x)$ | Encrypt/Decrypt function with key $K$ |
| $C_{alive}^{(e-1)}$ | Subset of the old committee that is alive before handoff |
| $B(i,j), B'(i,j)$ | Old/new $(t, 2t)$-degree polynomial for sharing the secret |
| $C_{B(x,j)}$ | KZG commitment to $B(x,j)$ |
| $W_{B(i,j)}$ | Witness to evaluation of $B(x,j)$ at $x = i$ |
| $Q(i,j)$ | Bivariate proactivization polynomial |
| $U$ | Nodes participating in the handoff |
| $U_1, U_2$ | Subset of $U$ guaranteed up to date after ACS |
| $H(x)$ | Hash function (link elements of set if necessary) |
| $\lambda_j$ | Lagrange coefficients |

of the protocol, it sorts the parties in the new committee based on lexicographic order of the public keys, to deterministically select a subset of old committee $U$ of size $4t + 1$.

The first step is to generate the dimension-reduced shares. Every party $c_i$ in the old committee sends a point with witness $\{B(i,j), W_{B(i,j)}\}$ to every party $c'_j \in U$.

The parties $c'_j$ in $U$ together invoke the OptShareReduce protocol and calculate the reduced share $B(x, j)$.

Then, the parties in $U$ collaboratively proactivize the shares by setting up an instance of AMVCS and invoke OptShareUpdate. After OptShareUpdate finishes, the parties get a degree-$2t$ local 0-share polynomial $P(j)$.

Next, all the parties in the new committee $C^{(e)}$ launch another instance of AMVCS. Note that, in this AMVCS, $C^{(e)}$ is the participating party set and $U_1$ is the input party set. They invoke OptShareGen to verify the sampled 0-share polynomials and generate the local shares for the new committee. Particularly, for parties outside $U$, they participate in the AMVCS right away.

Finally, the parties in the new committee end the handoff phase and calculate the full share by switching back to $t$-threshold.

In the next subsections, we go through the sub-protocols OptShareReduce, OptShareUpdate, OptShareGen, and EndHandoff in details.

## 4.6 OptShareReduce

Figure 2 shows the pseudocode of OptShareReduce. It verifies the points from the old committee and switches the dimension to $2t$.

For any party $c'_j \in U$, upon receiving at least $2t + 1$ points with witnesses from the old committee, it verifies the commitments and interpolates the points

**Algorithm 1:** Shanrang Protocol Outline

```
Public Input: $\left\{C_{B(x,j)} \middle| c_j \in C^{(e-1)}\right\}$
Input: Set of nodes $C^{(e-1)}$, $C^{(e)}$
Assertion: $\left|C^{(e-1)}_{alive}\right| = n \geq 3t+1$, $\left|C^{(e)}\right| = n' \geq 4t+1$
1  Order $C^{(e)}$ based on lexicographic order of $PK_i$
2  Let $U = \left\{c'_j \middle| j \in [4t+1]\right\}$ denote the first $4t+1$ nodes
3  Let $C^{(e-1)}_{alive} = \{c_i | i \in [n]\}$ denote all alive nodes in the old committee

   // Step 1: Generate reduced share
4  Each node $c_i \in C^{(e-1)}_{alive}$ do
5      for $c'_j \in U$ do
6          Send off-chain to node $c'_j$ a point with witness $\left\{B(i,j), W_{B(i,j)}\right\}$
7      end
8  Each node $c'_j \in U$ do
9      Invoke OptShareReduce and calculate $B(x,j)$

   // Step 2: Proactivate
10 Each node $c'_j \in U$ do
11     Set up Asynchronous common set protocol $\mathsf{AMVCS}_1$
12     Invoke OptShareUpdate and calculate $P(j)$

   // Step 3: Verification and Dimension Switch
13 Each node $c'_j \in C^{(e)}$ do
14     Set up Asynchronous common set protocol $\mathsf{AMVCS}_2$
15     Invoke OptShareGen and retrieve verification status.

   // Step 4: End Handoff and Interpolate Full Shares
16 Each node $c'_j \in C^{(e)}$ do
17     Set up Asynchronous binary agreement $\mathsf{BA}_{\mathsf{end}}$
18     Invoke EndHandoff and calculate $B'(j,y)$
```

to get $B(x,j)$, a share in the $(2t)$-degree dimension. The subprotocol returns the share $B(x,j)$ back to the outline.

## 4.7 OptShareUpdate

Figure 3 presents the subprotocol OptShareUpdate. This protocol asks all the parties $c'_j \in U$ to jointly sample a new degree-$2t$ polynomial $P$ s.t. $P(0) = 0$.

