CHURP: Dynamic-Committee Proactive Secret Sharing

Sai Krishna Deepak Maram*†
Cornell Tech

Fan Zhang*†
Cornell Tech

Lun Wang*
UC Berkeley

Andrew Low*
UC Berkeley

Yupeng Zhang*
Texas A&M

Ari Juels*
Cornell Tech

Dawn Song*
UC Berkeley

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.

KEYWORDS

secret sharing; dynamic committee; decentralization; blockchain

ACM Reference Format:
Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. 2019. CHURP: Dynamic-Committee Proactive Secret Sharing. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3319535.3363203

1 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 [6], 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 [63]. Many users thus store their cryptocurrency with exchanges such as Coinbase, which holds at least 10% of all circulating Bitcoin [9]. 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 $P(x)$ [67]. 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. [44], 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., [10, 19, 27, 47, 74], 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 [17, 44], assume weak, passive adversaries [26, 64], or incur prohibitive communication costs [12, 54, 66, 71, 73].

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

1.1 CHURP functionality

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

*Also part of IC3, The Initiative for CryptoCurrencies & Contracts
†The first two authors contributed equally to this work.

CCS '19, November 11–15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6747-9/19/11…$15.00
https://doi.org/10.1145/3319535.3363203

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 *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 $O(n^2)$ on-chain communication complexity (no off-chain cost). Both communication 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.

1.2 Technical challenges and solutions

CHURP is the *first* dynamic committee PSS scheme with an end-to-end implementation that is practical even for large committees. To achieve its low communication complexity, CHURP overcomes several technical challenges in a different manner than the prior work aimed at dynamic committees, as explained below.

The first challenge is that previous PSS schemes, relying on techniques from Herzberg et al. [44], 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. Alternative approaches to PSS that do not explicitly generate a shared polynomial exist [33, 62], but CHURP's 0-sharing technique is of independent interest: It can also lower the communication complexity of Herzberg et al. [44] 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., [66], address this problem using "blinding" approaches with costly communication, while [12], address it via impractical virtualization techniques. Instead, CHURP uses a low communication-complexity technique called *dimension-switching*, that is based on known share resharing techniques. 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. Each switching involves a round of share resharing. Although dimension-switching is based on known techniques, CHURP's novelty lies in applying them to the dynamic committee setting to tolerate $2t$ compromises.

Finally, most PSS schemes commit to secret degree-t polynomials using classical schemes (e.g., [30, 59]) with per-commitment size $O(t)$. CHURP uses an alternative due to Kate, Zaverucha, and Goldberg (KZG) [45] with size $O(1)$. Use of KZG for secret sharing isn't new [11], 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 $t < n/3$. The detection mechanisms used to hedge are efficient—$O(n)$ on-chain—and are KZG-free—so, our techniques can easily be adapted to future secret-sharing schemes that rely similarly on KZG or related non-standard assumptions.

We compose these techniques to realize CHURP with *provable security* and give a rigorous security proof.

1.3 Implementation and Experiments

We present an implementation of CHURP. Our experiments show very practical communication and computation costs—at least 1000x improvement over the existing state-of-the-art dynamic-committee PSS scheme [66] in the off-chain communication complexity for large committees (See Section 6).

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.

1.4 Outline and Contributions

After introducing the functional, adversarial, and communication models in Section 2, we present our main contributions:

- *CHUrn-Robust Proactive secret sharing (*CHURP*):* In Section 3, we introduce CHURP, a 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 4, *dimension-switching* technique to safeguard the secret in committee handoffs in Section 5.3, and hedging techniques for failures in the KZG commitment scheme in Section 5.5.
- *New point-to-point blockchain communication technique:* We introduce a novel point-to-point communication technique for permissionless blockchains in Section 7—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 6 and present performance measurements demonstrating its practicality.

We give a security proof for CHURP in Appendix A. We discuss related work in Section 8 and CHURP's many potential applications—threshold cryptography, smart contracts with private keys, consensus simplification for light clients, etc.—in Appendix B. We have released the CHURP system as an open-source tool at https://www.churp.io.

2 MODEL AND ASSUMPTIONS

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

In a secret-sharing scheme, a *committee* of nodes shares a fixed secret s. Let C denote a committee and $\{C_i\}_{i=1}^{n}$ denote the n nodes in

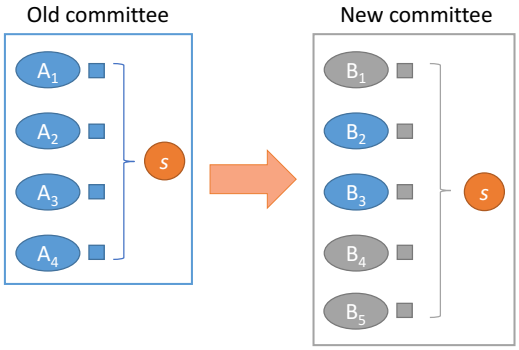


Figure 1: Handoff between two committees at the end of a dynamic proactive secret-sharing epoch. The secret s remains fixed. Committees may intersect, e.g., $B_2 = A_2$ and $B_3 = A_3$.

the committee. Each node 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 [12, 66], 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.

2.1 Functional model

Epoch: Time in CHURP, as in any proactive secret-sharing scheme [44], 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 $C^{(e)}$ of size $N^{(e)}$ shares s using a $(t, N^{(e)})$-threshold scheme.

