CHURP: Dynamic-Committee Proactive Secret Sharing

Sai Krishna Deepak Maram[†*], Fan Zhang[†*], Lun Wang[‡*], Andrew Low[‡*], Yupeng Zhang[‡*], Ari Juels[†*] and Dawn Song[‡*]
[†]Cornell Tech, [‡]UC Berkeley
[*]Initiative for CryptoCurrencies & Contracts

***Abstract*—We introduce CHURP (CHUrn-Robust Proactive secret sharing). CHURP enables secure secret-sharing in *dynamic* settings, where the committee of nodes storing a secret changes over time. Designed for blockchains, CHURP has lower communication complexity than previous schemes: $O(n)$ on-chain and $O(n^2)$ off-chain in the optimistic case of no node failures.**

CHURP includes several technical innovations: An efficient new proactivization scheme of independent interest, a technique (using asymmetric bivariate polynomials) for efficiently changing secret-sharing thresholds, and a hedge against setup failures in an efficient polynomial commitment scheme. We also introduce a general new technique for inexpensive off-chain communication across the peer-to-peer networks of permissionless blockchains.

We formally prove the security of CHURP, report on an implementation, and present performance measurements.

I. Introduction

Secure storage of private keys is a pervasive challenge in cryptographic systems. It is especially acute for blockchains and other decentralized systems. In these systems, private keys control the most important resources—money, identities [1], etc. Their loss has serious and often irreversible consequences.

An estimated four million Bitcoin (today worth $14+ Billion) have vanished forever due to lost keys [2]. Many users thus store their cryptocurrency with exchanges such as Coinbase, which holds at least 10% of all circulating Bitcoin [3]. Such centralized key storage is also undesirable: It erodes the very decentralization that defines blockchain systems.

An attractive alternative is *secret sharing*. In (t, n)-secret sharing, a *committee* of n nodes holds *shares* of a secret s—usually encoded as $P(0)$ of a polynomial $P(x)$ [4]. An adversary must compromise at least $t + 1$ players to steal s, and at least $n-t$ shares must be lost to render s unrecoverable.

Proactive secret sharing (PSS), introduced in the seminal work of Herzberg et al. [5], provides even stronger security. PSS periodically *proactivizes* the shares held by players, while keeping s constant. Players obtain new shares of the secret s that are independent of their old shares, which are then discarded. Provided an adversary never obtains more than t shares between proactivizations, PSS protects the secret s against ongoing compromise of players.

Secret sharing—particularly PSS—would seem to enable users to delegate private keys safely to committees and avoid reliance on a single entity or centralized system. Indeed, a number of commercial and research blockchain systems, e.g., [6], [7], [8], [9], [10], rely on secret sharing to protect users' keys and other sensitive data.

These systems, though, largely overlook a secret-sharing problem that is critical in blockchain systems: *node churn*.

In *permissionless* (open) blockchains, such as Bitcoin or Ethereum, nodes may freely join and leave the system at any time. In *permissioned* (closed) blockchains, only authorized nodes can join, but nodes can fail and membership change. Thus blockchain protocols for secret sharing must support committee membership changes, i.e., *dynamic* committees.

Today there are no adequate PSS schemes for dynamic committees. Existing protocols support static, but not dynamic committees [5], [11], assume weak, passive adversaries [12], [13], are efficient only for batched secrets [14], or incur prohibitive communication costs [15], [16], [17], [18], [19].

In this paper, we address this critical gap by introducing a new dynamic-committee proactive secret-sharing protocol called CHURP (*CHUrn-Robust Proactivization*).

A. CHURP *functionality*

CHURP allows a dynamic committee, i.e., one undergoing churn, to maintain a shared secret s securely.

Like a standard PSS scheme, CHURP proactivizes shares in every fixed interval of time known as an *epoch*. It supports dynamic committees as follows. An old committee of size n with a (t, n)-sharing of a secret s can transition during a *handoff* to a possibly disjoint new committee of size n with a new (t, n)-sharing of s. CHURP achieves security against an *adaptive, active* adversary that compromises $t < n/2$ nodes in *each* of the old and new committees. CHURP also allows changes to t and n between epochs. (Periodic changes to s are specifically not a goal of PSS schemes, but are easy to add.)

Our main achievement in CHURP is its *very low communication complexity*: optimistic per-epoch communication complexity in a blockchain setting of $O(n)$ on-chain—which is optimal—and $O(n^2)$ off-chain, i.e., over point-to-point channels. While the on-chain complexity is lower than off-chain, it comes with the additional cost of placing transactions on the blockchain. Cheating nodes cause pessimistic on-chain / off-chain communication complexity $O(n^2)$ / $O(n^3)$. *Both* off-chain costs are substantially lower than in other schemes.

Despite somewhat complicated mechanics, CHURP realizes a *very simple abstraction*: It simulates a trusted third party

that stores s for secure use in a wide range of applications—threshold cryptography, secure multi-party computation, etc.

B. Technical challenges and solutions

To achieve its low communication complexity, CHURP must overcome several major technical challenges. The first challenge is that previous PSS schemes, relying on techniques from Herzberg et al. [5], incur high communication complexity for proactivization ($O(n^3)$ off-chain per epoch). CHURP uses a *bivariate* polynomial $B(x, y)$ to share secret s, and introduces a new proactivization protocol with cost $O(n^2)$. This protocol is based on efficient *bivariate 0-sharing*, i.e., generation of a randomized, shared polynomial $B(x, y)$ with $B(0, 0) = 0$ to refresh shares. CHURP's 0-sharing technique is of independent interest: It can also lower the communication complexity of Herzberg et al. [5] and related schemes.

The second challenge is that during a handoff, an adversary may control t nodes in each of the old and new committees, and thus $2t$ *nodes in total*. Compromise of $2t$ shares in a (t, n)-sharing would leak the secret s. Previous schemes, e.g., [17], address this problem using "blinding" approaches with costly communication. CHURP introduces a novel, low communication-complexity technique called *dimension-switching*. It uses an *asymmetric* bivariate polynomial $B(x, y)$, with degree t in one dimension and degree $2t$ in the other. During a handoff, it switches temporarily to a $(2t, n)$-sharing of s to tolerate up to $2t$ compromised shares; afterward, it switches back to a (t, n)-sharing.

Finally, most PSS schemes commit to secret degree-t polynomials using classical schemes (e.g., [20], [21]) with per-commitment size $O(t)$. CHURP uses an alternative due to Kate, Zaverucha, and Goldberg (KZG) [22] with size $O(1)$. Use of KZG for secret sharing isn't new [23], but CHURP introduces a novel KZG *hedge*. KZG assumes trusted setup and a non-standard hardness assumption. If these fail, CHURP still remains secure—but degrades to slightly weaker adversarial threshold $t < n/3$.

We compose these techniques to realize CHURP with *provable security*. We give a simulation-based security proof.

C. Implementation and Experiments

We present an implementation of CHURP. Our experiments show very practical communication and computation costs—2300x improvement over the existing state-of-the-art dynamic-committee PSS scheme [17] in the off-chain communication complexity for a committee of size 100 (See Section VI).

Additionally, to achieve inexpensive off-chain communication among nodes in CHURP, we introduce a new technique for permissionless blockchains that is of independent interest. It leverages the peer-to-peer gossip network as a low-cost anonymous point-to-point channel. We experimentally demonstrate off-chain communication in Ethereum with monetary cost orders of magnitude less than on-chain communication.

D. Outline and Contributions

After introducing the functional, adversarial, and communication models in Section II, we present our main contributions:

- *CHUrn-Robust Proactive secret sharing (CHURP):* In Section III, we introduce CHURP, a practical dynamic-committee PSS scheme with lower communication complexity than previous schemes.
- *Novel secret-sharing techniques:* We introduce a *new 0-sharing* protocol for efficient proactivization in Section IV, a new *dimension-switching* technique to safeguard the secret in committee handoffs in Section V-C, and hedging techniques for failures in the KZG polynomial-commitment scheme in Appendix D-B.
- *New point-to-point blockchain communication technique:* We introduce a novel point-to-point communication technique for permissionless blockchains in Section VII—usable in CHURP and elsewhere—with orders of magnitude less cost than on-chain communication.
- *Implementation and experiments:* We report on an implementation of CHURP in Section VI and present performance measurements demonstrating its practicality.
- *Provable security:* We give a simulation-based security proof for CHURP in Appendix A.

We discuss related work in Section VIII and CHURP's many potential applications—threshold signatures and decryption, smart contracts with private keys, consensus simplification for light clients, etc.—in Appendix B. We will release CHURP system as an open-source tool.

II. Model and Assumptions

We now describe the functional, adversarial, and communication models used for CHURP.

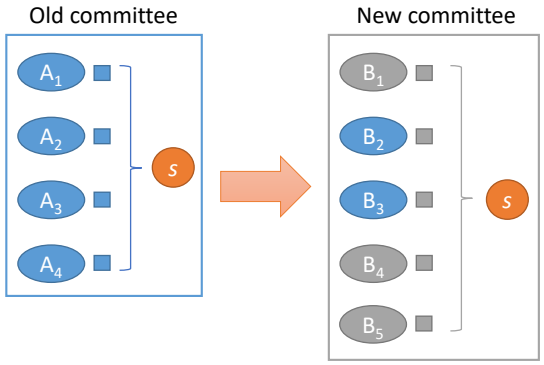


Fig. 1: Handoff between two committees a dynamic proactive secret-sharing epoch. The secret s remains fixed. Committees may intersect, e.g., in this example, $B_2 = A_2$ and $B_3 = A_3$.

In a secret-sharing scheme, a *committee* of nodes shares a fixed secret s. We let $\mathcal{C}$ denote a committee and denote the n nodes in the committee by $\{\mathcal{C}_i\}_{i=1}^{n}$. Each node $\mathcal{C}_i$ holds a distinct share s_i. CHURP *proactivizes* shares, i.e., changes them periodically to prevent leakage of s to an adversary that gradually compromises nodes. Again, we emphasize that CHURP does so for a *dynamic* committee [14], [17], i.e., nodes may periodically leave / join the committee.

Shares change in a proactive secret-sharing protocol such as CHURP during what is called a *handoff* protocol. Handoff

proactivizes s, i.e., changes its associated shares, while transferring s from an old committee to a new, possibly intersecting one. Fig. 1 depicts the handoff process. The adversarial model for proactive secret sharing in general limits adversarial control to a *threshold* t of nodes per committee. During a handoff, CHURP allows nodes to agree out of band on a change to t, as explained below.

A. Functional model

Epoch: Time in CHURP, as in any proactive secret-sharing scheme [5], is divided into fixed intervals of predetermined length called *epochs*. In each epoch, a specific committee of nodes assumes control of and then holds s. Concretely, in an epoch e, a committee $\mathcal{C}^{(e)}$ of size $N^{(e)}$ shares s using a $(t, N^{(e)})$-threshold scheme.

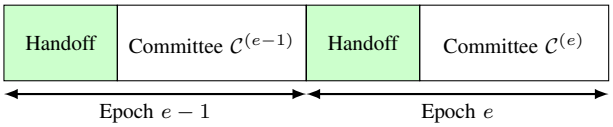


Fig. 2: Each epoch begins with a handoff phase during which the old committee hands off the secret s to the new committee. It is followed by a period of committee operation.

Handoff: As shown in Fig. 2, handoff takes place at the beginning of an epoch. It involves a transfer of s from an old committee, which we denote $\mathcal{C}^{(e-1)}$, to a new committee, denoted $\mathcal{C}^{(e)}$. Prior to completion of the handoff, $\mathcal{C}^{(e-1)}$ is able to perform operations using s.

Churn: In the dynamic-committee setting of CHURP, nodes can leave a committee at any time, but can only be added during a handoff. Let $\mathcal{C}^{(e-1)}_{left}$ denote the set of nodes that have left the committee before the handoff protocol in epoch e. Let $\mathcal{C}^{(e-1)}_{alive} = \mathcal{C}^{(e-1)} \setminus \mathcal{C}^{(e-1)}_{left}$ denote the set of nodes that participate in the handoff protocol. We let *churn rate* α denote a bound such that $|\mathcal{C}^{(e-1)}_{alive}| \geq |\mathcal{C}^{(e-1)}|(1-\alpha)$. Later, we provide a lower bound on the committee size using the churn rate α.

Keys: We assume that every node in CHURP has private / public key pair and that public keys are known to all nodes in the system. Such a setup is common in secret-sharing systems [5], [17].

B. Adversarial model

We consider an adversary $\mathcal{A}$ that is *active* and *adaptive*. It may decide to 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. (A node may thus be "released" by an adversary in a new epoch so that it is no longer corrupted.) Corrupted nodes are allowed to deviate from the protocol arbitrarily. The proofs of correctness used by nodes in CHURP requires that we assume a *computationally bounded* (polynomial-time) adversary.

As noted above, we limit the adversary $\mathcal{A}$ to corruption of no more than a threshold of nodes in a given committee. This threshold, as noted above, may change in CHURP through out-of-band agreement by committees. In this case, letting t and t' denote corruption thresholds for old and new committees respectively, $\mathcal{A}$ may control at most t nodes in $\mathcal{C}^{(e-1)}$ and t' nodes in $\mathcal{C}^{(e)}$. We present the protocol in CHURP for threshold changes in Appendix E. For simplicity of exposition, however, we assume in what follows that $t = t'$, i.e., the corruption threshold t remains fixed.

Observe that during the handoff between epochs $e-1$ and e, members of both committees, $\mathcal{C}^{(e-1)}$ and $\mathcal{C}^{(e)}$, are active. Thus $\mathcal{A}$ may control up to $2t$ nodes at this time. As committees may intersect, i.e., an adversary may control a given node i in both the old and new committees. Alternatively, $\mathcal{A}$ may control node i in one committee, but not the other, reflecting either a fresh corruption or node recovery.

We provide a security proof using the framework of Universal Composition (UC) [24]. We say that a protocol π securely realizes an ideal functionality $\mathcal{F}$ if for every adversary $\mathcal{A}$ attacking the real interaction with the protocol, there is a simulator $\mathcal{S}$ interacting with the ideal functionality, such that for all environments $\mathcal{Z}$, the following is negligible:

$$|\Pr[\mathsf{REAL}(\mathcal{Z}, \mathcal{A}, \pi) = 1] - \Pr[\mathsf{IDEAL}(\mathcal{Z}, \mathcal{S}, \mathcal{F}) = 1]| \,.$$

C. Communication model

We aim to minimize communication complexity in CHURP. Specifically, we optimize for on-chain complexity and off-chain complexity in that order. We also consider the round complexity of our protocol designs, but prioritize communication complexity because blockchains—particularly permissionless ones—incur high costs for on-chain operations. We measure the communication complexity of our protocol (and related ones) in terms of *on-chain* and *off-chain* communication cost, as follows:

a) On-chain: Existing approaches such as MPSS [17] use PBFT [25] for consensus. Instead, we assume the availability of a blockchain (or other bulletin-board abstraction) to all nodes in the committee. We do this for two reasons. First, abstracting away the consensus layer results in simpler, more modular, and easier-to-understand secret-sharing protocols. Second, it makes sense to capitalize on the availability of blockchains today, rather than re-engineer their functionality.

In our model, nodes can either post a message (or) retrieve any number of messages from the blockchain. After a node posts a message to the blockchain, within a finite time period T, it gets *published*, i.e., blockchain access is *synchronous* and the message is now retrievable by any node. This channel is assumed to be *reliable*: messages posted are not lost.

Permissionless blockchains: While our techniques apply also to permissioned blockchains, we focus on permissionless blockchains—e.g., Ethereum. On such chains, users pay (heavily) for writes, but reads are free. Thus we measure on-chain communication complexity only in terms of writes, e.g., $O(n)$ on-chain cost means $O(n)$ bits written to the blockchain.

b) Off-chain: Nodes may alternatively communicate point-to-point (P2P) without direct use of the blockchain. We assume that every node has such a channel with every other node. P2P channels are also assumed to be *reliable*: all messages arrive without getting lost. We work in a *synchronous model*, i.e., any message sent via this channel will be received within a known bounded period of time, T'.

Off-chain P2P channels can be implemented in different ways depending on the deployment environment. In a decentralized setting, though, nodes are often assumed not to have P2P communication, to protect them from targeted attacks and anonymity compromise. In such cases, one can use anonymous channels, such as Tor [26], to preserve anonymity with additional setup cost and engineering complexity. Alternatively, off-chain channels can be implemented by an overlay on top of the existing blockchain infrastructure. We show how to leverage the gossip network of a blockchain system [27] for inexpensive off-chain communication in Section VII.

We measure off-chain communication complexity as the total number of bits transmitted in point-to-point channels. In general, where we refer informally to proactivization protocols' *cost* in this work, we mean their communication complexity, on-chain or off-chain, as the case may be.