Firstly, the party locally generates a random degree-$2t$ polynomial $P'_j$ s.t. $P'_j(0) = 0$. The party calculates $4t+1$ different evaluation points $P'_j(k), k \in [4t+1]$. Then, it encrypts $P'_j(k)$ with the symmetric key of $K_{j,k}$ and inputs the ciphertext $E_{K_{j,k}}(P'_j(k))$ to the *k-th party channel* of AMVCS instance $\mathsf{AMVCS}_1$, for every $k \in [4t+1]$. We use AMVCS here to make sure that the parties agree on a single 0-shared polynomial.

**Algorithm 2:** OptShareReduce (*For node* $c'_j$)

```
1 Upon receiving at least $2t+1$ points with witnesses
  $\{(B(i,j), W_{B(i,j)}) | i \in [2t+1]\}$
                                   // Assuming from the first 2t+1 nodes
2   for $i \in [2t+1]$ do
3       Invoke VerifyEval $(C_{B(x,j)}, i, B(i,j), W_{B(i,j)})$
4   end
5   Interpolate correct points to construct $B(x,j)$
```

**Algorithm 3:** OptShareUpdate (*For node* $c'_j$)

```
1 Generate a random polynomial $P'_j$ s.t. deg $(P'_j) = 2t, P'_j(0) = 0$
2 Encrypt points and input $\{E_{K_{j,k}}(P'_j(k)) | k \in [4t+1]\}$ to AMVCS$_1$
3 Upon appearing in the output of AMVCS$_1$
    // All nodes agree on $3t+1$ out of $4t+1$ nodes whose valuations
    of polynomial will contribute to the global one
                              // Denote the aggreed subset as $U_1$
                                // Now we construct $P = \sum_{c'_i \in U_1} P'_i$
4   Decrypt points and retrieve $\{P'_i(j) | c'_i \in U_1\}$
5   Calculate $P(j) = \sum_{c'_i \in U_1} P'_i(j)$
```

Whenever the party sees an output from $\mathsf{AMVCS}_1$, it decrypts the points and retrieves $\{P'_i(j)|c'_i \in U_1\}$. Then, it calculates $P(j) = \sum_{c'_i \in U_1} P'_i(j)$ and returns the share.

## 4.8 OptShareGen

Figure 4 shows the pseudocode of subprotocol OptShareGen.

At the beginning, each party in $U$ generates a random degree-$t$ polynomial $R_j$ s.t. $R_j(0) = P(j)$. Conceptually, now we have a 0-shared degree-$(t, 2t)$ bivariate polynomial $Q(x,y) = R_y(x)$, and $Q(0,0) = R_j(0)|_{j=0} = P(j)|_{j=0} = P(0) = 0$.

The party calculates $B'(x,j) = B(x,j) + R_j(x)$, effectively the refreshed new share. It also calculates $Z_j(x) = R_j(x) - P(j)$ and the commitment $Comm_j = \{g^{P(j)}, C_{Z_j}, W_{Z_j(0)}, C_{B'(x,j)}\}$.

Then, it enters the encrypted point $E_{K_{i,j}}(B'(i,j), W_{B'(i,j)})$ and the commitment $Comm_j$ as input to the *i-th channel* of another instance $\mathsf{AMVCS}_2$, for every $i \in U_1$.

Upon $\mathsf{AMVCS}_2$ outputs, the party retrieves the commitments $\{Comm_i|c'_i \in U_2\}$ and the ciphertexts of $\{\{B'(j,k), W_{B'(j,k)}\}|c'_k \in U_2\}$. It then verifies that $P$ is really a zero-share polynomial

$$\prod_{c'_k \in U_2} \left(g^{P(k)}\right)^{\lambda_k} = 1.$$

Finally, it checks the commitments for the shares sent from parties in $U_2$.

**Algorithm 4:** OptShareGen (*For node* $c'_j$)

```
1  Generate a random polynomial $R_j$ s.t. $\deg(R_j) = t, R_j(0) = P(j)$
2  Calculate $B'(x,j) = B(x,j) + R_j(x)$ and $Z_j(x) = R_j(x) - P(j)$
3  Generate commitments $Comm_j = \left\{g^{P(j)}, C_{Z_j}, W_{Z_j(0)}, C_{B'(x,j)}\right\}$
4  Encrypt points and input $\left\{Comm_j, E_{K_{i,j}}\left(B'(i,j), W_{B'(i,j),P(j),}\right) \middle| c'_i \in U_1\right\}$ to
   $\mathsf{AMVCS}_2$
5  Upon receiving output of $\mathsf{AMVCS}_2$
       // All nodes agree on $2t+1$ out of $3t+1$ nodes whose evaluations
       of the polynomial will contribute to the global one
                                       // Denote the agreed subset as $U_2$
6      Retrieve $\{Comm_i | c'_i \in U_2\}$
7      Decrypt points and retrieve $\left\{\left\{B'(j,k), W_{B'(j,k)}\right\} \middle| c'_k \in U_2\right\}$
8      Verify the commitments are valid and the zero-share property
       $\prod_{c'_k \in U_2} \left(g^{P(k)}\right)^{\lambda_k} = 1$
9      for $c'_k \in U_2$ do
10         Verify $g^{P(k)} = Comm_k[0]$
11         Invoke VerifyEval $\left(C_{Z_k}, 0, W_{Z_k(0)}\right)$
12         Verify polynomial $C_{B'(x,k)} = C_{B(x,k)} \times C_{Z_k} \times g^{P(k)}$
13         Invoke VerifyEval $\left(C_{B'(x,k)}, j, B'(j,k), W_{B'(j,k)}\right)$
14     end
```