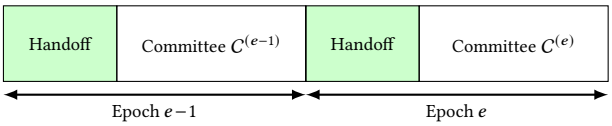


Figure 2: Each epoch begins with a handoff phase where the old committee hands off the secret s to the new committee. It is followed by a period of committee operation.

Handoff: Fig. 2 depicts the handoff at the beginning of an epoch. It involves a transfer of s from an old committee, which we denote $C^{(e-1)}$, to a new committee, denoted $C^{(e)}$. Prior to completion of the handoff, $C^{(e-1)}$ 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 $C^{(e-1)}_{left}$ denote the set of nodes that have left the committee before the handoff in epoch e. Let $C^{(e-1)}_{alive} = C^{(e-1)} \setminus C^{(e-1)}_{left}$ denote the set of nodes that participate in the handoff. We let *churn rate* α denote a bound such that $|C^{(e-1)}_{alive}| \geq |C^{(e-1)}|(1-\alpha)$. Later, we provide a lower bound on the committee size using the 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 [44, 66].

2.2 Adversarial model

We consider a powerful active adversary $\mathcal{A}$. 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 $C^{(e-1)}$ and t' nodes in $C^{(e)}$. We present the protocol in CHURP for threshold changes in Section 5.4. 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, $C^{(e-1)}$ and $C^{(e)}$, 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.

Definition 1. *A protocol for* dynamic-committee proactive secret sharing *satisfies the following properties in the functional model above for any probabilistic polynomial time adversary $\mathcal{A}$ with threshold t:*

***Secrecy**: If $\mathcal{A}$ corrupts no more than t nodes in a committee of any epoch, $\mathcal{A}$ learns no information about the secret s.*

***Integrity**: If $\mathcal{A}$ corrupts no more than t nodes in each of the committees $C^{(e-1)}$ and $C^{(e)}$, after the handoff, the shares for honest nodes can be correctly computed and the secret s remains intact.*

2.3 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:

On-chain: Existing approaches such as MPSS [66] use PBFT [18] 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, and more modular 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. This model is widely adopted in the literature (e.g., See [48, 58, 72]).

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.

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 time period, T'.

We emphasize that synchronicity of the P2P network is required only for performance, *not* for liveness, secrecy or integrity. Looking ahead, without enough synchronicity, the off-chain protocol halts and the execution switches to the on-chain channel. In other words, an adversary may slow down the protocol execution temporarily by delaying messages, but she cannot learn or corrupt the secret. Moreover, CHURP only requires a short period of synchronicity (e.g., a few minutes) at the end of every epoch (a relatively long epoch, e.g., a day, would be the norm for CHURP). We discuss synchronicity assumptions in Section 5.3.3.

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 [69], 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 [28] for inexpensive off-chain communication in Section 7.

We measure off-chain communication complexity as the total number of bits transmitted in P2P 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.

3 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.

3.1 Key secret-sharing techniques

Recall that in an ordinary (t,n)-threshold Shamir secret sharing (see [67]), 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. (Details in Section 4).

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—and vice versa. We apply known resharing techniques [24, 32] via bivariate polynomials to switch between different sharings. 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. (Details in Section 5.3.)

3.2 CHURP: Overview

We now give an overview of CHURP execution. We first consider the optimistic case, and discuss pessimistic cases below in Section 3.5.

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

The committee $C^{(e-1)}$ 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 $C_i^{(e-1)}$ holds share $s_i = B(i,y)$, and can compute $B(x,0)$ for $x = i$. So it is easy to see that 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 $C^{(e-1)}$ and $C^{(e)}$ switch their sharing of s to the degree-$2t$ dimension of $B(x,y)$, resulting in what we call *reduced* shares.

Specifically, node $C_j^{(e)}$ holds share $s_j = B(x,j)$. Node $C_j^{(e)}$ 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 $C^{(e-1)}$ and $C^{(e)}$.

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 4). $Q(x,y)$ 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 $C^{(e)}$. At this point, the steady state is achieved for epoch e. Committee $C^{(e)}$ holds a (t,n)-sharing of s using $B'(x,y)$.

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

- *Share reduction*: Nodes switch from the degree-t dimension of $B(x,y)$ to the degree-$2t$ dimension. As a result, each node $C_j^{(e)}$ 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 $C_j^{(e)}$ 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, remaining nodes in old committee can still perform operations using s. So there is no operational discontinuity in CHURP.

3.3 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 $C^{(e-1)} = \{A_1,A_2,A_3\}$. A_3 leaves at the end of the epoch, and a new node A_3' joins. The new committee is thus $C^{(e)} = \{A_1,A_2,A_3'\}$. 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, as follows.

Share reduction: To start the handoff, each node j in the new committee constructs its reduced share $B(x,j)$ from points received from $C^{(e-1)}$. 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 $B(x,2)$.

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 $Q(x,y)$.

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 $B'(1,y)$.

3.4 Active 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 5.3—are required against an active adversary. These proofs do not alter the overall structure of the protocol.