III. Overview of CHURP

Now we provide an overview of CHURP, with intuition behind our core techniques. First, we briefly review two key new techniques used in CHURP: *bivariate 0-sharing* and *dimension-switching*. (We defer details until later in the paper.) Then we give an overview and example of optimistic execution of CHURP. Finally, we briefly discuss pessimistic execution paths in CHURP, i.e., what happens when nodes are faulty, and our third key technique of hedging against failures in KZG.

A. Key secret-sharing techniques

Recall that in an ordinary (t, n)-threshold Shamir secret sharing (see [4]), shares of secret s are points on a univariate polynomial $P(x)$ such that $P(0) = s$. Instead, to enable its two key techniques, CHURP employs a *bivariate* polynomial $B(x, y)$ such that $B(0, 0) = s$. A share of $B(x, y)$ is itself a univariate polynomial: Either $B(x, i)$ or $B(i, y)$ where i is the node index.

Bivariate 0-sharing: Proactivization in nearly all secret-sharing schemes involves generating a fresh, random polynomial that shares a 0-valued secret, e.g., $Q(x, y)$ such that $Q(0, 0) = 0$. This is added to the current polynomial that encodes the secret s. We call such a polynomial $Q(x, y)$ a *0-hole* polynomial and generation of this polynomial *0-sharing*. Previous approaches' main communication bottleneck is naïve 0-sharing that incurs high ($O(n^3)$ off-chain) communication complexity. Our 0-sharing protocol achieves lower ($O(n^2)$ off-chain) complexity. We give details in Section IV.

Dimension-switching: CHURP uses a bivariate polynomial $B(x, y)$ asymmetric and of *non-uniform degree*. Specifically, it uses a polynomial $B(x, y)$ of degree $\langle t, 2t\rangle$. By this, we mean that it is degree-t in x (highest term x^t) and degree-$2t$ in y (highest term y^{2t}).

This structure enables our novel *dimension-switching* technique in CHURP. Nodes can switch between a sharing in the degree-t dimension of $B(x, y)$ and the degree-$2t$ dimension. The result is a change from a (t, n)-sharing of s to a $(2t, n)$-sharing—or vice versa. As we show, dimension switching provides an efficient way to address a key challenge mentioned above. During a handover, the adversary can control up to $2t$ nodes, but between handovers, we instead want a (t, n)-threshold sharing of s. See Section V-C for details.

B. CHURP: Overview

We now give an overview of the execution of CHURP. We first consider the optimistic case, and discuss pessimistic cases below in Section III-E.

At the end of a given epoch $e-1$, before a handoff occurs, the current committee $\mathcal{C}^{(e-1)}$ is in what we call a *steady state*.

Specifically, the committee $\mathcal{C}^{(e-1)}$ holds a (t, n)-sharing of $s = B(0, 0)$. This sharing uses the degree-t dimension of $B(x, y)$, as noted above. Node $\mathcal{C}_i^{(e-1)}$ holds share $s_i = B(i, y)$, and can compute $B(x, 0)$ for $x = i$. So it is easy to see that shares s_i is actually a share in a (t, n)-sharing of $B(0, 0)$. We refer to the shares in steady state as *full shares*.

During the handoff in epoch e, nodes in the old and new committees $\mathcal{C}^{(e-1)}$ and $\mathcal{C}^{(e)}$ switch their sharing of s to the degree-$2t$ dimension of $B(x, y)$, resulting in what we call *reduced* shares.

Specifically, node $\mathcal{C}_j^{(e)}$ holds share $s_j = B(x, j)$. Node $\mathcal{C}_j^{(e)}$ can compute $B(0, y)$ for $y = j$, and consequently s_j is a share in a $(2t, n)$-sharing of $B(0, 0)$. The share s_j here has "reduced" power in the sense that $2t + 1$ of these shares (as opposed to $t + 1$ full shares in steady state) are needed to reconstruct s. Thus the adversary cannot recover s despite potentially compromising $2t$ nodes across the old and new committees $\mathcal{C}^{(e-1)}$ and $\mathcal{C}^{(e)}$.

After share reduction, the polynomial $B(x, y)$ is *proactivized*. A 0-hole bivariate polynomial $Q(x, y)$, i.e., such that $Q(0, 0) = 0$, is generated (using the new protocol given in Section IV). $Q(x, y)$ is then added to $B(x, y)$, yielding a fresh polynomial $B'(x, y) = B(x, y) + Q(x, y)$. Nodes update their reduced shares accordingly. Because $Q(x, y)$ is 0-hole, the secret s remains unchanged, i.e., $s = B'(0, 0)$.

Shares in $B'(x, y)$, i.e., for the new committee, are now *independent of those for* $B(x, y)$, i.e., for the old committee. So it is now safe to perform *full-share distribution*, i.e., to switch to the degree-t dimension of $B'(x, y)$. This involves distributing full shares to the new committee $\mathcal{C}^{(e)}$. At this point, the steady state is achieved for epoch e. Committee $\mathcal{C}^{(e)}$ holds a (t, n)-threshold sharing of s using $B'(x, y)$.

To summarize, the three phases in the CHURP handoff are:

- *Share reduction*: A switch is made from the degree-t dimension of $B(x, y)$ to the degree-$2t$ dimension. As a result, each node $\mathcal{C}_j^{(e)}$ in the new committee obtains a reduced share $B(x, j)$.

- *Proactivization*: The new committee generates $Q(x, y)$ such that $Q(0, 0) = 0$, and each node $\mathcal{C}_j^{(e)}$ obtains a reduced share: $B'(x, j) = B(x, j) + Q(x, j)$. Proactivization ensures that shares in the new committee are independent of those in the old.
- *Full-share distribution*: New shares $B'(i, y)$ are generated from reduced shares $\{B'(x, j)\}_j$, by switching back to the degree-t dimension of $B'(x, y)$. The protocol thus returns to its steady state.

Note that during the handoff, the old committee $\mathcal{C}^{(e-1)}$ can still perform operations using s, even if some nodes have left. So there is no operational discontinuity in CHURP.

C. An example

In Fig. 3, we show a simple example of the handoff protocol in CHURP assuming all nodes are honest. The old committee consists of three nodes $\mathcal{C}^{(e-1)} = \{A_1, A_2, A_3\}$. A_3 leaves at the end of the epoch, and a new node A'_3 joins. The new committee is thus $\mathcal{C}^{(e)} = \{A_1, A_2, A'_3\}$. The underlying polynomial $B(x, y)$ is thus of degree $\langle 1, 2\rangle$. Node A_i's share is $B(i, y)$ or 3 points: $B(i, 1), B(i, 2)$ and $B(i, 3)$. The figure depicts the three phases of the handoff protocol, as follows.

a) Share reduction: To start the handoff, each node j in the new committee constructs its reduced share $B(x, j)$ from points received from $\mathcal{C}^{(e-1)}$. As shown in the figure, node A'_3 receives points $B(1, 3)$ and $B(2, 3)$ from A_1 and A_2 respectively, from which $B(x, 3)$ can be constructed. Similarly, A_1 and A_2 construct $B(x, 1)$ and $B(x, 2)$ respectively.

b) Proactivization: Having reconstructed reduced shares $\{B(x, j)\}_j$, nodes in the new committee collectively generate a 0-hole bivariate polynomial $Q(x, y)$ of degree $\langle t, 2t\rangle$, with the constraint that each j only learns $Q(x, j)$. Reduced shares are updated as $B'(x, j) = B(x, j) + Q(x, j)$. In the example above, node j ends up with $Q(x, j)$ of a random 0-hole polynomial $Q(x, y)$.

c) Full-share distribution: Nodes in the new committee get their full shares from the updated reduced shares. Take A_1 as an example. By this point, A_1 has $B'(x, 1)$ and sends $B'(i, 1)$ to A_i for $i \in \{2, 3\}$. Other nodes do the same. Hence, A_1 receives $B'(1, 2)$ and $B'(1, 3)$ from A_2 and A'_3 respectively. It now has the necessary three points $\{B'(1, j)\}_{j\in[3]}$ in order to interpolate its full share $B'(1, y)$.

D. Adaptive security

As noted before, the above example assumes an honest-but-curious adversary. Additional machinery in the form of cryptographic proofs of correctness for node communications—detailed in Section V-C—are required to provide security against an active and adaptive adversary. These proofs do not alter the overall structure of the protocol.

E. Pessimistic CHURP execution paths

What we have described thus far is an optimistic execution of CHURP. This corresponds to a subprotocol Opt-CHURP that is highly efficient and optimistic: it only completes when all nodes are honest and the assumptions of the KZG scheme hold.

When things go wrong, CHURP can detect the violation and resort to pessimistic paths. Specifically, Exp-CHURP-A can hold malicious nodes accountable. Moreover, CHURP can also efficiently detect any soundness failure of the KZG scheme, due to either a compromised trusted setup or a falsified hardness assumption (t-SDH). When detected, CHURP switches to Exp-CHURP-B that only relies on DL and no trusted setup.

As noted above, the on-chain / off-chain communication complexity of CHURP is $O(n)$ / $O(n^2)$ in the optimistic case, and $O(n^2)$ / $O(n^3)$ for the two pessimistic paths. Opt-CHURP and Exp-CHURP-A requires $t < n/2$, while Exp-CHURP-B requires $t < n/3$. We give details on the optimistic and pessimistic paths in CHURP in Section V.

IV. Efficient Bivariate 0-Sharing

In this section, we introduce our technique for efficient 0-sharing of bivariate polynomials. It is a key new building block in CHURP, used in the proactivization phase of the handoff.

Recall that in the context of bivariate polynomials, 0-sharing means having a committee $\mathcal{C}$ generate a $\langle t, 2t\rangle$-bivariate polynomial $Q(x, y)$ such that $Q(0, 0) = 0$. Each node $\mathcal{C}_i$ holds a share $Q(i, y)$.

Previous works have naïvely extended 0-sharing techniques for univariate polynomials to the bivariate case: Each node generates its own 0-hole bivariate polynomial Q_i i.e., $Q_i(0, 0) = 0$, and distributes points on it. Thus each node transmits $O(n)$ univariate polynomials, resulting in $O(n^2)$ off-chain communication complexity per node, and $O(n^3)$ in total.

Our new technique, specified as protocol BivariateZeroShare, brings the total off-chain communication complexity down to just $O(tn)$ in the optimistic case. In the pessimistic case, i.e., if a node is caught cheating, different protocols (see Section V) must then be invoked. Even in the pessimistic case, though, our techniques incur no more communication complexity than in previous schemes: $O(n^3)$ in the dynamic setting and $O(n^2)$ in the static Herzberg et al. setting.

BivariateZeroShare comprises two steps. In the first step, a 0-sharing subprotocol UnivariateZeroShare is executed among a subset $\mathcal{U}$ of $2t + 1$ nodes. At the end of this step, each node $\mathcal{U}_j$ holds a share s_j of a univariate polynomial $P(x)$. In the second step, each node in $\mathcal{U}$ reshares its share s_j among all nodes, i.e., the full committee. Each node $\mathcal{C}_i$ thereby obtains share $Q(i, y)$ of bivariate polynomial $Q(x, y)$, as desired.

BivariateZeroShare is formally specified in Fig. 15. (For the interest of space, we present all protocols formally in the appendix. Nonetheless, the text description here is sufficient to understand the paper.) For ease of presentation, we describe an honest-but-curious protocol version in this section. Our full protocol, which is secure against active, adaptive adversaries, is detailed in Section V-C.

a) First step—Sharing $P(x)$: As noted, BivariateZeroShare first chooses a subset $\mathcal{U} \subseteq \mathcal{C}$ of $2t + 1$ nodes, i.e., $|\mathcal{U}| = 2t + 1$. This can be done as follows: Order nodes lexicographically by their public keys and choose the first $2t + 1$. Without loss of generality, $\mathcal{U} = \{\mathcal{C}_j\}_{j=1}^{2t+1}$.

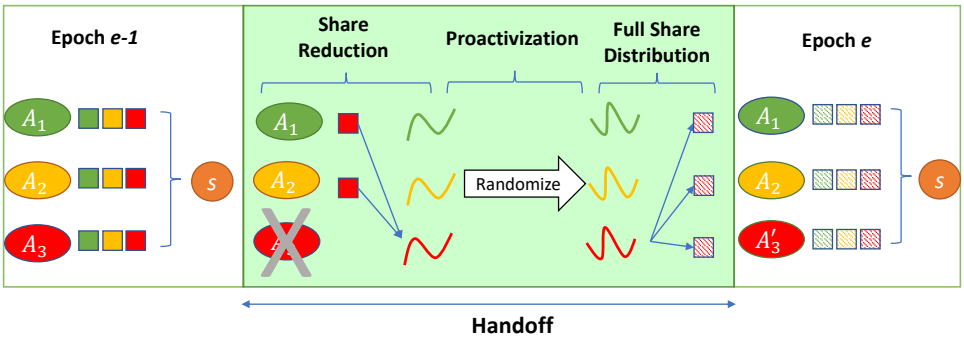


Fig. 3: An example of the handoff protocol: Curves denote univariate polynomials (reduced shares) while squares denote points on these polynomials. See Section III-C for a description.

The nodes of $\mathcal{U}$ then execute the univariate 0-sharing subprotocol UnivariateZeroShare presented in Fig. 14. This subprotocol is not new—it was previously used for proactivization in [5]. Each node $\mathcal{U}_j$ generates a degree-$2t$ univariate 0-hole polynomial $P_j(x)$. The sum $P(x) = \sum_{j=1}^{2t+1} P_j(x)$ is itself a degree-$2t$ univariate 0-hole polynomial $P(x)$. Then, $\mathcal{U}_j$ redistributes points on its local polynomial $P_j(x)$, enabling every $\mathcal{U}_i$ at the end of the step to compute its share $s_i = P(i)$.[1]

b) Second step—Resharing $P(x)$: Nodes in $\mathcal{U}$ now reshare $P(x)$ among all of $\mathcal{C}$, resulting in a sharing of the desired bivariate polynomial $Q(x, y)$.

Each node $\mathcal{U}_j$ generates a degree-t univariate polynomial $R_j(x)$ uniformly at random under the constraint $R_j(0) = s_j$, i.e., $R_j(x)$ encodes the node's share s_j. Together, the $2t + 1$ degree-t polynomials $\{R_j(x)\}$ uniquely define a degree-$\langle t, 2t\rangle$ bivariate polynomial $Q(x, y)$ such that $Q(x, j) = R_j(x)$ for $j = 1, 2, \ldots, 2t + 1$ and $Q(0, 0) = 0$.

Node $\mathcal{U}_j$ sends $R_j(i) = Q(i, j)$ to every other node $\mathcal{C}_i$ in the full committee. Using the received points, each committee member $\mathcal{C}_i$ interpolates to compute its share—a $2t$-degree polynomial $Q(i, y)$. The constraint $Q(0, 0) = 0$ is satisfied because the zero coefficients of $R_j(x)$ are composed of shares generated from the 0-sharing step before, i.e., UnivariateZeroShare. Since each node in $\mathcal{U}$ transmits n points, the overall cost incurred is just $O(tn)$ off-chain.

We use (t, n)-BivariateZeroShare as a subroutine in CHURP with some modifications. As explained before, it can also reduce the off-chain communication complexity of Herzberg et al.'s PSS scheme [5], i.e., the static-committee setting, by a factor of $O(n)$. Due to lack of space, we present this application in the technical report, the formal protocol can be found in the Appendix (See Appendix C).

V. CHURP Protocol Details

CHURP consists of a suite of tiered protocols with different trust assumptions and communication complexity. The execution starts at the top tier—a highly efficient optimistic protocol. Only upon detection of adversarial misbehavior, does the execution fall back to lower tiers. The three tiers of CHURP and their relationship are shown in Fig. 4, detailed as below.

The top tier, Opt-CHURP, is the default protocol of CHURP. It is optimistic and highly efficient: if no node misbehaves, the execution completes incurring only $O(n)$ on-chain and $O(n^2)$ off-chain cost. As a design choice, Opt-CHURP does not identify faulty nodes but rather just detects faulty behavior, upon which the execution switches to a lower tier protocol, also referred to as a pessimistic path.

The second tier is Exp-CHURP-A, the main pessimistic path of CHURP. Unlike Opt-CHURP, Exp-CHURP-A is able to identify and hence expel faulty nodes using proofs of correctness. Exp-CHURP-A trades performance for robustness: the execution is guaranteed to complete as long as the adversarial threshold $t < n/2$, but incurs $O(n^2)$ on-chain and $O(n^3)$ off-chain worst-case cost.

Both Opt-CHURP and Exp-CHURP-A use KZG commitments to achieve $t < n/2$. As noted before, this commitment scheme requires a trusted setup phase to generate public keys with a trapdoor. The trapdoor must be "destroyed" after the setup; otherwise soundness is lost. KZG introduces the only trusted setup in CHURP, and thus represents its main protocol-level vulnerability. KZG also relies on a non-standard hardness assumption, the t-Strong Diffie-Hellman assumption (t-SDH).