## 4.9 EndHandOff

Figure 5 presents the last piece of the Shanrang protocol. In EndHandOff subprotocol, if all the verfication passed, the party broadcasts the verfication details together with the sender's signatures. It locally interpolates $\left\{\left\{B'(j,k), W_{B'(j,k)}\right\} \middle| c'_k \in U_2\right\}$ to construct $B'(j,y)$. Finally, it broadcasts the KZG commitments of the new shares $\left\{C_{B'(x,j)} \middle| c'_j \in U_2\right\}$ to all the parties with a Byzantine tolerating protocol. Note that, since at least $2t+1$ parties participate the broadcasting, any party will receive at least $t+1$ sets of commitments. Among them, at least one set is valid.

## 4.10 Analysis

Assuming $n$ parties in the old commitee and $n'$ parties in the new committee. Shanrang generates $O(nt)$ messages in generating the reduced shares. In OptShareUpdate, each input to the AMVCS is $O(t)$, so the communication complexity is $O(\lambda t^3 \log t)$. In OptShareGen, each input to the AMVCS is $O(n')$, so the communication complexity is $O(\lambda t n'^2 \log n')$ since only $O(t)$ among $n'$ nodes are actually broadcasting. As the AMVCS instances run the binary agreements in parallel, the total time complexity is in expectation $O(\log n')$.

**Algorithm 5:** EndHandoff (*For node $c'_j$*)

```
1  Upon all verifications passed
2      Broadcast successful verification
3      Interpolate points $\{\{B'(j,k), W_{B'(j,k)}\} | c'_k \in U_2\}$ to construct $B'(j,y)$
4      Byzantine broadcast KZG commitments of new reduced shares
         $\{C_{B'(x,j)} | c'_j \in U_2\}$ to all the parties.
5  Upon some commitment verification failed
6      Byzantine broadcast failed verification and its detail to blame the party.
7  Upon at most $t$ nodes did not output success
8      End handoff and move to the normal phase of the epoch
9  Upon at least $t+1$ nodes output fail
10     Fall to pessimistic path
```

## 5 Proofs

**Theorem 1 (Liveness).** *The adversary $\mathcal{A}$ cannot prevent the protocol from making progress, i.e. all honests nodes can eventually end handoff phase.*

*Proof.* We follow the three stages of the main algorithm and prove that an adequate amount of nodes are up to date for the three algorithms to proceed. To do that we prove the following three lemmas:

**Lemma 1.** *Given that $\left|C^{(e-1)}_{alive}\right| \geq 3t+1$, all $4t+1$ nodes in $U$, including at least $3t+1$ honest ones, can eventually construct their original reduced share $B(x,j)$ in algorithm* OptShareReduce*.*

*Proof.* $\mathcal{A}$ can corrupt at most $t$ nodes in $C^{(e-1)}_{alive}$, so at least $3t+1-t=2t+1$ honest nodes will send points with witnesses to all nodes in $U$. Nodes in $U$ are guaranteed to receive at least $2t+1$ points with witnesses, among which at most $t$ points are sent from corrupted nodes. At least $t+1$ points will pass verification. They are all correct points and each node has enough points to interpolate a $t$-degree polynomial $B(x,j)$.

**Lemma 2.** *Given that $|U| = 4t+1$, All honest nodes in $U$ can eventually agree on polynomial $P$, summing the proposed polynomial of $3t+1$ nodes, including at least $2t+1$ honest ones, in algorithm* OptShareUpdate*.*

*Proof.* A total of $4t+1$ nodes participate in $\mathsf{AMVCS}_1$ with at most $t$ corrupted. According to the totality property of the AMVCS primitive, all honest nodes (at least $3t+1$) can produce an output. Also, according to the validity property, it contains the inputs of at least $3t+1$ nodes, $2t+1$ of which are honest. This guarantees the successful agreement on $P$.

**Lemma 3.** *Given that $\left|C^{(e)}\right| \geq 4t+1$, All honest nodes in $C^{(e)}$ can eventually agree on polynomial $Q$, interpolating the proposed polynomial of $2t+1$ nodes, including at least $t+1$ honest ones, in algorithm* OptShareGen*.*

*Proof.* A total of at least $3t + 1$ nodes participate in $\mathsf{AMVCS}_2$ with at most $t$ corrupted. According to the totality property of the AMVCS primitive, all honest nodes can produce an output. Also, according to the validity property, it contains the input of at least $2t+1$ nodes, $t+1$ of which are honest nodes. This guarantees the successfull interpolation on $Q$.