3.5 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 introduces a *novel* hedge against any soundness failure of the KZG scheme, due to either a compromised trusted setup or a falsified hardness assumption (t-SDH). The hedging technique is efficient and incurs only $O(n)$ on-chain cost to detect such failures. 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. Unlike the optimistic path, the two pessimistic paths do not use the off-chain channel and incur $O(n^2)$ on-chain cost. Opt-CHURP and Exp-CHURP-A requires $t<n/2$, while Exp-CHURP-B requires $t<n/3$. We give more details on all the paths in CHURP in Section 5.

4 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. The bivariate 0-sharing protocol uses resharing techniques [24, 32] as a building block.

Recall that in the context of bivariate polynomials, 0-sharing means having a committee C generate a $\langle t,2t\rangle$-bivariate polynomial $Q(x,y)$ such that $Q(0,0)=0$. Each node 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 5) must then be invoked. Even in the pessimistic case, though, our techniques incur no more cost 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 C_i thereby obtains share $Q(i,y)$ of bivariate polynomial $Q(x,y)$, as desired.

BivariateZeroShare is formally specified in Fig. 10. (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 adversaries, is detailed in Section 5.3.

First step—Sharing $P(x)$: As noted, BivariateZeroShare first chooses a subset $\mathcal{U} \subseteq C$ of $2t+1$ nodes, i.e., $|\mathcal{U}| = 2t+1$. This can be done

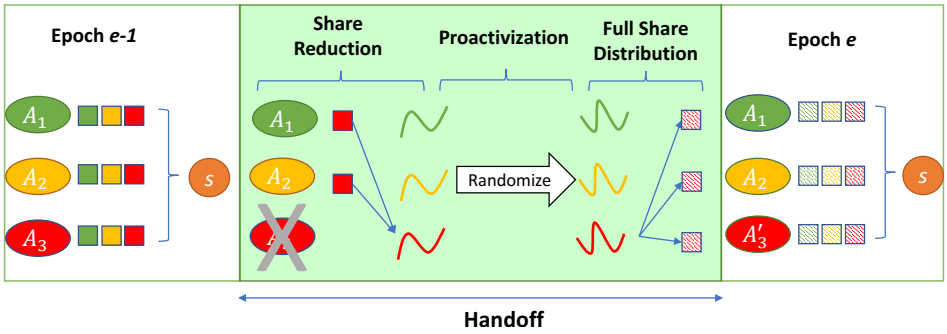


Figure 3: An example of the handoff protocol: Curves denote univariate polynomials (reduced shares) while squares denote points on these polynomials. See Section 3.3 for a description.

as follows: Order nodes lexicographically by their public keys and choose the first $2t+1$. Without loss of generality, $\mathcal{U} = \{C_j\}_{j=1}^{2t+1}$.

The nodes of $\mathcal{U}$ then execute the univariate 0-sharing subprotocol UnivariateZeroShare presented in Fig. 9. This subprotocol is not new—it was previously used for proactivization in [44]. Each node $\mathcal{U}_j$ generates a degree-$2t$ univariate 0-hole polynomial $P_j(x)$.[1] 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 $s_i = P(i)$.

Second step—Resharing $P(x)$: Nodes in $\mathcal{U}$ now reshare $P(x)$ among all of 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 C_i in the full committee. Using the received points, each committee member 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 [44], i.e., the static-committee setting, by a factor of $O(n)$. Due to lack of space, we present this application in the online full version of the paper [50].

5 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.

[1] An attack is outlined in [51] that breaks the UnivariateZeroShare protocol in [44]. It does so in an adversarial model similar to ours, i.e., the adversary controls t nodes in old and new committees and thus $2t$ in total, rather than t in total as in [44]. CHURP defeats this attack via dimension-switching, using reduced shares during the handoff.

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 the optimistic path, Exp-CHURP-A exclusively uses on-chain communication channel, which allows to identify and 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 communication in the worst case.

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, i.e., binding property of KZG is broken. 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, $t < n/3$. More details on StateVerif in Section 5.5.

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 communication 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 2 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 C.

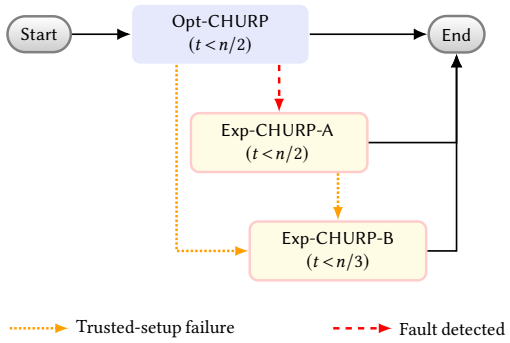


Figure 4: CHURP protocol tiers. Opt-CHURP is the default protocol of CHURP. Exp-CHURP-A and Exp-CHURP-B are run only if a fault occurs in Opt-CHURP.

5.1 Notation and Invariants

We now introduce the notation and invariants that will be used to explain the protocols of CHURP. Notation is summarized in Table 1.

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 in Fig. 8 and [45].)

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 C be a committee of n nodes $\{C_i\}_{i=1}^{n}$. 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 three invariants below must hold:

- Inv-Secret: The secret s is the same across handoffs.
- Inv-State: Each node 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 3 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 nodes 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.

Notation	Description
$C^{(e-1)}, C^{(e)}$	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 C_i
$FS_i(y) = B(i,y)$	Full share held by 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 1: Notation

5.2 CHURP Setup

The setup phase of CHURP sets the system to a proper initial steady state. To start, an initial committee $C^{(0)}$ is selected. The setup of KZG is performed and the secret is shared among $C^{(0)}$. Using their shares, members of $C^{(0)}$ 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, e.g., [37]. We leave committee selection out-of-scope for this paper. Readers can refer to, e.g., [40], for a discussion.