To hedge against soundness failure in KZG (either due to a falsified trust assumption or a compromised trusted setup), we introduce an additional verification step (StateVerif), which can be executed at the end of Opt-CHURP or Exp-CHURP-A. StateVerif is highly efficient—incurs only $O(n)$ on-chain complexity. Any fault detected by StateVerif indicates that KZG is unusable, and triggers a KZG-free pessimistic path named Exp-CHURP-B. Exp-CHURP-B has the same cost as Exp-CHURP-A, but one drawback: It tolerates a lower adversarial threshold, $t < n/3$. We defer the details of StateVerif to Appendix D-B.

In summary, the three tiers (subprotocols) of CHURP are:

1) Opt-CHURP: The default protocol of CHURP. It incurs $O(n)$ on-chain and $O(n^2)$ off-chain communication complexity under the optimal resilience bound $t < n/2$.
2) Exp-CHURP-A: Invoked if Opt-CHURP fails. It incurs $O(n^2)$ on-chain and $O(n^3)$ off-chain communication

[1] An attack is outlined in [28] that breaks the original variant of UnivariateZeroShare in [5]. It does so by considering an adversarial model similar to ours, where the adversary controls t nodes in old and new committees and thus $2t$ in total, rather than t in total as in [5]. CHURP defeats this attack via dimension-switching, using reduced shares during the handoff.

complexity under the optimal bound $t < n/2$.

3) Exp-CHURP-B: Invoked if a soundness breach of KZG is detected by StateVerif. It incurs the same cost as Exp-CHURP-A, but requires $t < n/3$.

Table II summarizes the three tiers. Due to space constraints, we present only Opt-CHURP in the body of the paper and present Exp-CHURP-A and Exp-CHURP-B in Appendix D.

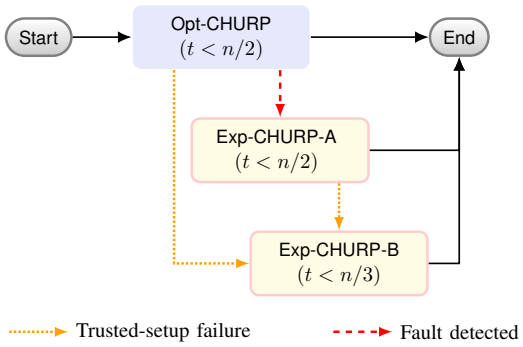


Fig. 4: CHURP protocol tiers. Opt-CHURP is the default protocol of CHURP. Exp-CHURP-A and Exp-CHURP-B are run only if a fault occurs in Opt-CHURP.

A. Notation and Invariants

We now introduce the notation and invariants that will be used to explain the protocols of CHURP. Notation introduced in this section is summarized in Table I.

a) KZG polynomial commitments: KZG commitment allows a prover to commit to a polynomial $P(x)$ and later prove the correct evaluation $P(i)$ to a verifier. Further details can be found in Appendix A and [22].

b) CHURP invariants: We say the system arrives at a *steady state* after it completes a successful handoff. The following invariants stipulate the desired properties of a steady state. We use invariants to explain the protocol and reason about its security.

Let $\mathcal{C}$ be a committee of n nodes $\{\mathcal{C}_i\}_{i=1}^{n}$. Let $B(x, y)$ denote the asymmetric bivariate polynomial of degree $\langle t, 2t\rangle$ used to share the secret s, i.e., $s = B(0, 0)$. In a steady state, the following three invariants must hold:

- Inv-Secret: The secret s is the same across handoffs.
- Inv-State: Each node $\mathcal{C}_i$ holds a full share $B(i, y)$ and a proof to the correctness thereof. Specifically, the full share $B(i, y)$ is a degree-$2t$ polynomial, and hence can be uniquely represented by $2t + 1$ points $\{B(i, j)\}_{j=1}^{2t+1}$. The proof is a set of witnesses $\{W_{B(i,j)}\}_{j=1}^{2t+1}$.
- Inv-Comm: KZG commitments to reduced shares ($\{B(x, j)\}_{j=1}^{2t+1}$) are available to all nodes.

The first invariant Inv-Secret ensures the secret remains unchanged, a core functionality of CHURP.

Inv-State and Inv-Comm ensures the correctness of the protocol. For example, recall from Section III that during the handoff (the Share Reduction phase), nodes in the old and the new committee switch their dimension of sharing, from full shares to reduced shares. Using the commitments (specified by Inv-Comm) and the witnesses (specified by Inv-State), new committee members can verify the correctness of reduced shares, thus the correctness of dimension-switching.

Note that to realize Inv-Comm, hashes of KZG commitments are put on-chain for consensus while the commitments are transmitted off-chain between nodes.

Opt-ShareReduce

1 : **Public Input:** $\{C_{B(x,j)}\}_{j\in[2t+1]}$

2 : **Input:** Set of nodes $\{\mathcal{C}_i\}_{i\in[n]}$ where each node $\mathcal{C}_i$ is given $\{B(i,j), W_{B(i,j)}\}_{j\in[2t+1]}$. Set of nodes $\{\mathcal{C}'_j\}_{j\in[n']}$ s.t. $n' \geq 2t+1$

3 : **Output:** $\forall j \in [2t+1]$, node $\mathcal{C}'_j$ output $B(x, j)$

4 : Order $\{\mathcal{C}'_j\}$ based on lexicographic order of their public keys

5 : Choose the first $2t+1$ nodes, denoted as $\mathcal{U}'$, w.l.o.g., $\mathcal{U}' = \{\mathcal{C}'_j\}_{j\in[2t+1]}$

6 : node $\mathcal{C}_i$:

7 : $\forall j \in [2t+1]$, send a point and witness $\{B(i,j), W_{B(i,j)}\}$ to $\mathcal{U}'_j$ off-chain

8 : node $\mathcal{U}'_j$:

9 : Wait and receive n points and witnesses, $\{B(i,j), W_{B(i,j)}\}_{i\in[n]}$

10 : $\forall i \in [n]$, invoke VerifyEval$(C_{B(x,j)}, i, B(i,j), W_{B(i,j)})$

11 : Interpolate any $t+1$ verified points to construct $B(x, j)$

Fig. 5: Opt-ShareReduce between the committees $\mathcal{C}$ and $\mathcal{C}'$.

Notation	Description
$\mathcal{C}^{(e-1)}, \mathcal{C}^{(e)}$	Old, New committee
$B(x, y)$	Bivariate polynomial used to share the secret
$\langle t, k\rangle$	Degree of $\langle x, y\rangle$ terms in B
$RS_i(x) = B(x, i)$	Reduced share held by $\mathcal{C}_i$
$FS_i(y) = B(i, y)$	Full share held by $\mathcal{C}_i$'s
$C_{B(x,j)}$	KZG commitment to $B(x, j)$
$W_{B(i,j)}$	Witness to evaluation of $B(x, j)$ at i
$Q(x, y)$	Bivariate proactivization polynomial
$\mathcal{U}'$	Subset of nodes chosen to participate in handoff
λ_i	Lagrange coefficients

TABLE I: Notation

B. CHURP Setup

The setup phase of CHURP sets the system to a proper initial steady state. To start, an initial committee $\mathcal{C}^{(0)}$ is selected. The setup of KZG is performed and the secret is shared among $\mathcal{C}^{(0)}$. Using their shares, members of $\mathcal{C}^{(0)}$ can generate commitments to install the three invariants.

The setup of KZG can be performed by a trusted party or a committee assuming at least one of them is honest. The secret to be managed by CHURP can be generated by a trusted party or in a distributed fashion using, e.g., [29].

We leave committee selection out-of-scope for this paper. Readers can refer to, e.g., [30], for a discussion.

C. CHURP Optimistic Path (Opt-CHURP)

Recall that Opt-CHURP transfers shares of some secret s from an old committee, denoted $\mathcal{C} = \mathcal{C}^{(e-1)}$, to a new committee $\mathcal{C}' = \mathcal{C}^{(e)}$. CHURP can support both committee-size and threshold changes, i.e., a transition from (n, t) to some

Opt-Proactivize

Public Input: $\{C_{B(x,j)}\}_{j\in[2t+1]}$

Input: Set of nodes $\{\mathcal{C}'_i\}_{i\in[n']}$. Let $\mathcal{U}' = \{\mathcal{C}'_j\}_{j\in[2t+1]}$, each node $\mathcal{U}'_j$ is given $B(x,j)$

Output: $\mathcal{U}'_j$ outputs `success` and $B'(x,j)$ for a degree-$\langle t,k\rangle$ bivariate polynomial $B'(x,y)$ with $B'(0,0) = B(0,0)$ (or) `fail`

Public Output: $\{C_{B'(x,j)}\}_{j\in[2t+1]}$

Invoke $(2t, 2t+1)$-UnivariateZeroShare among the nodes $\{\mathcal{U}'_j\}_{j\in[2t+1]}$ to generate shares $\{s_j\}_{j\in[2t+1]}$

node $\mathcal{U}'_j$:

  Generate random t-degree polynomial $R_j(x)$ such that $R_j(0) = s_j$

Denote the bivariate polynomial $Q(x,y)$ where $\{Q(x,j) = R_j(x)\}_{j\in[2t+1]}$

Denote the bivariate polynomial $B'(x,y) = B(x,y) + Q(x,y)$

node $\mathcal{U}'_j$:

  Compute $B'(x,j) = B(x,j) + Q(x,j)$ and $Z_j(x) = R_j(x) - s_j$

  Send $\{g^{s_j}, C_{Z_j}, W_{Z_j(0)}, C_{B'(x,j)}\}$ off-chain to all nodes in $\mathcal{C}'$, where $C_{Z_j} = \mathsf{Commit}(Z_j)$; $W_{Z_j(0)} = \mathsf{CreateWitness}(Z_j, 0)$; $C_{B'(x,j)} = \mathsf{Commit}(B'(x,j))$

  Publish hash of the commitments on-chain $H_j = H(g^{s_j}||C_{Z_j}||W_{Z_j(0)}||C_{B'(x,j)})$

node $\mathcal{C}'_i$:

  $\forall j \in [2t+1]$, retrieve on-chain hash H_j, also receive $\{g^{s_j}, C_{Z_j}, W_{Z_j(0)}, C_{B'(x,j)}\}$ off-chain

  $\forall j \in [2t+1]$, if $H_j \neq H(g^{s_j}||C_{Z_j}||W_{Z_j(0)}||C_{B'(x,j)})$ or $\mathsf{VerifyEval}(C_{Z_j}, 0, 0, W_{Z_j(0)}) \neq \mathsf{True}$ or $C_{B'(x,j)} \neq C_{B(x,j)} \times C_{Z_j} \times g^{s_j}$, output `fail`

  Using Lagrange coefficients in Eq. (1), if $\prod_{j=1}^{2t+1}(g^{s_j})^{\lambda_j^{2t}} \neq 1$ output `fail`

node $\mathcal{U}'_j$:

  Output `success` and $B'(x,j)$

Fig. 6: Opt-Proactivize updates the reduced shares.

Opt-ShareDist

Public Input: $\{C_{B'(x,j)}\}_{j\in[2t+1]}$

Input: Set of nodes $\{\mathcal{C}'_i\}_{i\in[n']}$. Let $\mathcal{U}' = \{\mathcal{C}'_j\}_{j\in[2t+1]}$, each node $\mathcal{U}'_j$ is given $B'(x,j)$

Output: $\forall i \in [n']$, $\mathcal{C}'_i$ outputs `success` and $B'(i,y)$ (or) `fail`

node $\mathcal{U}'_j$:

  $\forall i \in [n']$, send a point and witness off-chain $\{B'(i,j), W'_{B(i,j)}\}$ to $\mathcal{C}'_i$ where $W'_{B(i,j)} = \mathsf{CreateWitness}(B'(x,j), i)$

node $\mathcal{C}'_i$:

  Wait and receive points and witnesses $\{B'(i,j), W'_{B(i,j)}\}_{j\in[2t+1]}$

  $\forall j \in [2t+1]$, invoke $\mathsf{VerifyEval}(C_{B'(x,j)}, i, B'(i,j), W'_{B(i,j)})$

  If all $2t+1$ points are correct, interpolate to construct $B'(i,y)$

  Output `success` and the full share $B'(i,y)$

  In all other cases, output `fail`

Fig. 7: Opt-ShareDist uses the updated reduced shares to distribute full shares in $\mathcal{C}'$.

(n', t') in any epoch. For ease of exposition here, though, we allow n to change across epochs assuming a constant threshold t. Changing the threshold is discussed in Appendix E.

Opt-CHURP proceeds in three phases. The first phase, Opt-ShareReduce, performs dimension-switching to tolerate an adversary capable of compromising $2t$ nodes across the old and new committees. By the end of this phase, reduced shares are constructed by members of the new committee. The second phase, Opt-Proactivize, proactivizes these reduced shares so that new shares are independent of the old ones. The third and the final phase, Opt-ShareDist, restores full shares from reduced shares, and thus completes a return to the steady state.

At the beginning of Opt-CHURP, each node in $\mathcal{C}'$ requests the set of KZG commitments from any node in $\mathcal{C}$, say $\mathcal{C}_1$. Recall that by the invariant Inv-Comm, each node in $\mathcal{C}$ holds the KZG commitments to the current reduced shares, $\{C_{B(x,j)}\}_{j=1}^{2t+1}$, while the corresponding hashes are on-chain. The received commitments are verified using the on-chain hashes. Optimistically, each node in $\mathcal{C}'$ receives the correct set of commitments. If a node receives corrupt ones, we switch to a pessimistic path where the KZG commitments are published on-chain. The phases of Opt-CHURP in detail are as follows:

*1) Share Reduction (*Opt-ShareReduce*):* The protocol starts by choosing a subset $\mathcal{U}' \subseteq \mathcal{C}'$ of $2t+1$ members (possible because $|\mathcal{C}'| > 2t$). The nodes in $\mathcal{U}'$ are denoted $\{\mathcal{U}'_j\}_{j=1}^{2t+1}$.

Some members in the old committee $\mathcal{C}$ may have left the protocol by this point. Let $\mathcal{C}_{alive} \subseteq \mathcal{C}$ denote the subset of nodes that are present, w.l.o.g., let this subset be $\{\mathcal{C}_i\}_{i=1}^{|\mathcal{C}_{alive}|}$.

Recall that by the invariant Inv-State, each node $\mathcal{C}_i$ holds a full share $B(i,y)$. Now, $\mathcal{C}_i$ distributes points on its full share allowing computation of reduced shares $B(x,j)$ by all members of $\mathcal{U}'$—making a *dimension-switch* from the degree-t dimension of $B(x,y)$ to the degree-$2t$ dimension. Specifically, $\mathcal{C}_i$ sends $B(i,j)$ to $\mathcal{U}'_j$, which interpolates the received points to get its reduced share $B(x,j)$. Note that in the optimistic path we require all $2t+1$ nodes in $\mathcal{U}'$ to participate. If any adversarial nodes fail to do so, we switch to a pessimistic path as detailed above.

The received points are accompanied by witnesses allowing for verification using the KZG commitments received previously. Since $t+1$ correct points are sufficient to reconstruct the reduced share, we need at least $2t+1$ points ($|\mathcal{C}_{alive}| > 2t$) to guarantee liveness.

The size of $\mathcal{C}_{alive}$ is governed by the bounded churn rate α, i.e., $|\mathcal{C}_{alive}| \geq |\mathcal{C}|(1-\alpha)$. Thus, the condition for liveness, $|\mathcal{C}_{alive}| > 2t$, places a lower bound on the committee size, $|\mathcal{C}|(1-\alpha) > 2t$ or $|\mathcal{C}| > \lfloor 2t/1-\alpha \rfloor$.

The protocol Opt-ShareReduce is formally specified in Fig. 5. At the end of Opt-ShareReduce, dimension-switching is complete and each node $\mathcal{U}'_j$ has a reduced share $B(x,j)$.

Communication complexity: Each node in $\mathcal{U}'$ receives $O(n)$ points, so Opt-ShareReduce incurs $O(nt)$ off-chain cost.

*2) Proactivization (*Opt-Proactivize*):* In this phase, $\mathcal{U}'$ proactivizes the bivariate polynomial $B(x,y)$—a key step in generating new shares independent of the old ones held by members of $\mathcal{C}$. The polynomial $B(x,y)$ is updated using a random bivariate polynomial $Q(x,y)$ generated such that $Q(0,0) = 0$. The result is a new polynomial $B'(x,y) = B(x,y) + Q(x,y)$. The fact that $Q(0,0) = 0$ ensures preservation of our first invariant Inv-Secret.

We achieve this by adapting the bivariate 0-sharing technique (BivariateZeroShare) presented in Section IV to handle active adversaries. Recall that BivariateZeroShare comprises two steps. First, a univariate 0-sharing subroutine generates shares of the number 0. These shares are then re-shared in a second step resulting in a sharing of $Q(x, y)$ among $\mathcal{C}'$.