Thus, all nodes can eventually receive enough points and commitments from $\mathsf{AMVCS}_2$. If all the inputs are from honest nodes, for each honest node, all verification will pass and it will output success and interpolate the correct full share. On the other hand if there are corrupted nodes among them, honest nodes will fail the verification and output fail.

Based on how the corrupted nodes behave, there are several possibilities.

- If any corrupted node violate zero-share and generate a wrong new share, such behavior can be detected during the verification involving only the commitment package, i.e. line 10-14 of algorithm $\mathsf{OptShareGen}$. This means all honest nodes will fail some of these verifications.
- On the other hand, if all the corrupted nodes follow zero-share, but anyone of them sends a faulty value of the correct new share, such behavior can be detected during the KZG evaluation verification, i.e. line 15 of algorithm $\mathsf{OptShareGen}$. This means there can be only part of the honest nodes that fail these verifications.
- Also, we take into consideration that corrupted nodes may refuse to submit their verifications in an attempt to stall the handoff.

We require that the handoff phase ends when at most $t$ nodes do not output success. At that time we can guarantee the success of zero-share and the correctness of the handoff, as well as enough points to recover the secret and enough nodes to proceed the protocol. If some honest nodes finish their verification and output success later on, they can still add in to the new committee. If they output fail, they can just refrain from joining.

On the other hand, if at least $t+1$ nodes output fail, the handoff fails and we fall into pessimistic path (same as CHURP) to single out corrupted nodes. This way the protocol is destined to proceed and liveness is guaranteed.

**Theorem 2 (Integrity).** *When handoff phase eventually ends, either all honest nodes hold their correct full shares and all three invariants hold, or at least $t+1$ honest nodes output fail.*

Note: The three invariants are:

- **Inv-Secret**: The secret $s = B(0,0)$ remains the same.
- **Inv-Comm**: KZG commitments to a complete set of reduced shares, e.g. $\{B(x,k) | k \in [2t+1]\}$, are available to all nodes.
- **Inv-State**: At least $3t+1$ nodes of the new committee hold a full share $B(j,y)$ and a proof to its correctness, which contains a set of $2t+1$ points and their respective witnesses.

*Proof.* We notice that a complete set of reduced shares only requires that there are $2t+1$ of them. It does not need exactly be $k \in [2t+1]$, because $C_{B(x,j)} = g^{B(x,j)}$ and we can apply lagrange interpolation to generate the KZG commitment of any specific share, as long as there are $2t+1$ available ones.

Specifically, assuming $k \in P$, we can derive the following formula from lagrange interpolation $B(x,l) = \sum_{k \in P} \lambda'_k B(x,k)$:

$$C_{B(x,l)} = g^{B(\alpha,l)} = \prod_{k \in P} g^{\lambda'_k B(\alpha,k)} = \prod_{k \in P} C_{B(x,k)}^{\lambda'_k} \tag{1}$$

This means a set of $2t+1$ witnesses are sufficient to generate the witness of any point of a reduced share.

Note that there may be fractions on the exponential, which is equivalent to extracting the root of a group element. Although we cannot derive the actual value of the root due to *t-CDH* assumption, there is a way we can detour this operation by tweaking the KZG commitment scheme. The modified scheme and its proof is provided in Appendix A.

According to lemma 1, all honest nodes in $U$ can correctly construct their reduced share. Then, in algorithm OptShareGen, the new committee will verify the points they receive to prove the correctness of $Q$, ensuring the correctness of the new full share. Specifically, we can sequentially prove the correctness of each step of generating new shares:

- The commitments received are valid and consistent, as in line 8 of algorithm OptShareGen, .
- $P$ is a zero-hole polynomial, i.e. $P(0) = 0$, as in line 9,
- $Z_k$ is a zero-hole polynomial, i.e. $Z_k(0) = R_k(0) - P(j) = 0$, as in line 12. (This means $B'(0,j) = B(0,j) + P(j)$, and by interpolation it means $B'(0,0) = B(0,0) + P(0) = B(0,0)$. Thus it proves invariant **Inv-Secret**)
- $B'(x,k)$ is correct, i.e. $B'(x,k) = B(x,k) + Z_k(x) + P(k)$, as in line 12.
- $C_{B'(x,k)}$ is the correct KZG commitment of $B'(x,k)$, as in line 13. (This proves invariant **Inv-Comm**)
- $B'(j,k)$ is a correct valuation of $B'(x,k)$ at $j$, as in line 13. (This proves invariant **Inv-State**)

Based on these validations we know that, after interpolation, the new full share $B'(j,y)$ guarantees the three invariants, which proves the integrity of the algorithm.