5.3 CHURP Optimistic Path (Opt-CHURP)

Recall that Opt-CHURP transfers shares of some secret s from an old committee, denoted $C = C^{(e-1)}$, to a new committee $C' = C^{(e)}$. CHURP can support both committee-size and threshold changes, i.e., a transition from (n,t) to some (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 Section 5.4.

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 returns to the steady state.

At the beginning of Opt-CHURP, each node in C' requests the set of KZG commitments from any node in C, say C_1. Recall that by the invariant Inv-Comm, each node in 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 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 above check enabled by the on-chain hashes ensures that new committee nodes receive the correct set of commitments. The phases of Opt-CHURP are as follows:

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

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

Recall that by the invariant Inv-State, each node C_i holds a full share $B(i,y)$. Now, 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, C_i sends $B(i,j)$ to $\mathcal{U}'_j$, which interpolates the received points to get its reduced share $B(x,j)$.[2] 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 ($|C_{alive}| > 2t$) to guarantee liveness.

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

The protocol Opt-ShareReduce is formally specified in Fig. 11. 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.

*5.3.2 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 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 4 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 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:

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

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 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 were received previously.

In total, each node $\mathcal{U}'_j$ generates the following set of commitment and witness information during Opt-Proactivize, $\left\{g^{s_j}, C_{Z_j}, W_{Z_j(0)}, C_{B'(x,j)}\right\}$. While this set is transmitted off-chain to all nodes in the full committee 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 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. 12. 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 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.

*5.3.3 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 $\mathcal{U}' \subseteq C'$ holds a new reduced share $B'(x,j)$.

[2] Dimension-switch can be thought as a resharing of the shares. The zero points on full shares $B(i,0)$ i.e., shares of the secret s, are reshared.

Protocol	On-chain, Off-chain	Threshold	Optimistic
Opt-CHURP	$O(n)$,$O(n^2)$	$t<n/2$	Yes
Exp-CHURP-A	$O(n^2)$,n/a	$t<n/2$	No
Exp-CHURP-B	$O(n^2)$,n/a	$t<n/3$	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 2: On-chain costs and Off-chain costs for the dynamic setting. An optimistic protocol ends successfully only if no faulty behavior is detected. n/a **indicates Not Applicable.**

Now, $\mathcal{U}'_j$ distributes points on $B'(x,j)$, allowing computation of full shares $B'(i,y)$ by all members of C'—we make a *dimension-switch* from the degree-$2t$ dimension of $B'(x,y)$ to the degree-t dimension. Specifically, each 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. 13. 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 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 C delete their old full shares and members of $\mathcal{U}'$ delete their new reduced shares.
Communication complexity: Each node in C' receives $2t+1$ points, thus Opt-ShareDist incurs $O(nt)$ off-chain cost.

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. 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, we reiterate that the pessimistic paths of CHURP are discussed in Appendix C. Table 2 compares on-chain and off-chain costs of the three paths of CHURP and Schultz-MPSS [66], the latter will be explained in more detail in Section 6.3.1.

Theorem 1. *Protocol* Opt-CHURP *is a dynamic-committee proactive secret sharing scheme by Definition 1.*

We present the security proof in Appendix A.

Notes on the synchronicity assumptions. As discussed in Section 2, CHURP works in the synchronous model and assumes a latency bound for both on-chain and off-chain communication. While the former is a well-accepted assumption (e.g., see [48, 58, 72]), the latter is assumed by the blockchain consensus protocol itself, as the required difficulty of proof-of-work is dependent on the maximum network delay [57]. However, we emphasize that synchronicity for off-chain communication is needed only for performance, *not* for liveness or safety of the full protocol. In the optimistic path, if messages take longer to deliver, a fault is detected and the protocol switches to the pessimistic path. After that, nodes communicate via the on-chain channel only.

5.4 Change of threshold

Thus far we have focused on schemes that allow the committee size to change while the threshold t remains constant. We now briefly describe how to enable an old committee with threshold t_{e-1} (i.e. the adversary can corrupt up to t_{e-1} nodes) to hand off shares to a new committee with a different threshold t_e.

Generally, we follow the same methodology as that of [53, 66]. To increase the threshold (i.e., $t_e > t_{e-1}$), the new committee generates a $(t_e, 2t_e)$-degree zero-hole polynomial $Q(x,y)$ so that the proactivized sharing has threshold t_e. To reduce the threshold (i.e., $t_e < t_{e-1}$), the old committee creates $2\times(t_{e-1}-t_e)$ *virtual servers* that participate in the handoff as honest players, but expose their shares publicly. At the end of the handoff, the new commitment incorporates the virtual servers' shares to form a sharing of threshold t_e in a similar process as the public evaluation scheme in [53].

To make changes of the threshold verifiable, we also need to extend the KZG commitment scheme with the degree verification functionality such that given a commitment $C_{\phi,d}$ to a polynomial ϕ, it can be publicly verified that ϕ is at most d-degree. Our extension relies on the q-power knowledge of exponent (q-PKE [41]) assumption. Due to lack of space, we refer readers to Appendix D for more details.