By the end of the previous, i.e., Share Reduction phase, every node $\mathcal{U}'_j$ in the set of $2t+1$ nodes $\mathcal{U}'$ holds a reduced share $B(x, j)$. Now, by the end of the current, i.e., Proactivization phase, we update these reduced shares by adding $Q(x, j)$ from the generated bivariate polynomial $Q(x, y)$.

The protocol starts by invoking the 0-sharing subroutine UnivariateZeroShare introduced previously, which is the first step of BivariateZeroShare. Specifically, $(2t, 2t + 1)$-UnivariateZeroShare is run among $\mathcal{U}'$ to generate shares s_j at each $\mathcal{U}'_j$. To handle active adversaries, $\mathcal{U}'_j$ sends a commitment to the share, g^{s_j}, to all other nodes in $\mathcal{U}'$ (where g is a publicly known generator). Lagrange coefficients $\{\lambda_j^{2t}\}_j$ can be precomputed to interpolate and verify if the shares form a 0-sharing, $\sum_{j=1}^{2t+1} \lambda_j^{2t} s_j = 0$. Translating it to the commitments, all nodes check the following:

$$\prod_{j=1}^{2t+1} (g^{s_j})^{\lambda_j^{2t}} = 1 \ . \tag{1}$$

Then, $\mathcal{U}'_j$ generates a random degree-t univariate polynomial $R_j(x)$ that encodes the node's share s_j, i.e., $R_j(0) = s_j$. Together, the $2t + 1$ polynomials uniquely define a 0-hole bivariate polynomial $Q(x, y)$ such that $\{Q(x, j) = R_j(x)\}_{j=1}^{2t+1}$. $\mathcal{U}'_j$ also updates the reduced share, $B'(x, j) = B(x, j)+R_j(x)$. Points on $B'(x, j)$ will be distributed to the entire committee $\mathcal{C}'$ in the next phase of Opt-CHURP. (We make a modification to BivariateZeroShare: In the re-sharing step of BivariateZeroShare, points on $Q(x, j)$ were distributed directly.)

Each $\mathcal{U}'_j$ sends constant-size information to other nodes off-chain enabling verification of the above step. Let $Z_j(x) = R_j(x) - s_j$ denote a 0-hole polynomial, the commitment to $Z_j(x)$, C_{Z_j}, and a witness to the evaluation at zero are distributed enabling verification of the statement: $Z_j(0) = 0$; equivalent to $R_j(0) = s_j$. The commitment to the updated reduced share $B'(x, j)$ is also distributed. Since $B'(x, j) = B(x, j) + Z_j + s_j$, the homomorphic property of the commitment scheme allows other nodes to verify if $C_{B'(x,j)} = C_{B(x,j)} \times C_{Z_j} \times C_{s_j}$ where $C_{s_j} = g^{s_j}$ and the other two commitments were received previously.

In total, each $\mathcal{U}'_j$ generates the following set of commitment and witness information during Opt-Proactivize, $\{g^{s_j}, C_{Z_j}, W_{Z_j(0)}, C_{B'(x,j)}\}$. While this set is transmitted off-chain to all nodes in the full committee $\mathcal{C}'$, a hash of it is published on-chain. The received commitments can then be verified using the published hash, thereby ensuring that everyone receives the same commitments. Note that the set of commitments is sent to $\mathcal{C}'$ instead of just the subset $\mathcal{U}'$ to preserve the invariant Inv-Comm, i.e., ensure that all nodes hold KZG commitments to the updated reduced shares.

The verification mechanisms used in this protocol are sufficient to detect any faulty behavior, although they do not identify which nodes are faulty. Thus, the adversary can disrupt the protocol without revealing his / her nodes. For example, it could send corrupt commitments to nodes selectively. Although the published hash reveals this, a verifiable accusation cannot be made since the commitments were sent off-chain. Another example would be a corrupt node sending points from a non-0-hole polynomial in the UnivariateZeroShare protocol. Again, we detect such a fault but cannot identify which nodes are faulty. So detection of a fault simply leads to a switch to the pessimistic path, Exp-CHURP-A. While Exp-CHURP-A is capable of identifying misbehaving nodes, note that we do *not* retroactively identify the faulty nodes from Opt-CHURP.

The protocol Opt-Proactivize is formally specified in Fig. 6. By the end of this, if no faults are detected, each $\mathcal{U}'_j$ holds $B'(x, j)$. The invariants Inv-Secret and Inv-Comm hold as $s = B'(0, 0)$ and all of $\mathcal{C}'$ hold the KZG commitments respectively. In the next phase, we preserve the other invariant Inv-State.

Communication complexity: Each node in $\mathcal{U}'$ publishes a hash on-chain and transmits $O(t)$ data off-chain. Hence, Opt-Proactivize incurs $O(t)$ on-chain and $O(t^2)$ off-chain cost.

*3) Full Share Distribution (*Opt-ShareDist*):* In the final phase, full shares are distributed to all members of the new committee, thus preserving the Inv-State invariant. A successful completion of this phase marks the end of handoff.

By the end of the previous phase, each $\mathcal{U}'_j$ in the chosen subset of nodes $\mathcal{U}' \subseteq \mathcal{C}'$ holds a new reduced share $B'(x, j)$.

Now, $\mathcal{U}'_j$ distributes points on $B'(x, j)$, allowing computation of full shares $B'(i, y)$ by all members of $\mathcal{C}'$—we make a *dimension-switch* from the degree-$2t$ dimension of $B'(x, y)$ to the degree-t dimension. Specifically, each $\mathcal{C}'_i$ receives $2t + 1$ points $\{B'(i, j)\}_{j=1}^{2t+1}$, which can be interpolated to compute $B'(i, y)$, its full share. This is made verifiable by sending witness along with the points.

Since the point distribution is off-chain, a faulty node can send corrupt points without getting identified similar to the previous phase. In this event, we switch to the pessimistic path Exp-CHURP-A without identifying which nodes are faulty.

The protocol Opt-ShareDist is formally specified in Fig. 7. If all nodes receive correct points, this phase ends successfully and the optimistic path ends. The remaining invariant Inv-State is fulfilled as each node in $\mathcal{C}'$ receives a full share, and hence the system returns to the steady state. After a successful completion of CHURP, we require that members of the old committee $\mathcal{C}$ delete their old full shares and members of $\mathcal{U}'$ delete their new reduced shares.

Communication complexity: Each node in $\mathcal{C}'$ receives $2t + 1$ points, thus Opt-ShareDist incurs $O(nt)$ off-chain cost (assuming $O(n') = O(n)$).

Each of the three phases in Opt-CHURP (and thus Opt-CHURP itself) incur no more than $O(n)$ on-chain and $O(n^2)$ off-chain cost assuming $O(t) = O(n)$. In terms of round complexity, it completes in three rounds (one for each phase) that does not depend on the committee size. Due to lack of space,

Protocol	On-chain, Off-chain	Threshold	Optimistic
Opt-CHURP	$O(n), O(n^2)$	$t < n/2$	Yes
Exp-CHURP-A	$O(n^2), O(n^3)$	$t < n/2$	No
Exp-CHURP-B	$O(n^2), O(n^3)$	$t < n/3$	No
Opt-Schultz-MPSS	$O(n), O(n^4)$	$t < n/3$	Yes
Schultz-MPSS	$O(n^2), O(n^4)$	$t < n/3$	No

TABLE II: On-chain cost and Off-chain costs of all protocols for the dynamic setting. An optimistic protocol ends successfully only if no faulty behavior is detected.

we reiterate that the pessimistic paths of CHURP are discussed in Appendix D. Table II compares on-chain and off-chain costs of the three paths of CHURP and Schultz-MPSS [17], the latter will be explained in more detail in Section VI-C.

Note on security: Note that the adversary can learn at most $2t$ reduced shares and t full shares of $B^{(e)}(x, y)$ for a given epoch e. Before Opt-CHURP ends, the adversary learns t full shares and t reduced shares of epoch e from t corrupted nodes in $\mathcal{C}^{(e)}$. In the subsequent handoff between committees $\mathcal{C}^{(e)}$ and $\mathcal{C}^{(e+1)}$, the adversary learns (at most) another t reduced shares for epoch e, during the dimension-switch of the Opt-ShareReduce phase, leading to a total worst-case leakage of $2t$ reduced shares and t full shares of $B^{(e)}(x, y)$. It can be shown that the above leakage is not enough to compute $B^{(e)}(x, y)$. Since shares are proactivized in the handoff, subsequent corruptions do not leak any more information about $B^{(e)}(x, y)$.

Theorem 1. *Protocol* Opt-CHURP *securely realizes the functionality* $\mathcal{F}_{\mathsf{Opt-CHURP}}$*, under the* t*-strong Diffie-Hellman assumption (*t*-SDH) of the* KZG *scheme.*

We give the ideal functionality $\mathcal{F}_{\mathsf{Opt-CHURP}}$ and the simulator $\mathcal{S}$ in Appendix A, and show that it is indistinguishable from the real world, which proves the security of our protocol Opt-CHURP as per our security definition.

VI. CHURP Implementation & Evaluation

We now report on implementation and experiments with CHURP, including a comparison with the state-of-the-art alternative, Schultz-MPSS [17].

A. Implementation

We implemented Opt-CHURP in about $2,100$ lines of Go code. Our implementation uses the GNU Multiprecision Library [31] and the Pairing-Based Cryptography Library [32] for cryptographic primitives, and gRPC [33] for network infrastructure. We plan to open-source our code for CHURP in the near future.

For polynomial arithmetic, we used the polynomial ring $\mathbb{F}_p[x]$ for a 256-bit prime p. For the KZG commitment scheme, we used a type A pairing on an elliptic curve $y^2 = x^3 + x$ over $\mathbb{F}_q$ for a 512-bit q. The order of the EC group is also p. We use SHA256 for hashing.

Blockchain Simulation: CHURP can be deployed on both permissioned and permissionless blockchains. To abstract away the specific choice, we simulate one using a trusted node. Note that when deployed in the wild, writing to the blockchain would incur an additional constant latency.

B. Evaluation

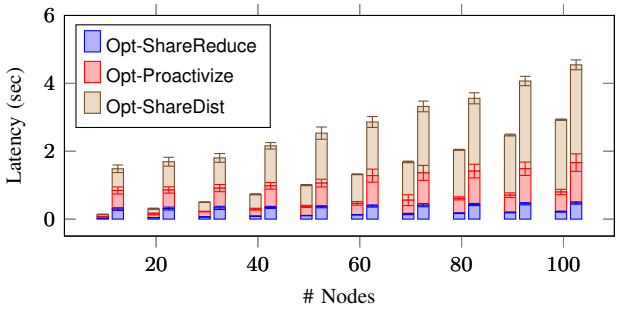
In our evaluation, experiments are run in a distributed network of up to 1000 EC2 `c5.large` instances, each with 2 vCPU and 4GB of memory. Each instance acts as a node in the committee and the handoff protocol is executed assuming a static committee. All experiments are averaged over 1000 epochs, i.e., 1000 invocations of Opt-CHURP. We measure three metrics for each epoch of the protocol: the latency (the total execution time), the on-chain communication complexity (the total bytes written to the blockchain (i.e. the trusted node)), and the off-chain communication complexity (the total bytes transmitted between all nodes). The evaluation results are presented below.

1) Latency: In the first set of experiments, all EC2 instances belong to the same region, also referred to as the LAN setting. This setting is useful to understand the time spent in computation. Fig. 8 shows the latency of Opt-CHURP. The experimental results show a quadratic increase consistent with the $O(n^2)$ asymptotic computational complexity of Opt-CHURP and suggests a low constant, e.g., for a committee of size 1001 the total protocol execution time is only about 3 minutes (Fig. 8b). As noted before, this does not include the additional latency for on-chain writes. Note that Opt-CHURP involves only 1 on-chain write per node which happens at the end of Opt-Proactivize, and in Ethereum currently each write takes about 15 seconds. Fig. 8b also shows that among the three phases, Opt-ShareDist dominates the execution time due to the relatively expensive $O(n)$ calls to KZG's CreateWitness per node. (CreateWitness involves $O(n)$ group element exponentiation, thus a total of $O(n^2)$ computation.)

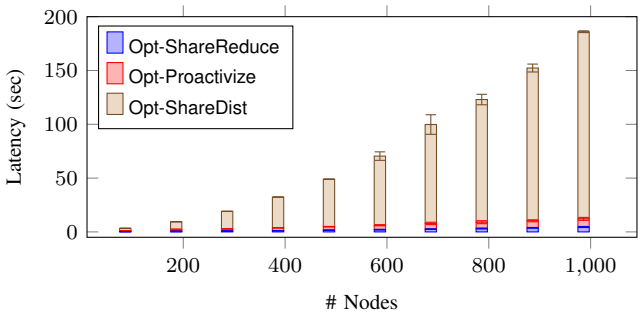
In the second set of experiments, we select EC2 instances across multiple regions in US, Canada, Asia and Europe, also referred to as the WAN setting. In this setting the network latency is relatively unstable, although even in the worst-case it is still sub-second. Hence, during a handoff of Opt-CHURP in the WAN setting, we expect a constant increase in the latency over the LAN setting. Moreover, we expect this constant to be relatively small compared to the time spent in computation. We validate our hypothesis—for a committee size of 100, the WAN latency is 4.54 seconds while the LAN latency is 2.92 seconds (Fig. 8a), i.e., the additional time spent in network latency is around 1.6 sec and constant across different committee sizes as expected. Note that we were unable to execute experiments in the WAN setting for committee sizes beyond 100 due to scaling limitations in the Amazon infrastructure. (We plan to get around this soon.)

2) On-chain communication complexity: Opt-CHURP incurs a linear on-chain communication complexity—n hashes, i.e. $32n$ bytes, are written to the blockchain in each handoff.

3) Off-chain communication complexity: Fig. 9 compares the off-chain complexity for different committee sizes for



(a) Opt-CHURP latency for the LAN & WAN setting with committee sizes 11-101, in increments of 10. The left bar is LAN latency and the right bar is WAN latency.



(b) Opt-CHURP latency for the LAN setting with committee size 101-1001, in increments of 100.

Fig. 8: Latency

Opt-CHURP and [17], a discussion about the comparison is in Section VI-C. Now, we discuss the off-chain costs of Opt-CHURP. The concrete performance numbers are consistent with the expected $O(n^2)$ asymptotic complexity.

The off-chain data transmitted per node includes: $2n$ (polynomial point, witness) pairs in the share reduction and the share distribution phase, and n elements of $\mathbb{F}_p$ in the proactivization phase; each node also sends 1 commitment to share, 3 commitments to polynomials, and 1 witness. With aforementioned parameters, a commitment to a t-degree polynomial is of size $65 \times t$ bytes (with compression) and points on polynomial are of of size 32B. For example, for $t = 50$ and $n = 101$, the off-chain complexity of Opt-CHURP is about $226n^2 + 325n \approx 2.3$MB. In Fig. 9, the expected curve is slightly below the observed data points because of trivial header messages unaccounted in the above calculations.

As we'll show now, the above is about 2300x lower than the communication complexity of the state of the art.

C. Comparison with Schultz's MPSS

The Mobile Proactive Secret Sharing (MPSS) protocol of Schultz et al. [17], referred to as Schultz-MPSS hereafter, achieves the similar goal as CHURP in asynchronous settings, assuming $t < n/3$. Compared to [17], Opt-CHURP achieves an $O(n^2)$ improvement for off-chain communication complexity. To evaluate the concrete performance improvement, we also implemented the optimistic path of Schultz-MPSS (Section 5 of [17] and evaluated the communication complexity empirically.

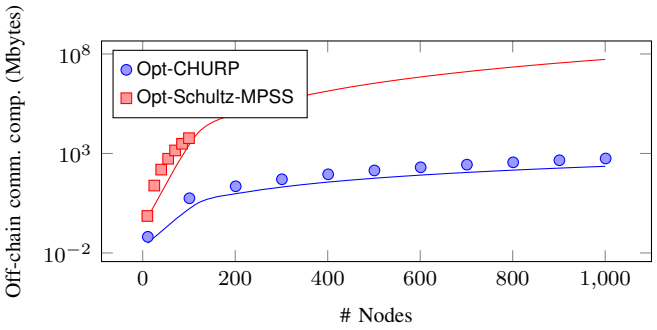


Fig. 9: Off-Chain communication complexity for Opt-CHURP and Schultz-MPSS, with log-scale y-axis. Points show experimental results; expected polynomial curves (respectively quadratic and quartic) are also shown.

1) Asymptotic improvement: Schultz-MPSS extends the usage of expensive blinding polynomials introduced by Herzberg et al. [5] to enable a dynamic committee membership. We recall briefly the asymptotic complexity of Schultz-MPSS and refer readers to [17] for details. Each node in the old committee generates a proposal of size $O(n^2)$ and send it to other nodes, resulting in an $O(n^4)$ off-chain communication complexity in total. Each node then validates the proposals and reaches consensus on the set of proposals to use by sending $O(n)$ accusations to the primary, incurring a $O(n^2)$ on-chain communication complexity. In the optimistic case where no accusation is sent—labelled Opt-Schultz-MPSS—the consensus publishes $O(n)$ hashes of proposals on chain and thus only incurs $O(n)$ on-chain communication complexity.