**Theorem 3 (Secrecy).** *The adversary $\mathcal{A}$, corrupting no more than $t$ nodes in both the old and the new committee, can deduce nothing about the secret $s$, i.e. all possible values of it is equally likely.*

*Proof.* Based on the hiding property of the KZG commitment, KZG witnesses and commitments give no information about the secret $s$ because $\mathcal{A}$ cannot determine the respective polynomial with non-negligible probability.

Based on the security of the symmetric encryption scheme, corrupted nodes cannot learn the private inputs to $\mathsf{AMVCS}_2$.

Based on the security of the Diffie-Hellman assumption, encrypted messages are also secure to corrupted nodes if they do not take part in the exchange.

Other information retrieved by $\mathcal{A}$ also provide no information about the secret $s$ according to the following three lemmas.

**Lemma 4.** *Corrupting no more than $t$ nodes in both the old and the new committee, $\mathcal{A}$ sees all values of secret $s$ as equally likely in algorithm* $\mathsf{OptShareReduce}$.

*Proof.* $\mathcal{A}$ can learn at most $2t$ reduced share $B(x, j)$, in worst case $t$ of which from corrupting nodes in $U$ and the other $t$ from corrupting nodes in $C_{alive}^{(e-1)}$ that stay in the new committee. Apart from that $\mathcal{A}$ can learn at most $t$ full share $B(j, y)$ from corrupting nodes in $C_{alive}^{(e-1)}$. But since $B$ is a $(t, 2t)$-degree polynomial, the numbers of both full shares and reduced shares are no more than the respective degree of it. So these shares are not sufficient to interpolate the secret-sharing function $B$. Thus the value of any point not included in these shares, including the secret $s = B(0, 0)$, are viewed completely random.

**Lemma 5.** *$\mathcal{A}$ Corrupting no more than $t$ nodes in the new committee, the new committee can generate a random polynomial $Q$.*

*Proof.* According to lemma 2, the output of $\mathsf{AMVCS}_1$ contains the input of $2t+1$ honest nodes, guaranteeing the generation of $P$ to be random. According to lemma 3, the output of $\mathsf{AMVCS}_2$ contains the input of $t + 1$ honest nodes, guaranteeing the interpolation of $Q$ to be random.

**Lemma 6.** *Corrupting no more than $t$ nodes in the new committee, $\mathcal{A}$ sees all values of secret $s$ as equally likely in algorithm* $\mathsf{OptShareGen}$.

*Proof.* $\mathcal{A}$ can learn at most $t$ full share $B'(j, y)$ from corrupting nodes in $C_{(e)}$. It is no more than the respective degree of the polynomial $B'$, which is also $t$. So these shares are not sufficient to interpolate the secret-sharing function $B'$. Thus, the value of any point not included in these shares, including the secret $s = B'(0, 0)$, are completely random.

In this way secrecy of the protocol is proved.

## 6 Evaluation

We implemented Shanrang with Python and tested on geographically distributed machines on Amazon EC2. Specifically, we simulate an old committee of $3t + 1$ parties performing handoffs with a new committee of $4t + 1$ parties. We use Honeybadger BFT [11] as the AMVCS protocol. For the symmetric encryption system, we use AES-256 in the CBC mode. We instantiate the common coin primitive with Boldyreva's signature scheme [3]. We use the Charm wrapper of PBC library for the bilinear group used in KZG scheme. We used the symmetric bilinear group SS512, which provides security of 159 bits. Each commitment takes 64 bytes. For each data point, we ran 5 times.

We evaluated Shanrang on two different setups:

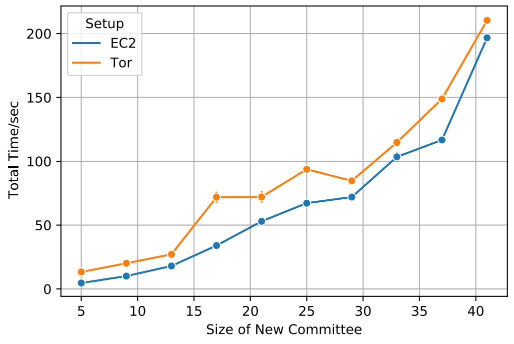

**Fig. 2.** Total Time vs. Size of New Committee

- Running all the parties on Amazon EC2 machines located in 12 regions across different continents. Specifically, we use `t2.micro` machines from North Virginia, Ohio, North California, Oregon, Ireland, London, Frankfurt, Sao Paulo, Singapore, Sydney, Tokyo and Central Canada. This setup shows how Shanrang performs on the global network. Geographically separated nodes provides strong robustness against failure or corruption. This highly increases the difficulty of adversaries to corrupt them in a short amount of time.
- Running all the parties on The Onion Router (Tor) relay network. Tor is a secure encrypted protocol that transmits communications with onion routers, where the encrypted messages are relayed through a number of relay nodes. Tor network is known unstable and it has very high latency [7]. Despite the wide range of latency of Tor [5], Shanrang provides the liveness guarantee without any trouble.