5.5 State Verification (StateVerif)

Both Opt-CHURP and Exp-CHURP-A make use of the KZG commitment scheme, which requires a trusted setup phase and its security (binding property) relies on the t-SDH assumption. Now, we devise a *hedge* against these—a verification phase that relies only on discrete log assumptions. At a high level, StateVerif includes checks to ensure that the two important invariants, Inv-Secret and Inv-State, hold, without using the KZG commitments on-chain.

Checking Inv-Secret*:* Assume that the commitment to the secret g^s is on-chain from the beginning (done as part of the setup phase). Recall that at the end of Opt-CHURP or Exp-CHURP-A, each new committee node C'_i holds a full share $B'(i,y)$. The secret can also be computed from the zero points of the full shares, $s=\sum_{i=1}^{n}\lambda_i B'(i,0)$, where $n=|C'|$ and $\lambda_i=\lambda_i^{n-1}$ as defined in Eq. (1). Each C'_i computes $s_i=B'(i,0)$ and publishes g^{s_i}. All nodes verify that Inv-Secret remains intact by checking $g^s=\prod_{i=1}^{n}(g^{s_i})^{\lambda_i}$.

Checking Inv-State*:* In this check, we ensure that the bivariate polynomial $B'(x,y)$ is of degree $\langle t,2t\rangle$. We achieve this by checking that the $2t+1$ reduced shares $\{B'(x,j)\}_{j\in[2t+1]}$ are of degree t. We build an efficient procedure that reduces the checks to a single check through a random linear combination. If the degree of $P_r(x)\stackrel{\text{def}}{=}\sum_{j=1}^{2t+1}r_j B'(x,j)$ is t, where r_js are chosen randomly, then with high probability, the degree of all $B'(x,j)$ is t. It is important that the adversary does not know the randomness a priori, as adversarial nodes can then choose reduced shares of degree $>t$ (in the proactivization phase) in such a way that the higher degree coefficients cancel in the linear combination. In practice, r_js can be obtained from a public source of randomness [15].

Each C'_i computes $s'_i=P_r(i)=\sum_{j=1}^{2t+1}r_j B'(i,j)$ and publishes $g^{s'_i}$ on-chain. All nodes now compute powers of the coefficients of P_r. Let $P_r(x)=\sum_{j=1}^{n}a_j x^j$, then $a_j=\sum_{i=1}^{n}\lambda_{ij}P_r(i)$, where λ_{ij} are Lagrange coefficients (an extension of Eq. (1)). Therefore, $g^{a_j}=\prod_{i=1}^{n}(g^{s'_i})^{\lambda_{ij}}$. All nodes check $\forall j>t, g^{a_j}=1$, thus $P_r(x)$ is t-degree.

The two checks above incur $O(n)$ on-chain cost in total, thus StateVerif is highly efficient. StateVerif can fail due to two possible reasons: either the commitments are computed incorrectly by adversarial nodes, or the assumptions in the KZG scheme fail. Additional tests need to be performed to determine the cause of failure, these incur $O(n^2)$ on-chain cost and are discussed in Appendix C.2. If adversarial nodes are detected, the protocol expels these nodes and switches to Exp-CHURP-A. On the other hand, if KZG assumptions fail, the protocol switches to Exp-CHURP-B.

6 CHURP IMPLEMENTATION & EVALUATION

We now report on an implementation and evaluation of CHURP, including a comparison with the state-of-the-art alternative, Schultz-MPSS [66].

6.1 Implementation

We implemented Opt-CHURP in about 2,100 lines of Go and the code is available at https://www.churp.io. Our implementation uses the GNU Multiprecision Library [3] and the Pairing-Based Cryptography Library [5] for cryptographic primitives, and gRPC [4] for network infrastructure.

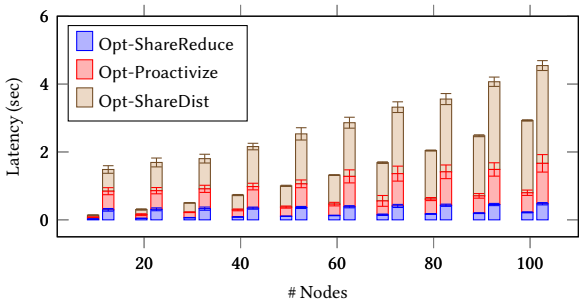
For polynomial arithmetic, we used the polynomial ring $\mathbb{F}_p[x]$ for a 256-bit prime p. For the KZG commitment scheme, we used a type A pairing on an elliptic curve $y^2 = x^3 + x$ over $\mathbb{F}_q$ for a 512-bit q. The order of the EC group is also p. We use SHA256 for hashing.

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

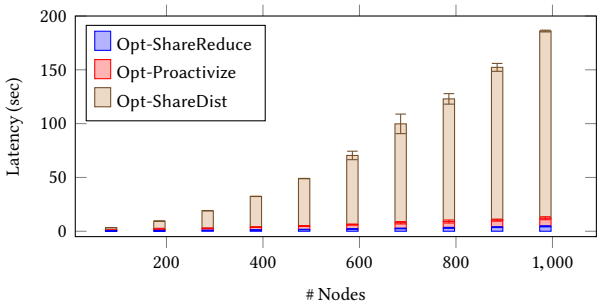
6.2 Evaluation

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 protocol epoch: the latency (the total execution time), the on-chain complexity (the total bytes written to the blockchain (i.e. the trusted node)), and the off-chain complexity (the total bytes transmitted between all nodes). The evaluation results are presented below.