Table II compares the asymptotic communication complexity of Schultz-MPSS and CHURP. Schultz-MPSS has the same on-chain complexity as CHURP, but is $O(n^2)$ more expensive for off-chain communication.

2) Performance evaluation: We implemented the optimistic path of Schultz-MPSS (Section 5 of [17]) in about $3,100$ lines of Go code. To adapt Schultz-MPSS to the blockchain setting, we replace the BFT component of Schultz-MPSS with a trusted node. Fig. 9 compares the off-chain communication complexity of Opt-Schultz-MPSS and Opt-CHURP.

For practical parameterizations, our experiments show that Opt-CHURP can incur *orders of magnitude* less (off-chain) communication complexity than Opt-Schultz-MPSS. For example, for a committee of size 100, the off-chain communication complexity of Schultz-MPSS is $53.667n^4 \approx 5.3$GB, whereas that for Opt-CHURP is only 2.3MB, a 2300x improvement! Since Schultz-MPSS incurs excessive (Gigabytes) off-chain cost, we do not run it for committee sizes beyond 100.

VII. Point-to-Point Communication Technique

CHURP takes advantage of a hybrid on-chain / off-chain communication model to minimize communication costs. A blockchain is used to reach consensus on a total ordering of messages, while much cheaper and faster off-chain P2P communication transmits messages with no ordering requirement.

Off-chain P2P channels can be implemented in different ways depending on the deployment environment. However, in a decentralized setting, establishing *direct* off-chain connection between nodes is undesirable, as it would compromise nodes' anonymity. Revealing network-layer identities (e.g., IP addresses) would also be dangerous, as it could lead to targeted attacks. One can instead use anonymizing overlay networks, such as Tor—but at the cost of considerable additional setup cost and engineering complexity.

Alternatively, off-chain channels can be implemented as an overlay on existing blockchain infrastructure. In this section, we present *Transaction Ghosting*, a technique for *cheap* P2P messaging on a blockchain. The key trick to reduce cost is to *overwrite transactions* so that they are broadcast, but subsequently dropped by the network. Most of these transactions—and their embedded messages—are then essentially broadcast for free. We focus on Ethereum, but similar techniques can apply to other blockchains, e.g., Bitcoin.

A. Transaction Ghosting

A (simplified) Ethereum transaction $\mathsf{tx} = (n, m, g)$ includes a nonce n, payload m, and a per-byte *gas price* g paid to the miner of tx. For a basic ("send") transaction, Alice pays a miner $f_0 + |m| \times g$, where f_0 is a base transaction cost and $|m|$ is the payload size. (We make this more precise below.)

Alice sends tx to network peers, who add tx to their pool of unconfirmed transactions, known as the *mempool* [34]. They propagate tx so that it can be included ultimately in all peers' view of the mempool. tx remains in the mempool until a miner includes it in a block, at which point it is removed and $f_0 + |m| \times g$ units of currency is transferred from Alice to the miner.

The key observation is, until tx is mined, Alice can overwrite it with another transaction tx'. When this happens, tx is dropped from the mempool. Thus, both tx and tx' are propagated to all nodes, but Alice only pays for tx'.

Two additional techniques can further reduce costs. Alice can embed m in tx only, putting no message data in tx'. She then only pays *nothing* for the data containing m, only the cost associated with tx'. Additionally, this technique generalizes to multiple overwrites, i.e., Alice can embed a large message m in multiple transactions $\{\mathsf{tx}_i\}_{i\in[k-1]}$, which is useful given bounds (e.g., 32kB in Ethereum) on transaction sizes. Alice will still pay only the cost of the final transaction tx_k.

B. Choosing overwrite rate k

An optimal strategy is to make k as high as possible, i.e., overwrite many times. Ethereum, though, imposes a constraint on overwriting: the sender must raise the transaction fee in a fresh transaction by at least a minimum fraction ρ. (In Ethereum clients, ρ ranges from 10% to 12.5%).

Here we determine the optimal value of k. Recall that the fee for a transaction with $|m|$ bytes of data is $f = f_0 + g \times |m|$, for constants f_0 and g. Overwriting transactions with a fractional fee increase of ρ results in an average per-byte fee of $\frac{f\times\rho^k}{(1+k)\times|m|}$ for k overwritings, assuming the kth transaction gets mined. For $\rho = 12.5\%$, the optimal strategy is to write or overwrite $k = 7$ times, yielding average cost $0.29 \times \frac{f}{|m|}$ per byte, about 70% less than without overwriting. Moreover, if we send the first $k-1$ transactions with $|m|$ bytes of data and the last one empty, the average cost is driven down to $\frac{f_0\times\rho^k}{|m|\times k}$ per byte (because one only pays for the last empty transaction).

The above analysis assumes the kth transaction can always successfully overwrite previous ones, which happens in our experiments for two reasons. First, the kth transaction is smaller and higher-priced, thus preferred by miners; second, previous transactions usually remain pending for a long time (tens of minutes or longer), always allowing enough time for the kth to fully propagate.

C. Experiments

	On-chain	Transaction Ghosting
Bandwidth (KB/sec)	≤ 6.4	32.3 (9.31)
Latency (sec)	varies (Fig. 10)	1.09 (0.82)
Message transmission cost (USD/MB)	varies (Fig. 10)	\$0.06 (\$0.02)
Transaction delivery rate	100%	92.2% (14.2%)

TABLE III: Comparison between communication via the Ethereum blockchain and via Transaction Ghosting. Numbers in parentheses are standard deviations. The cost for Transaction Ghosting is based on an initial gas price of 1GWei. See Section VII-C for details.

We validate our ideas experimentally on the Ethereum blockchain (mainnet). The sender and receiver are full nodes connected to the Ethereum P2P network—with no out-of-band channel. The goal is for the sender to transmit messages to the receiver by embedding them in pending transactions. To overwrite a pending transaction in Ethereum, the sender reuses the same nonce and raises the gas price.

We set $k = 7$ in the experiment. Each of the first 7 transactions contains 31KB of data and the 8th is empty. A total of approximately 100MB data is successfully transmitted in $4,200$ transactions, in about 1 hour. Table III summarizes the results of our experiments, which we now discuss.

a) Bandwidth: Experimentally, we can propagate overwritten transactions at a rate of just under once a second, yielding approximate bandwidth 32.3KB/s, as the maximum permitted per-transaction data is 32KB [35]. While this suffices for CHURP, we believe more engineering would yield higher bandwidth. Studies of blockchain arbitrage [36] show that arbitrageurs can overwrite transactions in hundreds of milliseconds.

We emphasize that the shown bandwidth is *per channel*. One can establish N concurrent channels by overwriting N transactions simultaneously.

b) Message-transmission cost: Transaction costs for message delivery in Transaction Ghosting are extremely low: \$0.06 per megabyte on average, with gas price 1 GWei. The gas price should be chosen minimum required to get transactions relayed by peer nodes. Empirically of late, a gas price between 1 to 2 GWei offers good delivery rate, which we now explain.

c) Transaction delivery rate: Although a sender can make sure overwriting succeeds in her mempool, overwritten transactions are not guaranteed to arrive on the receiver's side. Possible reasons are an overloaded mempool [34], network congestion and/or out-of-order delivery. Generally transactions with a higher transaction fee are relayed preferentially by peer nodes, and less frequently dropped. The 8th transaction in our rewriting sequence has the highest fee and the smallest payload, and is always delivered in our experiments.

Overall, we observe an average transaction delivery rate of 91.9% in our experiments, or a $\approx 9\%$ loss rate. Our Transaction Ghosting is thus an erasure channel. A sender can either erasure-code m to ensure full delivery without interaction with the receiver, or use a standard network retransmission protocol so the receiver can signal a delivery failure. These techniques are out of scope for our exploration here.

D. Comparison to on-chain communication

For comparison, we estimate the same metrics for on-chain communication, i.e. using the Ethereum blockchain as a message carrier. The results are summarized in Table III.

An upper bound on the on-chain bandwidth is estimated assuming a 8 million block gas limit. Each block can hold at most three 32KB transactions, thus a total of 96KB data every 15 seconds, or 6.4 KB/s.

The message transmission cost per megabyte is estimated as that of sending 32 transactions with 32KB data in each, assuming an exchange rate of 1ETH = \$200. The latency, i.e., the time between a transaction first appears in the mempool and the time it is mined, depends on the gas price and the network condition. A lower latency requires a higher gas price and thus a higher transmission cost. Several services such as [37], [38] collect metrics for gas price vs. latency tradeoff. We used [37] for our estimation. The tradeoff between latency and message transmission cost is shown in Fig. 10.

Notes: While our techniques may seem an abuse of the Ethereum P2P network, the idea of leveraging the network for alternative forms of communication has been under consideration by the community for some time; see, e.g., [39].

At the time of writing, gas prices in Ethereum have been consistently low for a period of approximately two months [40], preventing experimentation in a high-gas-price regime. We believe, however, that the same techniques would still work in such settings—with higher overall cost.

VIII. Related Work

Verifiable Secret Sharing (VSS): Polynomial-based secret sharing was introduced by Shamir [4]. Feldman [20] and Pedersen [21] proposed an extension called *verifiable secret sharing (VSS)*, in which dealt shares' correctness can be verified against a commitment of the underlying polynomial. In these schemes, a commitment to a degree-t polynomial has size $O(t)$. The polynomial-commitment scheme of Kate et al. [22] (KZG) reduces this to $O(1)$, and is adopted for secret sharing in, e.g., [23], and in CHURP.

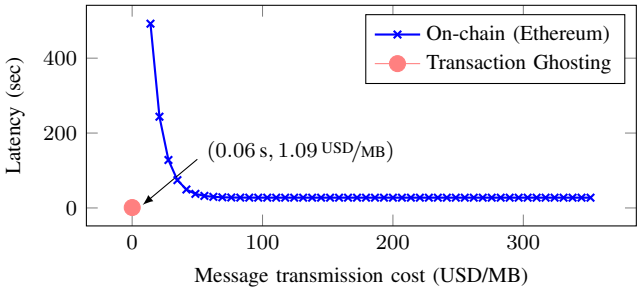


Fig. 10: Tradeoff in latency vs. message transmission cost. The blue curve shows the observed on-chain tradeoff. The red dot at $(0.06\,\text{s}, 1.09\,^{\text{USD}}/_{\text{MB}})$ corresponds to Transaction Ghosting.

Protocol	Dynamic	Adversary	Network	Threshold	Cost
Herzberg et al. [5]	No	active	synch.	$t < n/2$	$O(n^2)$
Cachin et al. [11]	No	active	asynch.	$t < n/3$	$O(n^4)$
Desmedt et al. [48]	Yes	passive	synch.	$t < n/2$	$O(n^2)$
Wong et al. [15]	Yes	active	synch.	$t < n/2$	$\exp(n)$
Zhou et al. [16]	Yes	active	asynch.	$t < n/3$	$\exp(n)$
Schultz-MPSS [17]	Yes	active	asynch.	$t < n/3$	$O(n^4)$
CHURP	Yes	active	synch.	$t < n/2$	$O(n^3)$

TABLE IV: Comparison of Proactive Secret Sharing (PSS) schemes—those above the line do not handle dynamic committees while the ones below do so. Cost indicates the off-chain communication complexity.

Proactive Secret Sharing (PSS): Proactive security, the idea of refreshing secrets to withstand compromise, was first proposed by Ostrovsky and Yung [41] for multi-party computation (MPC). It was first adapted for secret sharing by Herzberg et al. [5], whose techniques continue to be used in subsequent works, e.g., [42], [43], [44], [45], [17], [46], [47], and in CHURP (in UnivariateZeroShare). As noted, a result of independent interest in our work is an $O(n)$ reduction in the off-chain communication complexity of [5]. (See Fig. 17.)

All the above schemes assume a synchronous network model and computationally bounded adversary; CHURP does too, given its blockchain setting. PSS schemes have also been proposed in asynchronous settings [11], [16], [17] and unconditional settings [18], [19]. Nikov and Nikova [28] provide a survey of the different techniques used in PSS schemes along with some attacks (which CHURP addresses via its novel dimension-switching techniques).

Dynamic committee membership: Desmedt and Jajodia [48] propose a scheme that can change the committee and threshold in a secret-sharing system, but is unfortunately not verifiable. Wong et al. [15] build a verifiable scheme assuming that the nodes in the new committee are non-faulty. Subsequent works by Zhou et al. [16] and Schultz et al. [17] were the *first* to allow for dynamic committee membership with an adversarial model similar to ours. While [16] incurs exponential communication cost, [17] improves it to $O(n^4)$ off-chain cost. In contrast, as Table IV shows, CHURP achieves worst-case $O(n^3)$ cost.

Baron et al. [14], [49] recently proposed dynamic-committee schemes with $O(1)$ communication complexity per secret. Their scheme has impractically high constants, though, and assumes amortization over large numbers of

secrets (specifically for MPC), while we want efficiency even for small numbers of secrets.

Bivariate polynomials: Bivariate polynomials have been explored extensively in the secret-sharing literature, to build VSS protocols [50], [11], [51], for secret-sharing with unconditional security [18], [19], [47], etc. Few works [12], [13], however, have considered application to dynamic committees for secret sharing, and these have been limited to passive adversaries. CHURP's novel use of *dimension-switching* provides security against active, adaptive adversaries.

0-sharing, the technique of generating a 0-hole polynomial has been widely used for proactive security since the work of [5]. As we explain before, prior works [12], [13], [19] have naively extended these to the bivariate case leading to expensive 0-sharing protocols. Instead, CHURP applies known share re-sharing techniques [52], [48] to build an efficient bivariate 0-sharing protocol.

REFERENCES

[1] "Decentralized identity foundation (DIF) homepage," https://identity.foundation/, 2018.

[2] J. J. Roberts and N. Rapp, "Exclusive: Nearly 4 Million Bitcoins Lost Forever, New Study Says," 11 2017. [Online]. Available: http://fortune.com/2017/11/25/lost-bitcoins/

[3] B. Armstrong, "Coinbase is not a wallet," https://blog.coinbase.com/coinbase-is-not-a-wallet-b5b9293ca0e7, Feb. 25, 2018.

[4] A. Shamir, "How to share a secret," *Communications of the ACM*, 1979.

[5] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *Annual International Cryptology Conference*, 1995.

[6] A. Asayag, G. Cohen, I. Grayevsky, M. Leshkowitz, O. Rottenstreich, R. Tamari, and D. Yakira, "Helix: a scalable and fair consensus algorithm," Technical report, Orbs Research, Tech. Rep., 2018.

[7] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution," *arXiv preprint arXiv:1804.05141*, 2018.

[8] M. Egorov, M. Wilkison, and D. Nuñez, "Nucypher kms: decentralized key management system," *arXiv preprint arXiv:1707.06140*, 2017.

[9] E. Kokoris-Kogias, E. C. Alp, S. D. Siby, N. Gailly, L. Gasser, P. Jovanovic, E. Syta, and B. Ford, "Calypso: Auditable sharing of private data over blockchains," Cryptology ePrint Archive, 2018.

[10] G. Zyskind, O. Nathan *et al.*, "Decentralizing privacy: Using blockchain to protect personal data," in *Security and Privacy Workshops*, 2015.

[11] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl, "Asynchronous verifiable secret sharing and proactive cryptosystems," in *9th ACM conference on Computer and communications security*, 2002.

[12] N. Saxena, G. Tsudik, and J. H. Yi, "Efficient node admission for short-lived mobile ad hoc networks," in *13th IEEE International Conference on Network Protocols*, 2005.

[13] S. Dolev, J. Garay, N. Gilboa, and V. Kolesnikov, "Swarming secrets," in *Allerton Conference on Communication, Control, and Computing*, 2009.

[14] J. Baron, K. El Defrawy, J. Lampkins, and R. Ostrovsky, "Communication-optimal proactive secret sharing for dynamic groups," in *International Conference on Applied Cryptography and Network Security*, 2015.

[15] T. M. Wong, C. Wang, and J. M. Wing, "Verifiable secret redistribution for archive systems," in *the first International Security in Storage Workshop*, 2002.

[16] L. Zhou, F. B. Schneider, and R. Van Renesse, "Apss: Proactive secret sharing in asynchronous systems," *ACM transactions on information and system security (TISSEC)*, 2005.

[17] D. A. Schultz, B. Liskov, and M. Liskov, "Mobile proactive secret sharing," in *ACM symposium on Principles of distributed computing*, 2008.

[18] D. R. Stinson and R. Wei, "Unconditionally secure proactive secret sharing scheme with combinatorial structures," in *International Workshop on Selected Areas in Cryptography*, 1999.

[19] M. Nojoumian, D. R. Stinson, and M. Grainger, "Unconditionally secure social secret sharing scheme," *IET information security*, 2010.