## 6.1 Optimal Handoffs

Figure 2 shows the plot of the total time versus the size of the new committee. The x-axis is the size of the new committee, ranging from 5 to 41 for EC2 and from 5 to 21 for Tor experiments. The y-axis is the total time measured in seconds. We can see that for both settings, total time to complete one handoff increases as the number of nodes increases. Even with the high latency on the Tor network, performance remains on the same level with that on EC2 machine network. With our python implementation, we achieve a handoff time of 200 seconds for a network of up to 41 nodes.

Figure 3 and Figure 4 present how the message size and the number of messages change with the size of new committee. As our protocol is totally network agnostic, the message sizes and counts are the same across the EC2 and Tor setups. More specifically, when the size of the new committee reaches 42,

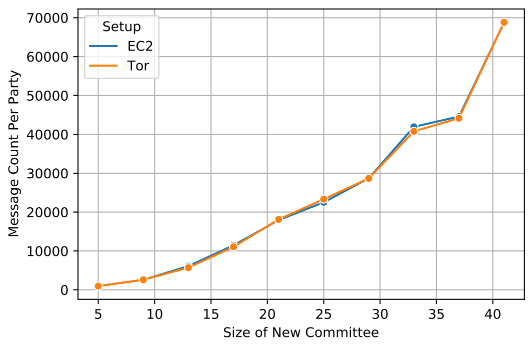

**Fig. 3.** Message Size (MB) per Node vs. Size of New Committee

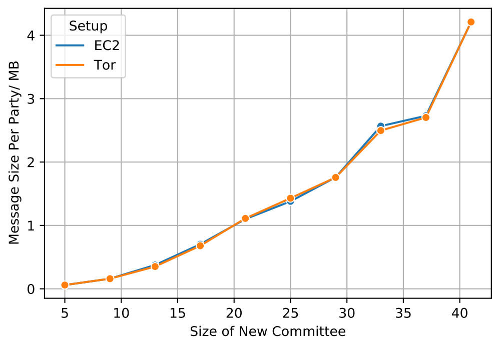

**Fig. 4.** Message Count per Node vs. Size of New Committee

the total network traffic reaches around 4MB, causing a network bandwidth of 20 KB/sec. This bandwidth does not pose any trouble even for the Tor network.

## 6.2 Handoff with Corrupted Nodes

To evaluate how Shanrang works against adversaries, we added silent adversaries to the committee to test the performance of the protocol under corruption. For silent adversaries, they do not respond to any messages. Specifically, we ran 41 nodes and vary the number of nodes that become silent adversaries. Figure 5 shows the total time of handoffs versus the number of corrupted nodes. We can see that the performance does not fluctuate much. This is mainly because, among EC2 machines across continents, message delivery does not have a consistent speed so even when all nodes behave honestly, some messages are to be received later and nodes that send them are viewed behind in progress. Namely, slow nodes are treated as adversaries automatically and the tolerance quota is always

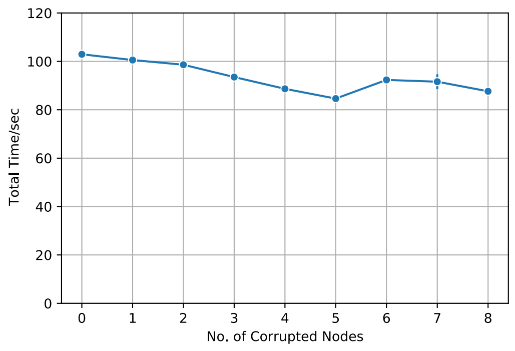

**Fig. 5.** Total Time vs. Number of Corrupted Nodes

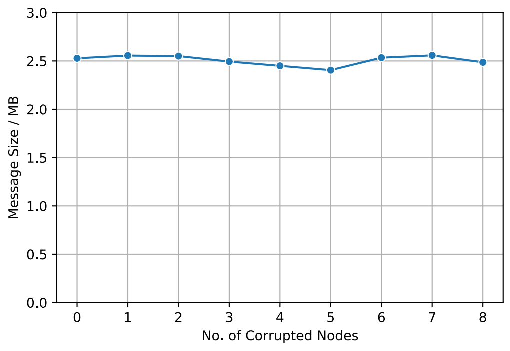

**Fig. 6.** Message Size (MB) per Honest Node vs. Number of Corrupted Nodes

utilized to speed up the protocol by having the fast nodes not waiting for the slow nodes. However, different from a true adversary that behaves maliciously, an honest slow node is able to catch up. Considering that the protocol can proceed without these messages, corrupted nodes that do not send messages at all will not significantly slow down the progress. Similarly, Figure 6 and Figure 7 show that the message size per honest node and message count per honest node does not change significantly with the number of adversaries.