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 computation time of Opt-CHURP, results are presented in Fig. 5. 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. 5b). 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. 5b also shows that among the three phases, Opt-ShareDist dominates the execution time due to the relatively expensive $O(n)$ calls to KZG's



(a) Latency for the LAN (left bar) and WAN (right bar) setting with committee sizes 11-101.



(b) Latency for the LAN setting with committee size 101-1001.

Figure 5: Latency

CreateWitness per node. (CreateWitness involves $O(n)$ group element exponentiation, thus total $O(n^2)$ 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. 5a), 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 AWS. (We plan to get around this soon.)

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.

Off-chain communication complexity: Fig. 6 compares the off-chain complexity for different committee sizes for Opt-CHURP and [66], a discussion about the comparison is in Section 6.3.1. Now, we discuss the off-chain costs of Opt-CHURP. The concrete performance numbers are consistent with the expected $O(n^2)$ 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 65B (with compression) and points on

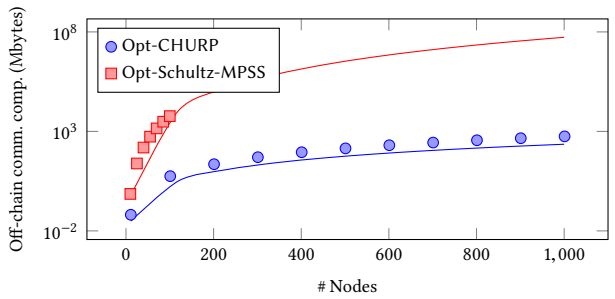


Figure 6: Concrete off-chain communication complexity for Opt-CHURP **and** Schultz-MPSS**, with log-scale y-axis. Points show experimental results; expected polynomial curves (respectively quadratic and quartic) are also shown.**

polynomial are 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. 6, the expected curve is slightly below the observed data points due to 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.

6.3 Comparison with other schemes

6.3.1 Schultz's MPSS. The Mobile Proactive Secret Sharing (MPSS) protocol of Schultz et al. [66], referred to as Schultz-MPSS hereafter, achieves the similar goal as CHURP in asynchronous settings, assuming $t < n/3$. Compared to [66], Opt-CHURP achieves an $O(n^2)$ improvement for off-chain communication complexity. To evaluate the concrete performance, we also implemented the optimistic path of Schultz-MPSS (Section 5 of [66]) and evaluated the communication complexity empirically.

Asymptotic improvement: Schultz-MPSS extends the usage of expensive blinding polynomials introduced by Herzberg et al. [44] to enable a dynamic committee membership. We recall briefly the asymptotic complexity of Schultz-MPSS and refer readers to [66] 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 2 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.

Performance evaluation: We implemented the optimistic path of Schultz-MPSS 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. 6 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 complexity of Schultz-MPSS is $53.667n^4 \approx 5.3$GB, whereas that for Opt-CHURP is only 2.3MB, a 2300x improvement! (If $n \geq 65$, the improvement is at least three orders of magnitude.) Since Schultz-MPSS incurs excessive (GB) off-chain cost, we do not run it for committee sizes beyond 100.

6.3.2 Baron et al. [12]. Baron et al. devise a batched secret-sharing scheme that incurs $O(n^3)$ cost to transfer $O(n^3)$ secrets from an old to a new committee. In the single secret setting of CHURP, [12] achieves worse asymptotic cost than CHURP's optimistic path ($O(n^3)$ vs $O(n^2)$) and equivalent in the pessimistic case. The asymptotic cost, though, masks the much worse practical performance caused by the use of impractical techniques to boost corruption tolerance. The implications are twofold. First, their protocol only works when the committee size is large (hundreds to thousands as we explain below), whereas CHURP works for arbitrary committee sizes. Second, even with a large committee, their protocol requires large subgroups of nodes (hundreds to thousands) to run maliciously-secure MPC, making their protocol significantly more expensive in practice.

The bottleneck in [12] lies in the use of virtualization techniques to achieve corruption threshold close to $t < n/2$. Virtualization involves two steps: first, the committee of size n is divided into n virtual groups of size $s < n$; then each group is treated as a node in the committee to execute the protocol using MPC. [12] uses the group construction techniques of [22] that only work for large committees: for a fixed $\epsilon > 0$, to achieve a corruption threshold $t < (1/2 - \epsilon)n$, the size of the constructed group is $16/\epsilon^2$ (See Appendix B.2 of [22]). We want ϵ to be small, e.g., $\epsilon = 0.01$—yielding t only slightly worse than CHURP. This, however, causes the group size to explode to $s = 160{,}000$. Even choosing a moderate ϵ, say $\epsilon = 1/6$—yielding $t < n/3$ which is worse than CHURP, still requires a group of size $s = 576$, meaning [12] needs to be run using maliciously-secure MPC among $n > 576$ groups of 576 nodes each, making it extremely impractical.

7 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.