[20] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *Foundations of Computer Science, 1987., 28th Annual Symposium on*, 1987.

[21] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual International Cryptology Conference*, 1991.

[22] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2010.

[23] M. Backes, A. Kate, and A. Patra, "Computational verifiable secret sharing revisited," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2011.

[24] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *IEEE Symposium on Foundations of Computer Science*, 2001.

[25] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems*, 2002.

[26] P. Syverson, R. Dingledine, and N. Mathewson, "Tor: The secondgeneration onion router," in *Usenix Security*, 2004.

[27] Ethereum, "Devp2p." [Online]. Available: https://github.com/ethereum/devp2p

[28] V. Nikov and S. Nikova, "On proactive secret sharing schemes," in *International Workshop on Selected Areas in Cryptography*, 2004.

[29] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," in *International Conference on the Theory and Applications of Cryptographic Techniques*, 1999.

[30] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *the 26th Symposium on Operating Systems Principles*, 2017.

[31] "Go language interface to gmp - gnu multiprecision library (golang)," https://github.com/ncw/gmp, (Accessed on 11/14/2018).

[32] "The pbc go wrapper," https://github.com/Nik-U/pbc, (Accessed on 11/14/2018).

[33] "grpc: A high performance, open-source universal rpc framework," https://grpc.io/, (Accessed on 11/22/2018).

[34] Parity, "Transaction queue," https://wiki.parity.io/Transactions-Queue.

[35] geth, "The maximum data size in a transaction is 32 KB," https://github.com/ethereum/go-ethereum/blob/6a3395473165866705646bf7573ed1c397f4750/core/tx_pool.go#L570.

[36] frontrun.me, "Visualizing Ethereum gas auctions," http://frontrun.me/.

[37] "ETH gas station — consumer oriented metrics for the ethereum gas market," https://ethgasstation.info/, (Accessed on 11/13/2018).

[38] "Ethereum gas price tracker," https://etherscan.io/gastracker, (Accessed on 11/13/2018).

[39] Ethereum, "Whisper." [Online]. Available: https://github.com/ethereum/wiki/wiki/Whisper

[40] Etherscan, "Ethereum average gasprice chart," https://etherscan.io/chart/gasprice, [Online; accessed 2018].

[41] R. Ostrovsky and M. Yung, "How to withstand mobile virus attacks," in *ACM symposium on Principles of distributed computing*, 1991.

[42] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust threshold dss signatures," in *International Conference on the Theory and Applications of Cryptographic Techniques*, 1996.

[43] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive public key and signature systems," in *the 4th ACM conference on Computer and communications security*, 1997.

[44] R. Cramer, R. Gennaro, and B. Schoenmakers, "A secure and optimally efficient multi-authority election scheme," *European transactions on Telecommunications*, 1997.

[45] H. Luo, P. Zerfos, J. Kong, S. Lu, and L. Zhang, "Self-securing ad hoc wireless networks," in *International Symposium on Computers and Communications*, 2002.

[46] K. D. Bowers, A. Juels, and A. Oprea, "Hail: A high-availability and integrity layer for cloud storage," in *the 16th ACM conference on Computer and communications security*, 2009.

[47] M. Nojoumian and D. R. Stinson, "On dealer-free dynamic threshold schemes." *Adv. in Math. of Comm.*, 2013.

[48] Y. Desmedt and S. Jajodia, "Redistributing secret shares to new access structures and its applications," Technical Report ISSE TR-97-01, George Mason University, Tech. Rep., 1997.

[49] J. Baron, K. El Defrawy, J. Lampkins, and R. Ostrovsky, "How to withstand mobile virus attacks, revisited," in *the 2014 ACM symposium on Principles of distributed computing*, 2014.
[50] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *ACM symposium on Theory of computing*, 1988.
[51] P. Feldman and S. Micali, "An optimal probabilistic protocol for synchronous byzantine agreement," *SIAM Journal on Computing*, 1997.
[52] Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung, "Optimal-resilience proactive public-key cryptosystems," in *38th Annual Symposium on Foundations of Computer Science*, 1997.
[53] D. Chaum, C. Crépeau, and I. Damgard, "Multiparty unconditionally secure protocols," in *the twentieth annual ACM symposium on Theory of computing*, 1988.
[54] R. Cramer, I. Damgård, and U. Maurer, "General secure multi-party computation from any linear secret-sharing scheme," in *International Conference on the Theory and Applications of Cryptographic Techniques*, 2000.
[55] Y. Desmedt and Y. Frankel, "Shared generation of authenticators and signatures," in *Annual International Cryptology Conference*, 1991.
[56] M. O. Rabin, "Randomized byzantine generals," in *Foundations of Computer Science, 1983., 24th Annual Symposium on*, 1983.
[57] J. J. Wylie, M. W. Bigrigg, J. D. Strunk, G. R. Ganger, H. Kiliccote, and P. K. Khosla, "Survivable information storage systems," *In Computer*, 2000.
[58] J. P. Njui, "Coinbase custody service secures major institutional investor worth $20 billion," *Ethereum World News*, July 24, 2018. [Online]. Available: https://ethereumworldnews.com/coinbase-custody-service-secures-major-institutional-investor-worth-20-billion/
[59] Y. Akkawi, "Bitcoin's most pressing issue summarized in two letters: UX," *Inc.*, 21 Dec. 2017.
[60] "uport," https://www.uport.me/, 2018.
[61] Kames, "The basics of decentralized identity how blockchain technology & cryptographic primitives embolden the future of digital identity," https://medium.com/uport/the-basics-of-decentralized-identity-d1ff01f15df1, 26 June 2018.
[62] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *Ieee Access*, 2016.
[63] G. Prisco, "Slock.it to introduce smart locks linked to smart ethereum contracts, decentralize the sharing economy," *Bitcoin Magazine*, 2015.
[64] I. Bentov, R. Pass, and E. Shi, "Snow white: Provably secure proofs of stake." *IACR Cryptology ePrint Archive*, 2016.
[65] V. Buterin, "Slasher: A punitive proof-of-stake algorithm," *Ethereum Blog URL: https://blog. ethereum. org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm*, 2014.
[66] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," *self-published paper, August*, 2012.
[67] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*, 2017.
[68] J. Groth, "Short pairing-based non-interactive zero-knowledge arguments," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2010.

APPENDIX A
SECURITY PROOF FOR Opt − CHURP

The ideal functionality of Opt − CHURP is defined in Figure 11. The simulator for $\mathcal{F}_{\mathsf{Opt-CHURP}}$ is given in Figure 13.

To prove the secrecy of the scheme, one can see that in step 5 of the share reduction, as $|\mathcal{C}^{(e)}_{corrupted}| \leq t$, $B^{*(e)}(x, y)$ and $B^{(e)}(x, y)$ are both uniformly random. In step 6, the witnesses $w^{*(e)}_{i,j}$ are indistinguishable to those in the real world (i.e. step 7 in Figure 5). Similarly, in step 13 of share distribution, the witnesses $w^{*(e+1)}_{i,j}$ are indistinguishable to those generate in step 5 of Figure 7. Therefore, $\mathcal{A}$ cannot distinguish between the real world interacting with the protocol and the ideal world interacting with $\mathcal{S}$.

APPENDIX B
APPLICATIONS IN DECENTRALIZED SYSTEMS

Secret sharing finds use in innumerable applications involving cryptographic secrets, including secure multi-party computation (MPC) [50], [53], [54], threshold cryptography [55], Byzantine agreement [56], survivable storage systems [57], and cryptocurrency custody [3], [58], to name just a few.

Decentralized systems, however, are an especially attractive application domain, though, for two reasons.

First, blockchain systems *task individual users with management of their own private keys*, an unworkable approach for most users. A frequent result, as noted above, is key loss [2] or centralized key management [3], [58] that defeats the main purpose of blockchain systems.

Second, *blockchain objects cannot keep private state*. This fact notably limits the useful applications of smart contracts, as they cannot compute digital signatures or manage encrypted data.

We briefly enumerate a few of the most important potential applications in decentralized systems of the (dynamic-committee proactive) secret-sharing enabled by CHURP:

a) Usable cryptocurrency management: Rather than relying on centralized parties (e.g., exchanges) to custody private keys for cryptocurrency, or using hardware or software wallets, which are notoriously difficult to manage [59], users could instead store their private keys with committees. These committees could authenticate users and enforce access-control, resulting in the decentralized equivalent of today's exchanges.

b) Decentralized identity: Initiatives such as the Decentralized Identity Foundation [1], which is backed by a number of major IT and services firms, as well as smaller efforts, such as uPort [60], envision an ecosystem in which users control their identities and data by means of private keys. Who will store these keys and how is left an open question [61]. The same techniques used in the cryptocurrency case for private-key management would similarly apply to assets such as identities. Additionally, a committee could manage *encrypted* identity documents on users' behalf.

c) Auditable access control: As proposed in [9], a committee could manage encrypted documents and decrypt them for recipients under a given access-control policy while logging their accesses on-chain. The result would be a strongly

$\mathcal{F}_{\mathsf{Opt-CHURP}}$

1 : **Input:** Shares of the old committee $B^{(e)}(i, j)$ for $i \in \mathcal{C}^{(e)}_{alive}$ and $j \in [2t+1]$.
2 : **Corruption:** $\mathcal{A}$ requests to corrupt parties in $\mathcal{C}^{(e)}$ and $\mathcal{C}^{(e+1)}$. For each party i in $\mathcal{C}^{(e)}$ $\mathcal{A}$ corrupts, send $B^{(e)}(i, j)$ for $j \in [2t + 1]$.
3 : **Computation:**
4 : Reconstruct degree $\langle t, 2t\rangle$ polynomial $B^{(e)}(x, y)$ through polynomial interpolation.
5 : Generate random polynomials $Q(x, y)$ such that $Q(0, 0) = 0$.
6 : Set $B^{(e+1)}(x, y) = B^{(e)}(x, y) + Q(x, y)$.
7 : **Output:** Shares of the new committee $B^{(e+1)}(i, j)$ for $i \in \mathcal{C}^{(e+1)}$ and $j \in [2t + 1]$.

Fig. 11: Ideal functionality $\mathcal{F}_{\mathsf{Opt-CHURP}}$

Kate's Polynomial Commitment

1 : $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Keygen}(1^\lambda, q)$: Select a bilinear group $(p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathsf{BilGen}(1^\lambda)$ and s randomly in $\mathbb{Z}_p^*$. Set $\mathsf{sk} = s$ and $\mathsf{pk} = g^s, g^{s^2}, \ldots, g^{s^q}$.

2 : $C_\phi \leftarrow \mathsf{Commit}(\phi(x), \mathsf{pk})$: Compute $C_\phi = g^{\phi(s)}$ using pk.

3 : $(\phi(i), W_i) \leftarrow \mathsf{CreateWitness}(\phi(x), i, \mathsf{pk})$: Compute $\phi(x) - \phi(i) = (x - i)w(x)$, set $W_i = g^{w(s)}$.

4 : $\{\mathsf{True}, \mathsf{False}\} \leftarrow \mathsf{VerifyEval}(C_\phi, i, \phi(i), W_i, \mathsf{pk})$: Output True if $e(C_\phi / g^{\phi(i)}, g) = e(g^{s-i}, W_i)$. Otherwise, output False.

Fig. 12: Protocols of Kate's polynomial commitment.

auditable access-control system. This application could be managed by a smart contract.

d) Smart-contract attestations: Committee management of smart-contract private keys could also enable *digital signing* by smart contracts. The idea would be that committee members execute threshold signatures using a shared private key, emitting a signature for a particular smart contract in response to a request issued by the contract on chain.

Such signing would be of particular benefit in creating a simple smart-contract interface with *off-chain* systems. For example, control of Internet-of-Things (IoT) devices is commonly proposed application of smart contracts [62] (smart locks being a notable early example [63]). If smart contracts cannot generate digital signatures, then the devices they control must monitor a blockchain, an ongoing resource-intensive operation. A smart contract that can generate a digital signature, however, can simply issue authenticable commands to target devices.

e) Simplified Committee-based consensus for light clients: A number of consensus schemes, e.g., proof-of-stake protocols [64], [65], [66], [67], aim to achieve good scalability by delegating consensus to committees. These committees change over time. Therefore verifying the blocks they sign requires awareness of their identities. By instead maintaining or only periodically rotating its public / private key pair, a committee could instead make it easier for light clients to verify signed blockchains.

f) Secure multiparty computation (MPC) for smart contracts: More generally, dynamic-committee secret sharing would enable decentralized secure multiparty computation (MPC) by smart contracts, effectively endowing them with confidential storage and computation functionalities, as envisioned in, e.g., [10], [7].

Appendix C
The Static Setting: Improved PSS

We also consider a different and narrower setting, one with a *static* committee i.e., the old and new committees are identical. The adversarial model is also weaker i.e., corruptions during the handoff phase are counted towards the threshold in both the adjacent epochs. The handoff in such a setting is simply an *update* since the committee is static. Hence, the update protocol consists of a *recovery* phase, enabling recovery of lost shares and a *refresh* phase, updating shares of all nodes.

In this section, we look at different techniques seen in literature for the static setting. Herzberg et al. [5] introduce this setting and present a protocol, Herzberg's PSS. A second technique seen in the literature makes use of bivariate polynomials. We then present an improved PSS protocol which achieves better overall performance than any known scheme.

		Univariate [5]	Bivariate	Improved PSS
Off-chain	Recovery	$O(n^2)$	$O(n)$	$O(n)$
	Refresh	$O(n^2)$	$O(n^3)$	$O(n^2)$
State		$O(1)$	$O(n)$	$O(n)$

TABLE V: Comparison of protocols in the static setting with a honest-but-curious adversary. The original protocol of Herzberg et al. is presented in the univariate column. Recovery costs are per node. Note that recovery costs of our protocol are amortized over the total number of nodes being replaced.

Herzberg's PSS: This protocol incurs $O(n^2)$ off-chain communication complexity for refresh and an expensive $O(n^2)$ per node recovery (See [5]).

Bivariate Polynomials: One way to avoid the expensive recovery cost is to perform secret sharing with a bivariate polynomial. This allows for efficient recovery, i.e., $O(n)$ off-chain communication complexity. As discussed previously in Section IV, existing techniques for refresh are expensive costing $O(n^3)$.

Improved PSS: Much like the dynamic setting, we build an improved PSS protocol using the efficient bivariate 0-sharing technique. This technique brings down the total communication complexity to just $O(n^2)$ off-chain. A comparison of communication costs incurred by different PSS schemes is in Table V.

Let $\mathcal{C}$ denote the committee, $\mathcal{C} = \mathcal{C}^{(e-1)} = \mathcal{C}^{(e)}$, comprising n nodes $\{\mathcal{C}_i\}_{i=1}^n$. The secret is shared using an asymmetric bivariate polynomial $B(x, y)$, $s = B(0, 0)$. Unlike before, the degree of bivariate polynomial is only $\langle t, t \rangle$ as we have a weaker adversary.

Recall that node's share is a single polynomial $B(i, y)$. In Fig. 17, we present the improved PSS assuming a honest-but-curious adversary. Throughout the protocol, each node sends out atmost $O(n)$ points. Thus, our improved PSS scheme completes in $O(n^2)$ off-chain cost.

Active adversaries: In face of adversarial behaviour, multiple reruns of the protocol might be needed. This is crucial since all the $t + 1$ received points need to be correct in order to compute the new share. Adversaries are detectable with the use of KZG commitments similar to the dynamic setting. We replace the detected adversarial nodes with uncorrupted nodes from $\mathcal{C}$ (guaranteed to find such a node, $|\mathcal{C}| \geq 2t + 1$). We stress that this protocol incurs $O(n^2)$ off-chain cost even after adapting to handle active adversaries. This is achieved due to the following key property: Honest nodes never rerun any phase of the protocol. This is possible by making a slight modification to the univariate 0-sharing (step 9): invoke (t, n)-UnivariateZeroShare among all nodes in $\mathcal{C}$ instead of executing it in a subset of nodes only. Observe that the set of univariate

Simulator $\mathcal{S}$ for $\mathcal{F}_{\mathsf{Opt-CHURP}}$

Input: Trapdoor of the KZG scheme $\mathsf{sk} = s$. Public key pk. Commitments of the polynomials $C_{B^{(e)}(x,j)}$ for $j \in [2t+1]$.

An adversary $\mathcal{A}$ sends requests to corrupt parties in $\mathcal{C}^{(e)}$ to $\mathcal{S}$. $\mathcal{S}$ forwards the requests to $\mathcal{F}_{\mathsf{Opt-CHURP}}$, and sends shares of these corrupted parties from $\mathcal{F}_{\mathsf{Opt-CHURP}}$ back to $\mathcal{A}$.

Share reduction:

$\mathcal{A}$ sends shares and witness for corrupted parties $B^{(e)}(i,j)$ and $w_{i,j}^{(e)}$ for $j \in [2t+1]$ and $i \in \mathcal{C}_{corrupted}^{(e)}$ to $\mathcal{S}$.