## 7 Conclusion

In this work, we present Shanrang, the first fully asynchronous proactive secret sharing scheme with dynamic committee support. It tolerates up to $n/4$ Byzantine faults.

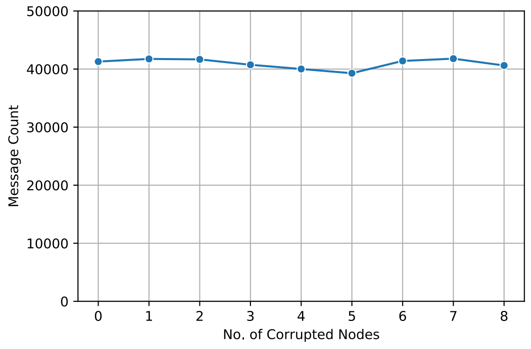

**Fig. 7.** Message Count per Honest Node vs. Number of Corrupted Nodes

## References

1. Lost passwords lock millionaires out of their bitcoin fortunes, https://www.nytimes.com/2021/01/12/technology/bitcoin-passwords-wallets-fortunes.html
2. Ben-Or, M., Kelmer, B., Rabin, T.: Asynchronous secure computations with optimal resilience. In: Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing. pp. 183–192 (1994)
3. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In: International Workshop on Public Key Cryptography. pp. 31–46. Springer (2003)
4. Cachin, C., Kursawe, K., Lysyanskaya, A., Strobl, R.: Asynchronous verifiable secret sharing and proactive cryptosystems. In: Proceedings of the 9th ACM Conference on Computer and Communications Security. pp. 88–97 (2002)
5. Cangialosi, F., Levin, D., Spring, N.: Ting: Measuring and exploiting latencies between all tor nodes. In: Proceedings of the 2015 Internet Measurement Conference. pp. 289–302 (2015)
6. Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. In: OSDI. vol. 99, pp. 173–186 (1999)
7. Dingledine, R., Murdoch, S.J.: Performance improvements on tor or, why tor is slow and what were going to do about it. Online: http://www. torproject. org/press/presskit/2009-03-11-performance. pdf p. 68 (2009)
8. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing or: How to cope with perpetual leakage. In: annual international cryptology conference. pp. 339–352. Springer (1995)
9. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: International conference on the theory and application of cryptology and information security. pp. 177–194. Springer (2010)
10. Maram, S.K.D., Zhang, F., Wang, L., Low, A., Zhang, Y., Juels, A., Song, D.: Churp: Dynamic-committee proactive secret sharing. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 2369–2386 (2019)

11. Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of bft protocols. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 31–42 (2016)
12. Mostéfaoui, A., Moumen, H., Raynal, M.: Signature-free asynchronous binary byzantine consensus with t¡ n/3, o (n2) messages, and o (1) expected time. Journal of the ACM (JACM) **62**(4), 1–21 (2015)
13. Pramanik, S., Upadhaya, S.: Vpss: a verifiable proactive secret sharing scheme in distributed systems. In: IEEE Military Communications Conference, 2003. MILCOM 2003. vol. 2, pp. 826–831. IEEE (2003)
14. Schultz, D.A., Liskov, B., Liskov, M.: Mobile proactive secret sharing. In: Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing. pp. 458–458 (2008)
15. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
16. Yurek, T., Luo, L., Fairoze, J., Kate, A., Miller, A.K.: hbacss: How to robustly share many secrets. IACR Cryptol. ePrint Arch. **2021**, 159 (2021)
17. Zhou, L., Schneider, F.B., Van Renesse, R.: Apss: Proactive secret sharing in asynchronous systems. ACM transactions on information and system security (TISSEC) **8**(3), 259–286 (2005)

## A Interpolation with KZG commitments

In this section, we discuss how we modify $\mathsf{PolyCommit}_{\mathsf{DL}}$ [9] to satisfy our need of interpolating commitments and witnesses. In short, we allow the commitments and witnesses to be represented in a "power + exponential" form, while still maintaining their uniqueness.

Specifically, in our scheme, we use $\langle \mathcal{C}, p\rangle = \mathcal{C}^{\frac{1}{p}}$ to represent a commitment. The same applies to witnesses. As long as $p \neq 1$, we cannot simplify it using the public key anymore because of the $t\text{-}CDH$ assumption. That's why we have to keep this form after the interpolation.

We follow the original setup and the implementations of $\mathsf{Commit}$ and $\mathsf{CreateWitness}$, except that for these two functions, we output a tuple, namely, $\langle C, 1\rangle$ instead of $C$ and $\langle W, 1\rangle$ instead of $W$.