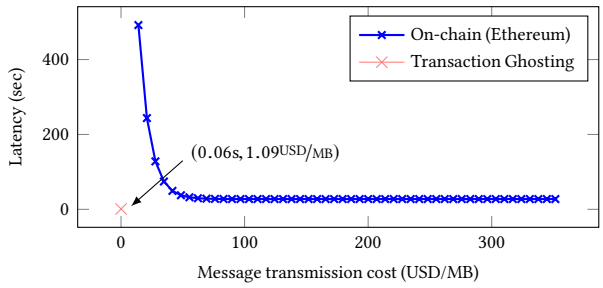


Figure 7: Tradeoff in latency vs. message transmission cost. The blue curve shows the on-chain tradeoff. The red dot at $(0.06\text{s}, 1.09^{\text{USD/MB}})$ corresponds to Transaction Ghosting.

We note that 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., [29].

7.1 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* [56]. 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 pays *nothing* for the data containing m, only the cost associated with tx'. This technique also 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 only pay the cost of the final transaction tx_k.

	On-chain	Transaction Ghosting
Bandwidth (KB/sec)	≤ 6.4	32.3 (9.31)
Latency (sec)	varies (Fig. 7)	1.09 (0.82)
Message transmission cost (USD/MB)	varies (Fig. 7)	$0.06 ($0.02)
Transaction delivery rate	100%	92.2% (14.2%)

Table 3: Comparison between communication via the Ethereum blockchain and via Transaction Ghosting. Numbers in parentheses are standard deviations. See Appendix E for details.

Here we summarize the results of the our experiments in Table 3 and Fig. 7, deferring details to Appendix E. Empirically, by employing Transaction Ghosting on the Ethereum blockchain, we can build cheap P2P messaging channels with an average bandwidth of 32.3KB/sec (5x the throughput upper bound of on-chain communication). Our technique achieves extremely low cost and latency compared with on-chain communication. As shown in Fig. 7, the cost of sending 1MB of data using Transaction Ghosting is $0.06 and the latency is 1.09 seconds. The lowest latency in on-chain communication is about 10-15 seconds [1, 2], costing hundreds of dollars per megabyte. To summarize, Transaction Ghosting enables efficient P2P communication in a decentralized setting, which we can leverage in CHURP and is of independent interest.

Protocol	Dynamic	Adversary	Network	Threshold	Cost
Herzberg et al. [44]	No	active	synch.	$t < n/2$	$O(n^2)$
Cachin et al. [17]	No	active	asynch.	$t < n/3$	$O(n^4)$
Desmedt et al. [24]	Yes	passive	synch.	$t < n/2$	$O(n^2)$
Wong et al. [71]	Yes	active	synch.	$t < n/2$	$\exp(n)$
Zhou et al. [73]	Yes	active	asynch.	$t < n/3$	$\exp(n)$
Schultz-MPSS [66]	Yes	active	asynch.	$t < n/3$	$O(n^4)$
Baron et al. [12]	Yes	active	synch.	$t < n(1/2-\epsilon)$	$O(n^3)$
CHURP (this work)	Yes	active	synch.	$t < n/2$	$O(n^2)$ (optimistic) $O(n^3)$ (pessimistic)

Table 4: 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 commn. complexity.

8 RELATED WORK

Verifiable Secret Sharing (VSS): Polynomial-based secret sharing was introduced by Shamir [67]. Feldman [30] and Pedersen [59] 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. [45] (KZG) reduces this to $O(1)$, and is adopted for secret sharing in, e.g., [11], and in CHURP.

KZG **hedge:** Prior works [42] hedge against the failure of a commitment scheme (or a cryptosystem [13]) by creating hybrid schemes that combine multiple schemes, in contrast to CHURP's approach of using protocol tiers with different schemes in each tier. This approach coupled with novel, efficient detection techniques to switch between tiers (StateVerif), allows CHURP to include an efficient top tier (optimistic path). The notion of graceful degradation in the event of a failure appears in several works [13, 35, 65]—loosely similar to how CHURP degrades to a lower corruption threshold when the KZG scheme fails (exact notion hasn't appeared before).

Proactive Secret Sharing (PSS): Proactive security, the idea of refreshing secrets to withstand compromise, was first proposed by Ostrovsky and Yung [55] for multi-party computation (MPC). It was first adapted for secret sharing by Herzberg et al. [44], whose techniques continue to be used in subsequent works, e.g., [16, 20, 36, 43, 49, 53, 66], 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 [44]. (Details in the full version [50].)

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 [17, 66, 73] and unconditional settings [54, 68]. Nikov and Nikova [51] 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 [24] propose a scheme that can change the committee and threshold in a secret-sharing system, but is unfortunately not verifiable. Wong et

al. [71] build a verifiable scheme assuming that the nodes in the new committee are non-faulty. Subsequent works [12, 25, 73] build schemes that do not make such assumptions, but are impractical for our use—[73] incurs exponential communication cost, [25] incurs exponential computation cost, and [12] uses impractical virtualization techniques (See Section 6.3.2). Schultz et al. [66] were the *first* to build a *practical* scheme under an adversarial model similar to ours. While [66] incurs $O(n^4)$ off-chain communication cost, as Table 4 shows, CHURP improves it to worst-case $O(n^3)$ off-chain cost ($O(n^2)$ in the optimistic case). We convert the on-chain cost incurred by CHURP to its equivalent off-chain cost in order to facilitate a comparison with prior work in the following manner: Instead of using a blockchain, use PBFT [18] to post messages on the bulletin board which incurs an extra $O(n)$ off-chain cost per bit.