$\mathcal{S}$ chooses a random polynomial $B^{*(e)}(x,y)$ such that $B^{*(e)}(i,j) = B^{(e)}(i,j)$ for $j \in [2t+1]$ and $i \in \mathcal{C}_{corrupted}^{(e)}$.

$\mathcal{S}$ computes $B^{*(e)}(i,j)$ and forges witnesses $w_{i,j}^{*(e)}$ using the trapdoor s for $i \in \mathcal{C}_{alive}^{(e)} \setminus \mathcal{C}_{corrupted}^{(e)}$. In particular, $w_{i,j}^{*(e)} = (C_{B^{(e)}(x,j)} / g^{B^{*(e)}(i,j)})^{(s-i)^{-1}}$. $\mathcal{S}$ sends these shares and witnesses to $\mathcal{A}$.

Verify the shares $\mathsf{VerifyEval}(\mathsf{C}(B^{(e)}(x,j)), j, B^{(e)}(i,j), w_{i,j}^{(e)})$ for $i \in \mathcal{C}_{corrupted}^{(e)}$ and $j \in [2t+1]$. If any of these output False, output abort.

$\mathcal{S}$ sends $B^{(e)}(i,j)$ for $j \in [2t+1]$ and $i \in \mathcal{C}_{corrupted}^{(e)}$ to $\mathcal{F}_{\mathsf{Opt-CHURP}}$. $\mathcal{S}$ receives $B^{(e+1)}(i,j)$ for $i \in \mathcal{C}_{corrupted}^{(e+1)}$ and $j \in [2t+1]$.

Proactivization:

$\mathcal{S}$ emulates the honest parties in $\mathcal{U}^{(e+1)}$ to perform a bivariate 0-sharing protocol with $\mathcal{A}$. (See step 6-13 in Figure 6.) If the protocol fails, output abort. Otherwise, by the end of the protocol, $\mathcal{S}$ obtains polynomials $Q(x,j)$ for $j \in \mathcal{U}^{(e+1)} \setminus \mathcal{U}_{corrupted}^{(e+1)}$.

$\mathcal{S}$ computes $B^{*(e+1)}(x,j) = B^{*(e)}(x,j) + Q(x,j)$ for $j \in \mathcal{U}^{(e+1)} \setminus \mathcal{U}_{corrupted}^{(e+1)}$.

Share distribution:

$\mathcal{S}$ forges witnesses $w_{i,j}^{*(e+1)}$ using the trapdoor s for $i \in \mathcal{C}^{(e+1)} \setminus \mathcal{C}_{corrupted}^{(e+1)}$. In particular, $w_{i,j}^{*(e+1)} = (C_{B^{(e+1)}(x,j)} / g^{B^{(e+1)}(i,j)})^{(s-i)^{-1}}$, where $C_{B^{(e+1)}(x,j)} = C_{B^{(e)}(x,j)} \cdot C_{Q(x,j)}$. $\mathcal{S}$ sends witnesses and $B^{(e+1)}(i,j)$ received from $\mathcal{F}_{\mathsf{Opt-CHURP}}$ for $i \in \mathcal{C}_{corrupted}^{(e+1)}$ and $j \in [2t+1]$ to $\mathcal{A}$.

Fig. 13: Simulator $\mathcal{S}$ for $\mathcal{F}_{\mathsf{Opt-CHURP}}$

$(2t, 2t+1)$-UnivariateZeroShare

Input: t, set of $2t+1$ nodes $\{\mathcal{U}_j\}_{j\in[2t+1]}$

Output: Each node $\mathcal{U}_j$ outputs a share $s_j = P(j)$ for randomly generated degree-$2t$ polynomial $P(y)$ with $P(0) = 0$

node $\mathcal{U}_j$

Generate a random $2t$-degree polynomial P_j s.t. $P_j(0) = 0$

Send a point $P_j(i)$ to node $\mathcal{U}_i$ for each $i \in [2t+1]$

Wait to receive points $\{P_i(j)\}_{i\in[2t+1]}$ from all other nodes

Let $P = \sum_{i\in[2t+1]} P_i$, compute share $P(j) = \sum_{i\in[2t+1]} P_i(j)$

Fig. 14: $(2t, 2t+1)$-UnivariateZeroShare between $2t+1$ nodes. A 0-hole univariate polynomial P of degree-$2t$ is generated.

(t, n)-BivariateZeroShare

Input: t, n, set of nodes $\{\mathcal{C}_i\}_{i\in[n]}$ $(2t < n)$

Output: Each node $\mathcal{C}_i$ outputs a share $Q(i,y)$ for randomly generated degree-$\langle t, 2t\rangle$ bivariate polynomial $Q(x,y)$ with $Q(0,0) = 0$

Order $\{\mathcal{C}_i\}_{i\in[n]}$ based on lexicographic order of their public keys

Choose first $2t+1$ nodes, w.l.o.g., $\mathcal{U} = \{\mathcal{C}_j\}_{j\in[2t+1]}$

Invoke $(2t, 2t+1)$-UnivariateZeroShare among $\{\mathcal{U}_j\}_{j\in[2t+1]}$ to generate shares $\{s_j\}_{j\in[2t+1]}$

node $\mathcal{U}_j$:

Generate a random t-degree polynomial R_j s.t $R_j(0) = s_j$

Send a point $R_j(i)$ to node $\mathcal{C}_i$ for each $i \in [n]$

Denote the bivariate polynomial $Q(x,y)$ where $\{Q(x,j) = R_j(x)\}_{j\in[2t+1]}$

node $\mathcal{C}_i$:

Wait to receive points $\{R_j(i)\}_{j\in[2t+1]} = \{Q(i,j)\}_{j\in[2t+1]}$

Interpolate to reconstruct a $2t$-degree polynomial $Q(i,y)$

Output share $Q(i,y)$

Fig. 15: (t,n)-BivariateZeroShare between n nodes. A 0-hole bivariate polynomial Q of degree-$\langle t, 2t\rangle$ is generated.

Improved-PSS

Input: Set of n nodes $\mathcal{C}$. Each node $\mathcal{C}_i'$ is given a degree-t polynomial $B(i,y)$

Output: $\mathcal{C}_i'$ outputs $B'(i,y)$ for a degree-$\langle t,t\rangle$ bivariate polynomial $B'(x,y)$ with $B'(0,0) = B(0,0)$

Order nodes in $\mathcal{C}$ based on the lexicographic ordering determined by public keys

Choose first $t+1$ nodes, $\mathcal{U} \subset \mathcal{C}$, $|\mathcal{U}| = t+1$

node $\mathcal{C}_i$:

send $B(i,j)$ to node $\mathcal{U}_j$, $\forall j \in [t+1]$

node $\mathcal{U}_j$:

Reconstruct degree-t polynomial $B(x,j)$

Invoke $(t, t+1)$-UnivariateZeroShare among $\mathcal{U}$ generating shares $\{s_j\}_j$, $\forall j \in [t+1]$

node $\mathcal{U}_j$:

Generate a degree-t polynomial $Q(x,j)$ s.t. $Q(0,j) = s_j$

Update the reconstructed polynomial $B'(x,j) = B(x,j) + Q(x,j)$

send $B'(i,j)$ to each node $i \in \mathcal{C}$

node $\mathcal{C}_i$:

Construct degree-t polynomial $B'(i,y)$ using $t+1$ received points

Fig. 16: Improved PSS for static setting, honest-but-curious adversary.

polynomials held by any $t+1$-sized subset in $\mathcal{C}$ defines a 0-hole bivariate polynomial. Thus, reruns are executed only by the new uncorrupt nodes that replace the detected faulty nodes.

APPENDIX D
CHURP PESSIMISTIC PATHS

In this section, we present protocols for the two pessimistic paths of CHURP: Exp-CHURP-A and Exp-CHURP-B.

A. Exp-CHURP-A

This path is invoked when a failure occurs in Opt-CHURP. It also consists of three phases: Exp-ShareReduce, Exp-Proactivize, Exp-ShareDist.

Before the first phase starts, commitments to reduced shares $\{B(x,j)\}_{j=1}^{2t+1}$ are published on-chain by $t+1$ nodes in the

Improved-PSS

1 : **Input:** Set of n nodes $\mathcal{C}$. Each node $\mathcal{C}'_i$ is given a degree-t polynomial $B(i, y)$
2 : **Output:** $\mathcal{C}'_i$ outputs $B'(i, y)$ for a degree-$\langle t, t\rangle$ bivariate polynomial $B'(x, y)$ with $B'(0, 0) = B(0, 0)$
3 : Order nodes in $\mathcal{C}$ based on the lexicographic ordering determined by public keys
4 : Choose first $t + 1$ nodes, $\mathcal{U} \subset \mathcal{C}$, $|\mathcal{U}| = t + 1$
5 : node $\mathcal{C}_i$:
6 : send $B(i, j)$ to node $\mathcal{U}_j$, $\forall j \in [t + 1]$
7 : node $\mathcal{U}_j$:
8 : Reconstruct degree-t polynomial $B(x, j)$
9 : Invoke $(t, t + 1)$-UnivariateZeroShare among $\mathcal{U}$ generating shares $\{s_j\}_j$, $\forall j \in [t + 1]$
10 : node $\mathcal{U}_j$:
11 : Generate a degree-t polynomial $Q(x, j)$ s.t. $Q(0, j) = s_j$
12 : Update the reconstructed polynomial $B'(x, j) = B(x, j) + Q(x, j)$
13 : send $B'(i, j)$ to each node $i \in \mathcal{C}$
14 : node $\mathcal{C}_i$:
15 : Construct degree-t polynomial $B'(i, y)$ using $t + 1$ received points

Fig. 17: Improved PSS for static setting, honest-but-curious adversary.

old committee. The on-chain hashes can be used to verify the posted commitments. As atleast one of the $t + 1$ nodes is honest, and thus each member of the new committee has the commitments.

*1) Share Reduction (*Exp-ShareReduce*):* This phase is the same as Opt-ShareReduce, and is not re-executed if Opt-ShareReduce ends successfully. This is because the degree of $B(x, j)$ is t and an honest member $\mathcal{U}'_j$ can successfully reconstruct the polynomial given $t + 1$ honest values from $\mathcal{C}_{alive}$ assuming $|\mathcal{C}_{alive}| \geq 2t + 1$.

*2) Proactivization (*Exp-Proactivize*):* The goal of this phase is to perform a bivariate 0-sharing, and identify and expel adversaries if malicious behavior is detected.

We first use a different zero-sharing protocol. Each node $\mathcal{U}'_i$ generates $2t+1$ sub-shares $\{s_{ij}\}_{j\in[2t+1]}$ that form a 0-sharing i.e., $\sum_{j=1}^{2t+1} \lambda_j^{2t} s_{ij} = 0$ where the Lagrange coefficients λ_j^{2t} are introduced before. $\mathcal{U}'_i$ then publishes $\{g^{s_{ij}}\}_{j\in[2t+1]}$ and $\{\mathsf{Enc}_{\mathsf{pk}_j}[s_{ij}]\}_{j\in[2t+1]}$ on-chain. $\mathcal{U}'_i$ also publishes a zk proof of correctness of the encrypted ciphertext. A receiving node $\mathcal{U}'_j$ verifies the set $\{g^{s_{ij}}\}_j$ using Eq. (1). Then, it decrypts the ciphertext to receive s_{ij}. Nodes publish verifiable accusations in case of a corruption. .

The advantage of this univariate zero-sharing protocol is that honest parties do not need to re-execute the protocol when an adversary is detected. They can simply discard the shares generated by the adversarial nodes. This is depicted pictorially in Fig. 18. One can see that by setting $s_j = \sum_{i\in\mathcal{U}'\setminus\mathcal{U}'_{corrupt}} s_{ij}$ for $j \in \mathcal{U}'$, the shares form a valid univariate zero-sharing among the honest parties.

After the univariate zero-sharing, the same protocol as that in Opt-Proactivize (step 6-16 in Figure 6) is executed with commitments and witnesses in step 12 posted on-chain. Finally, another major difference to the optimistic path is that if any adversary in the $\mathcal{U}'$ is expelled in this phase, we do not have enough nodes to recover the full shares in the next phase,

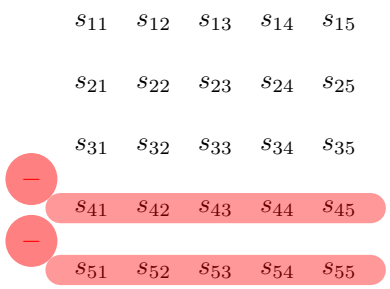


Fig. 18: Matrix of sub-shares. The sub-share s_{ij} is generated by node i and sent to j. Each node generates a row while it's share is the sum of sub-shares in a column. If nodes 4 and 5 are adversarial, sub-shares generated by them are discarded.

Exp-Proactivize

1 : **Public Input:** $\{C_{B(x,j)}\}_{j\in[2t+1]}$
2 : **Input:** Set of $2t + 1$ nodes $\{\mathcal{U}'_j\}_{j\in[2t+1]}$. Each node $\mathcal{U}'_j$ is given $B(x, j)$
3 : **Output:** $\mathcal{U}'_j$ outputs $B'(x, j)$ for a degree-$\langle t, k\rangle$ bivariate polynomial $B'(x, y)$ with $B'(0, 0) = B(0, 0)$
4 : **Public Output:** $\left\{C_{B'(x,j)}\right\}_{j\in[2t+1]}$
5 : node $\mathcal{U}'_i$:
6 : Generate $\{s_{ij}\}_{j\in[2t+1]}$ that form a 0-sharing i.e., $\sum_{j=1}^{2t+1} \lambda_j^{2t} s_{ij} = 0$.
7 : Publish $\{g^{s_{ij}}\}_{j\in[2t+1]}$ and $\{\mathsf{Enc}_{\mathsf{pk}_j}[s_{ij}]\}_{j\in[2t+1]}$ on-chain.
8 : node $\mathcal{U}'_j$:
9 : Decrypt $\{\mathsf{Enc}_{\mathsf{pk}_j}[s_{ij}]\}$ from node i and validate s_{ij} using $g^{s_{ij}}$ on-chain. Perform a verifiable accusation if it fails.
10 : node $\mathcal{U}'_j$:
11 : If any adversarial node i is detected in step 9, add it to $\mathcal{U}'_{corrupted}$, and publish s_{ji}.
12 : Set $s_j = \sum_{i\in\mathcal{U}'\setminus\mathcal{U}'_{corrupted}} s_{ij}$.
13 : Execute step 7-9, 11-12 of Opt-Proactivize in Figure 6, with messages posted on the chain in step 12.
14 : node $\mathcal{C}'$
15 : Execute step 16-17 of Opt-Proactivize in Figure 6. If it outputs `fail`, perform a verifiable accusation and add j to $\mathcal{U}'_{corrupted}$.
16 : node $\mathcal{C}_i$:
17 : For all malicious nodes j detected in step 9 and 15, publish point and witness $\{B(i, j), w_{i,j}\}$ on-chain.

Fig. 19: Exp-Proactivize protocol.

as the degree of $B'(i, y)$ is $2t$ and a member $\mathcal{C}'_i$ needs $2t + 1$ points to reconstruct the polynomial. To address this problem, we further ask members in the old committee to publish the shares and witnesses sent to the adversarial nodes during Opt-ShareReduce on the chain. In this way, all honest parties have access to those reduced shares that belong to adversarial nodes, which allows them to reconstruct the full shares in the next phase. The security of the new protocol still holds, as these shares were accessed by the adversary anyway.

The full protocol of Exp-Proactivize is presented in Figure 19. The on-chain cost of this phase is $O(t^2)$.

*3) Full Share Distribution (*Exp-ShareDist*):* Finally, full shares are distributed to all members of the new committee in this phase. To allow identification and expulsion of malicious nodes, members post all messages on the chain in contrast to

the optimistic path.

If adversarial nodes are detected in this phase, similar to the proactivization phase, we ask members of the old committee to publish the reduced shares sent to them in Opt-ShareReduce. In addition, honest members need to exclude the proactivization polynomials generated by the adversarial nodes in the second phase. In particular, they discard the sub-shares related to the adversaries in the new univariate zero-sharing protocol, as explained in the previous section, and post their sub-shares for the the adversaries publicly on-chain. Fortunately, this only incurs a small extra on-chain cost.

The full protocol of Exp-ShareDist is presented in Figure 20. The on-chain cost of this phase is also $O(tn)$. Therefore, the overall on-chain complexity of the Exp-CHURP-A is $O(n^2)$, and the off-chain complexity is $O(n^3)$.

B. State Verification (StateVerif)

The protocols of both Opt-CHURP and Exp-CHURP-A use the KZG commitment scheme, which requires a trusted setup phase and its security relies on the t-SDH assumption. In this section, we devise a hedge against the these — a verification phase, StateVerif, that relies *only* on discrete log assumptions.