As for $\mathsf{VerifyEval}$ function, we adjust it so that it accepts our weakened representation of commitments and witnesses:

$\mathsf{VerifyEval}\left(PK, \langle \mathcal{C}, p_c\rangle, i, \phi(i), \langle w_i, p_w\rangle\right)$ verifies that $\phi(i)$ is the evaluation at the index $i$ of the polynomial committed to by $\langle \mathcal{C}, p\rangle$. After calculating $d = (p_c, p_w) = gcd\left(p_c, p_w\right)$ and $l = [p_c, p_w] = lcm\left(p_c, p_w\right)$, the algorithm outputs 1 if $e\left(\mathcal{C}, g\right)^{\frac{p_w}{d}} = e\left(w_i, g^{\alpha - i}\right)^{\frac{p_c}{d}} \cdot e\left(g, g\right)^{l\phi(i)}$, and 0 otherwise.

Note that, if both the commitment and the witness are directly generated by the original $\mathsf{Commit}$ and $\mathsf{CreateWitness}$ implementations, where, both are not generated by interpolation and $p_c = p_w = 1$, then it is exactly the same as the original $\mathsf{VerifyEval}$ function.

The algorithm is still correct because if we let $\mathcal{C}' = \mathcal{C}^{\frac{1}{p_c}}$ and $w_i' = w_i^{\frac{1}{p_w}}$, then based on the original scheme we have:

$$\begin{aligned} e\left(\mathcal{C},g\right)^{\frac{p_w}{d}} &= e\left(\mathcal{C}',g\right)^{\frac{p_c p_w}{d}} = e\left(\mathcal{C}',g\right)^{l} \\ &= \left(e\left(w_i, g^{\alpha-i}\right) e\left(g,g\right)^{\phi(i)}\right)^{l} = e\left(w_i, g^{\alpha-i}\right)^{l} e\left(g,g\right)^{l\phi(i)} \\ &= e\left(w_i, g^{\alpha-i}\right)^{\frac{p_c}{d}} e\left(g,g\right)^{l\phi(i)} \end{aligned} \tag{2}$$

Also, such modification satisfies the property of evaluation binding. Suppose there exists an adversary $\mathcal{A}$ that is able to break the property of commitment $(\mathcal{C}, p_c)$ and compute two witness tuples $\langle i, \phi(i), \langle w_i, p_w\rangle\rangle$ and $\langle i, \phi(i)', \langle w_i', p_w'\rangle\rangle$, such that they are both accepted by VerifyEval, we can construct an adversary $\mathcal{B}$ that breaks the $t$-$SDH$ assumption using $\mathcal{A}$:

B presents the $t$-$SDH$ instance $\left\langle \mathcal{G}, g, g^{\alpha}, g^{\alpha^2}, ..., g^{\alpha^t}\right\rangle$ as public key to $\mathcal{A}$. $\mathcal{A}$ computes the commitment and two witness tuples as above. B calculates $(p_c, p_w, p_w') = d$, $p_c = dr_c$, $p_w = dr_w$, $p_w' = dr_w'$, and after offsetting it gets:

$$e\left(\mathcal{C},g\right)^{r_w} = e\left(w_i, g^{\alpha-i}\right)^{r_c} e\left(g,g\right)^{dr_c r_w \phi(i)} \tag{3}$$

$$e\left(\mathcal{C},g\right)^{r_w'} = e\left(w_i', g^{\alpha-i}\right)^{r_c} e\left(g,g\right)^{dr_c r_w' \phi(i)'} \tag{4}$$

For $\psi_i = \log_g w_i$ and $\psi_i' = \log_g w_i'$, we can get:

$$\begin{aligned} & eq.\ (3)^{r_w'} = eq.\ (4)^{r_w} \\ \Rightarrow & \psi_i(\alpha - i) r_c r_w' + dr_c r_w r_w' \phi(i) = \psi_i'(\alpha - i) r_c r_w + dr_c r_w r_w' \phi(i)' \\ \Rightarrow & \frac{\psi_i r_w' - \psi_i' r_w}{\phi(i)' - \phi(i)} = \frac{dr_w r_w'}{\alpha - i} \end{aligned} \tag{5}$$

Therefore, B can compute:

$$\left(\frac{w_i^{r_w'}}{{w_i'}^{r_w}}\right)^{\frac{1}{dr_w r_w'\left(\phi(i)'-\phi(i)\right)}} = g^{\frac{\psi_i r_w' - \psi_i' r_w}{dr_w r_w'\left(\phi(i)'-\phi(i)\right)}} = g^{\frac{1}{\alpha-i}}, \tag{6}$$

and output $\left\langle -i, g^{\frac{1}{\alpha-i}}\right\rangle$, thus breaking the $t$-$SDH$ assumption.

The modification also satisfies the hiding property, because we do not change how commitments and witnesses are generated. Any witness generated through the interpolation requires at least $t+1$ points, which is not allowed in the hiding property. Lastly, we note that the denominators should be polynomial-sized to guarantee the soundness.