Recall that by the end of Opt-CHURP or Exp-CHURP-A, each member $\mathcal{C}'_i$ in the new committee $\mathcal{C}'$ holds the full share $B'(i,y)$, a degree $2t$ univariate polynomial. StateVerif further checks that the invariants given in Section V-A still hold at this point. That is, (1) Inv-Secret: the secret is not changed; (2) Inv-State: these full shares form a $\langle t, 2t\rangle$ bivariate polynomial. (We don't check Inv-Comm as it is only used in the KZG scheme.) We describe the two checks below.

Exp-ShareDist

Public Input: $\left\{C_{B'(x,j)}\right\}_{j\in[2t+1]}$

Input: Set of nodes $\{\mathcal{C}'_i\}_{i\in[n']}$. Let $\mathcal{U}' = \{\mathcal{C}'_j\}_{j\in[2t+1]}$, each node $\mathcal{U}'_j$ is given $B'(x,j)$

Output: $\forall i \in [n']$, $\mathcal{C}'_i$ outputs $B'(i,y)$

node $\mathcal{U}'_j$:

$\forall i \in [n']$, publish $\mathsf{Enc}_{pk_i}(B'(i,j)), g^{B'(i,j)}, w'_{i,j}$ on-chain, where $w'_{i,j} = \mathsf{CreateWitness}(B'(x,j), i)$

node $\mathcal{C}'_i$:

Decrypt the message on-chain to get $\left\{B'(i,j), w'_{i,j}\right\}_{j\in[2t+1]}$

$\forall j \in \mathcal{U}' \setminus \mathcal{U}'_{corrupted}$, invoke $\mathsf{VerifyEval}(C_{B'(x,j)}, i, B'(i,j), w'_{i,j})$. If any of the checks fails, perform a verifiable accusation and add j to $\mathcal{U}'_{corrupted}$

node $\mathcal{C}_i$:

Publish $B(i,j), w_{i,j}$ for any new adversarial node j detected above.

node $\mathcal{U}'_i$:

Publish s_{ij} for any new adversarial node j detected above.

node $\mathcal{C}'_i$:

$\forall j \in \mathcal{U}'_{corrupted}$, validate their reduced shares posted by the old committee by $\forall i \in [n]$, $\mathsf{VerifyEval}(C_{B(x,j)}, i, B(i,j), w_{i,j})$.

$\forall j \in \mathcal{U}'_{corrupted}$ Interpolate any $t+1$ verified points to construct $B(x,j)$. Set $B'(i,j) = B(i,j) + \sum_{i\in honest} s_{ij}$

Interpolate all $B'(i,j)$ for $j \in [2t+1]$ to construct $B'(i,y)$

Output the full share $B'(i,y)$

Fig. 20: Exp-ShareDist protocol.

a) Checking Inv-Secret*:* To perform this check, we further require that the commitment to the secret g^s is public on the chain from the beginning of the protocols. The secret can also be computed from the zero points of the full shares. Using lagrange coefficients, we have $s = \sum_{i=1}^{n} \lambda_i B'(i,0)$ where $\lambda_i = \lambda_i^{n-1}$ (defined in Eq. (1)). Each node $\mathcal{C}_i$ computes $s_i = B'(i,0)$ and publishes g^{s_i}. Parties use this information to check that the invariant Inv-Secret remains intact:

$$g^s = \prod_{i=1}^{n} (g^{s_i})^{\lambda_i}$$

b) Checking Inv-State*:* As the degree of full shares $B'(i,y)$ is $2t$ (as they are interpolated from $2t+1$ points in ShareDist), to validate that $B'(i,y)$ for $i \in [n']$ form a degree $\langle t, 2t\rangle$ bivariate polynomial, it suffices to check that the degree of $B'(x,j)$ is t for $j \in [2t+1]$. To improve efficiency, we reduce the checks to a single check through a random linear combination. If the degree of $P_{rnd}(x) \stackrel{\text{def}}{=} \sum_{j=1}^{2t+1} r_j B'(x,j)$ is t, where r_js are randomly selected, then with high probability, the degree of all $B'(x,j)$ is t.

To perform this check, each node $\mathcal{C}'_i$ computes $s'_i = P_{rnd}(i) = \sum_{j=1}^{2t+1} r_j B'(i,j)$ and publishes $g^{s'_i}$ on-chain. In practice, r_js can be obtained from a public source of fresh randomness .

With these commitments of evaluations, all members can compute the commitments of the coefficients of $P_{rnd}(x)$ using Lagrange interpolation. Let $P_{rnd}(x) = a_0 + a_1 x + \ldots + a_{n'-1}x^{n'-1}$, then $a_j = \sum_{i=1}^{n'} P_{rnd}(i)\lambda_{ij}$, where λ_{ij} are coefficients of $\Lambda_i(x) = \sum_{j=1}^{n'} \lambda_{ij} x^j$ such that $P_{rnd}(x) = \sum_{i=1}^{n'} P_{rnd}(i)\Lambda_i(x)$. As λ_{ij} only depends on the degree of the polynomial, they can be precomputed by each member. Therefore, and $g^{a_j} = \prod_{i=1}^{n'} P_{rnd}(i)^{\lambda_{ij}}$. By checking that $\forall j > t, g^{a_j} = 1$ we can ensure that the degree of polynomial $P_{rnd}(x)$ is t.

The two checks above incur $O(n)$ on-chain cost in total.

Failure. There are two possible reasons that may cause StateVerif to fail: either the commitments are computed incorrectly by adversarial nodes, or the assumptions in the KZG scheme fails. We further perform the following test in case of a failure of StateVerif to determine the reason.

We make use of the on-chain KZG commitments (published in CHURP) to verify the commitments $Z_i = g^{s_i}$ and $Z_i^{rnd} = g^{s'_i}$. Each node i posts exponents of their state $\{g^{B'(i,j)}\}$ for $j \in [2t+1]$, and their witness $w'_{j,i}$ to the KZG polynomial commitments $C_{B'(x,j)}$ on the chain (each node already has these witnesses at the end of Opt-CHURP or Exp-CHURP-A). Then all members verify the message published by node i by: $\mathsf{VerifyEvalExp}(C_{B'(x,j)}, i, g^{B'(i,j)}, W_{j,i})$ for $j \in [2t+1]$.[2]

If the checks above pass, all members then validate Z_i, Z_i^{rnd} as:

[2]We make use of the following additional functionality in KZG scheme that allows us to verify the exponent of the evaluation without any changes to the scheme: $\{\mathsf{True}, \mathsf{False}\} \leftarrow \mathsf{VerifyEvalExp}(C_\phi, i, g^{\phi(i)}, W_i)$.

$$Z_i = \prod_{j=1}^{2t+1} (g^{B'(i,j)})^{\lambda_j^{2t}}, Z_i^{rnd} = \prod_{j=1}^{2t+1} (g^{B'(i,j)})^{r_j \lambda_j^{2t}}$$

If any of the checks above fails, it means the commitments are not correctly computed. The members can perform a verifiable accusations since all information is on-chain, and then switch to pessimistic path Exp-CHURP-A. Otherwise, it implies a failure of the assumptions in the KZG scheme. In this case, we switch to a different pessimistic path Exp-CHURP-B. In this test, each node publishes $O(n)$ data on-chain, incurring $O(n^2)$ on-chain cost overall.

C. Exp-CHURP-B

This pessimistic path is taken only after a detection of breach in the underlying assumptions of KZG commitment scheme.

In this path, we use relatively expensive polynomial commitments (Pedersen commitments) instead of KZG and supports a lower threshold on the number of adversarial nodes $n > 3t$. In the share reduction phase, as $n > 3t$, we rely on the error correcting mechanisms of Reed-Solomon codes to construct reduced shares, instead of the verification of KZG scheme. In the proactivization phase and full share distribution phase, we replace the KZG commitments and verification with the Pedersen commitments (step 13 in Figure 19 and step 5,8,12 in Figure 20). Exp-CHURP-B incurs $O(n^2)$ on-chain cost and $O(n^3)$ off-chain cost, assuming $n > 3t$. Due to the space limit, we omit the full protocol of Exp-CHURP-B.

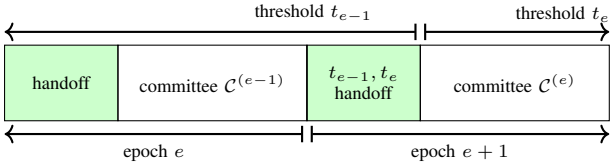
APPENDIX E
CHANGING THE THRESHOLD

Thus far we have focused on schemes that allow the committee size to change while the threshold t remains constant. This allows CHURP to be adaptive to changing churn rates: if an increased churn rate α is observed, the new committee can grow to a larger size of $^{2t}/_{1-\alpha}$.

We now describe how CHURP supports dynamic thresholds. Specifically, the (t_{e-1}, t_e)-handoff protocol presented below enables a committee $\mathcal{C}^{(e-1)}$ with threshold t_{e-1} (i.e. the adversary can corrupt up to t_{e-1} nodes of $\mathcal{C}^{(e-1)}$) to handoff shares to a new committee $\mathcal{C}^{(e)}$ with a different threshold t_e. Note that we assume an out-of-band mechanism by which the committee members reach the consensus to increase or decrease the threshold and leave details of governance for future work.

A. Increasing the threshold: $t_e > t_{e-1}$

Note that a change of the threshold reflects that of the adversary's power, i.e., the number of nodes it can corrupt in the committee $\mathcal{C}^{(e-1)}$ and $\mathcal{C}^{(e)}$, respectively. Therefore extra care is needed if we were to increase the power of the adversary (i.e. $t_e > t_{e-1}$). Similar to [17], increasing the threshold takes two steps: first, a handoff is executed between $\mathcal{C}^{(e-1)}$ and $\mathcal{C}^{(e)}$ assuming the threshold doesn't change; then we increase the threshold to t_e after the handoff. As illustrated below, the new threshold takes effect *after* the handoff.



To increase the threshold, (t_{e-1}, t_e)-handoff runs the proactivization phase with parameters $t = t_e$. That is, during the proactivization protocol, a bivariate polynomial $Q(x, y)$ of degree $(t_e, 2t_e)$ is generated such that $Q(0, 0) = 0$. Each node i holds a t_e-degree polynomial $Q(x, i)$ and commitments to $\{Q(x, i)\}_i$ are publicly available. The rest of the proactivization follows without modification, besides now each node i holds two polynomials with different degrees: $B'(x, i)$ that is t_{e-1}-degree while $Q(x, i)$ is t_e-degree. Thus the proactivized global polynomial $B'(x, y)$ is of degree $(t_e, 2t_e)$, concluding the threshold upgrade.

We also need to extend KZG to support dynamic thresholds. Essentially, the setup phase of the KZG fixes the highest degree (say, D) of polynomials it can work with. In the setting of a static threshold t, we set $D = t$ and Kate's commitment can guarantee that hidden polynomials are of degree $\leq t$, which is critical to the correctness of shares. To support dynamic thresholds up to $t_{\max}$, we run the trusted setup with $D = t_{\max}$ and extend the Kate's scheme with degree verification functionality. Specifically, given a commitment C_ϕ, it can be publicly verified that ϕ is at most d-degree. The extended scheme is specified in Fig. 21. Our extension relies on the q-power knowledge of exponent (q-PKE [68]) assumption.

KZG extended with degree verification

1) $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Keygen}(1^\lambda, q)$: Select a bilinear group $(p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathsf{BilGen}(1^\lambda)$, $q+1$ group elements $\{\alpha_i\}_{i\in[q]}$ and s randomly in $\mathbb{Z}_p^*$. Set $\mathsf{sk} = s$, $\mathsf{pk}_0 = \{g^s, \ldots, g^{s^d}\}$, $\mathsf{pk}_d = \{g^{\alpha_d s}, \ldots g^{\alpha_d s^d}\}$ for $d \in [q]$ and $\mathsf{pk} = \{\mathsf{pk}_0, \mathsf{pk}_1, \ldots, \mathsf{pk}_q\}$.

2) $C_\phi \leftarrow \mathsf{Commit}(\phi(x), \mathsf{pk})$: Let $d = \deg(\phi)$. Compute $C_\phi = (d, g^{\phi(s)}, g^{\alpha_d \phi(s)})$ using pk_0 and pk_d.

3) $(\phi(i), W_i) \leftarrow \mathsf{CreateWitness}(\phi(x), i, \mathsf{pk})$: Compute $\phi(x) - \phi(i) = (x - i)w(x)$, set $W_i = g^{w(s)}$.

4) $\{\mathsf{True}, \mathsf{False}\} \leftarrow \mathsf{VerifyEval}(C_\phi, i, \phi(i), W_i, \mathsf{pk})$: Parse C_ϕ as $(d, \mathsf{C}, \mathsf{C}_d)$. Output True if $e(\mathsf{C}/g^{\phi(i)}, g) = e(g^{s-i}, W_i)$. Otherwise, output False.

5) $\{\mathsf{True}, \mathsf{False}\} \leftarrow \mathsf{VerifyDegree}(C_\phi)$: Parse C_ϕ as $(d, \mathsf{C}, \mathsf{C}_d)$. Output True if $e(\mathsf{C}_d, g) = e(\mathsf{C}, g^{\alpha_d})$. Otherwise, output False.

Fig. 21: KZG [22] extended with degree verification.

B. Decreasing the threshold

The intuition of decreasing the threshold is to create $2 \times (t_{e-1} - t_e)$ *virtual nodes*, denoted as $\mathcal{V}$, and execute the handoff protocol between $\mathcal{C} = \mathcal{C}^{(e-1)}$ and $\mathcal{C}' = \mathcal{C}^{(e)} \cup \mathcal{V}$, assuming the threshold remains t_{e-1}. A virtual node participates in the protocol as if an honest player, but exposes its state publicly. At the end of the handoff protocol, nodes in $\mathcal{C}'$ incorporate $\mathcal{V}$'s state and restore the invariants. The handoff protocol is outlined as follows.

Decreasing the threshold

1) Choose a subset $\mathcal{U} \subseteq \mathcal{C}'$ of $2t_e + 1$ nodes. For notational simplicity, suppose $\mathcal{U} = \{1, \ldots, 2t_e + 1\}$ and $\mathcal{V} = \{2t_e + 2, \ldots, 2t_{e-1} + 1\}$. Each node $i \in \mathcal{U}$ recovers a reduced share $RS_i^{(e-1)}(x) = B(x, i)$. In addition, $\mathcal{C}$ publishes reduced shares for virtual nodes: $RS_j^{(e-1)}(x) = B(x, j)$ for $j \in \mathcal{V}$.

2) $\mathcal{U}$ executes the proactivization phase and collectively generate a $(t_e, 2t_e)$-degree bivariate zero-hole polynomial $Q(x, y)$. At the end of this phase, each node $i \in \mathcal{U}$ has $Q(x, i)$.

3) Let $V = \sum_{j \in \mathcal{V}} \lambda_j^{2t_{e-1}} RS_j^{(e-1)}(0)$. Each node $i \in \mathcal{U}$ incorprates virtual nodes' state and updates its state as $RS_i^{(e)}(x) = \frac{\lambda_i^{2t_{e-1}}}{\lambda_i^{2t_e}} \left(RS_i^{(e-1)}(x) + \frac{V}{\lambda_i^{2t_{e-1}(2t_e+1)}} \right) + Q(x, i)$ where $\lambda^{2t_{e-1}}$ and λ^{2t_e} are Lagrange coefficients for corresponding thresholds. One can verify that $RS_i^{(e)}(x)$ are $2t_e$-sharing of the secret, i.e., $B(0, 0)$ can be calculated from any $2t_e + 1$ of $RS_i^{(e)}(x)$.

4) Each node $i \in \mathcal{U}$ sends to every node $j \in \mathcal{C}'$ a point $RS_i^{(e)}(j)$. The full share of each node $j \in \mathcal{C}'$ consists of $2t_e + 1$ points $\{RS_i^{(e)}(j) = B'(i, j)\}_{i \in \mathcal{U}}$, from which j can compute $FS_j(y) = B'(j, y)$.

The updated reduced shares $RS_i^{(e)}(x)$ can be verified using the published value V, and the commitment to $RS_i^{(e-1)}(x)$ and $Q(x, i)$. At the end of the protocol, each node i has $2t_e + 1$ points on $B'(i, y)$. It remains to show that $\{FS_j(y) = B'(j, y)\}_j$ form a t_e-sharing of $B^{(e)}(0, 0)$, which can be checked by $\sum_{i=1}^{t_e+1} \lambda_i^{t_e} FS_i(0) = \sum_{j=1}^{2t_{e-1}+1} \lambda_j^{2t_{e-1}} RS_j^{(e-1)}(0) = B(0, 0)$.

Several optimizations are possible. For example, one can reduce the degree of $RS_i^{(e)}(x)$ to t_e (as opposed to t_{e-1} currently) by building new polynomials and proving equivalence to $RS_i^{(e-1)}(x)$. We leave further optimization for future work.