Bivariate polynomials: Bivariate polynomials have been explored extensively in the secret-sharing literature, to build VSS protocols [17, 31], for multipartite secret-sharing [70], to achieve unconditional security [54], and to build MPC protocols [14, 38]. Prior to CHURP, few works [26, 64] have considered application of bivariate polynomials to dynamic committees, but these have been limited to passive adversaries. CHURP's novel use of dimension-switching provides security against a strong active adversary controlling $2t$ nodes during the handoff. The dimension-switching technique applies well known resharing techniques [24, 38] via bivariate polynomials to switch between full and reduced shares.

0-sharing, the technique of generating a 0-hole polynomial has been widely used for proactive security since the work of [44]. As we explain before, prior works [26, 54, 64] have naively extended these to the bivariate case leading to expensive 0-sharing protocols. Instead, CHURP applies resharing techniques [24] to build an efficient bivariate 0-sharing protocol.

CHURP's use of two sharings appears in some prior works [60, 62] (with largely differing goals and detail) where each node stores an additive share of the secret and a backup share of every other node's additive share. Proactivization is achieved by resharing the additive shares, in contrast to CHURP's approach of generating a shared polynomial explicitly which is then used to update the reduced shares. We note that adapting these techniques for use in CHURP is non-trivial, moreover, CHURP's bivariate 0-sharing protocol has other uses as well, e.g., can reduce the off-chain cost of [44].

ACKNOWLEDGEMENT

This work was funded by NSF grants CNS-1514163, CNS-1564102, and CNS-1704615, as well as ARO grant W911NF16-1-0145 and support from IC3 industry partners.

REFERENCES

[1] [n.d.]. ETH Gas Station. https://ethgasstation.info/. (Accessed on 11/13/2018).
[2] [n.d.]. Ethereum Gas Price Tracker. https://etherscan.io/gastracker.
[3] [n.d.]. Go language interface to GMP - GNU Multiprecision Library.
[4] [n.d.]. gRPC: A high performance, open-source universal RPC framework.
[5] [n.d.]. The PBC Go Wrapper. https://github.com/Nik-U/pbc.
[6] 2018. Decentralized Identity Foundation (DIF) homepage.
[7] 2018. uPort. https://www.uport.me/.
[8] Yazin Akkawi. 21 Dec. 2017. Bitcoin's Most Pressing Issue Summarized in Two Letters: UX. *Inc.* (21 Dec. 2017).
[9] Brian Armstrong. Feb. 25, 2018. Coinbase is not a wallet. https://blog.coinbase.com/coinbase-is-not-a-wallet-b5b9293ca0e7.
[10] Avi Asayag, Gad Cohen, Ido Grayevsky, Maya Leshkowitz, Ori Rottenstreich, Ronen Tamari, and David Yakira. 2018. *Helix: a scalable and fair consensus algorithm.* Technical Report. Technical report, Orbs Research.
[11] Michael Backes, Amit Datta, and Aniket Kate. 2013. Asynchronous computational VSS with reduced communication complexity. In *CT-RSA*. Springer.
[12] Joshua Baron, Karim El Defrawy, Joshua Lampkins, and Rafail Ostrovsky. 2015. Communication-optimal proactive secret sharing for dynamic groups. In *ACNS*.
[13] Mihir Bellare, Zvika Brakerski, Moni Naor, Thomas Ristenpart, Gil Segev, Hovav Shacham, and Scott Yilek. 2009. Hedged public-key encryption: How to protect against bad randomness. In *ASIACRYPT*. Springer, 232–249.
[14] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *ACM TOCS*.
[15] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. 2015. On Bitcoin as a public randomness source. *IACR ePrint Archive* 2015 (2015), 1015.
[16] Kevin D Bowers, Ari Juels, and Alina Oprea. 2009. HAIL: A high-availability and integrity layer for cloud storage. In *16th ACM CCS*.
[17] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. 2002. Asynchronous verifiable secret sharing and proactive cryptosystems. In *ACM CCS*.
[18] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine fault tolerance and proactive recovery. *ACM TOCS* (2002).
[19] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. 2019. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts. In *2019 IEEE EuroS&P*.
[20] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. 1997. A secure and optimally efficient multi-authority election scheme. *ETT* (1997).
[21] Phil Daian, Rafael Pass, and Elaine Shi. 2016. Snow White: Provably Secure Proofs of Stake. Cryptology ePrint Archive, Report 2016/919.
[22] Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam Smith. 2008. Scalable multiparty computation with nearly optimal work and resilience. In *CRYPTO*. Springer, 241–261.
[23] Yvo Desmedt and Yair Frankel. 1991. Shared generation of authenticators and signatures. In *CRYPTO*.
[24] Yvo Desmedt and Sushil Jajodia. 1997. *Redistributing secret shares to new access structures and its applications.* Technical Report.
[25] Yvo Desmedt and Kirill Morozov. 2015. Parity check based redistribution of secret shares. In *ISIT*.
[26] Shlomi Dolev, Juan Garay, Niv Gilboa, and Vladimir Kolesnikov. 2009. Swarming secrets.
[27] Michael Egorov, MacLane Wilkison, and David Nuñez. 2017. Nucypher KMS: decentralized key management system. *arXiv preprint arXiv:1707.06140* (2017).
[28] Ethereum. [n.d.]. Devp2p. https://github.com/ethereum/devp2p
[29] Ethereum. [n.d.]. Whisper. https://github.com/ethereum/wiki/wiki/Whisper
[30] Paul Feldman. 1987. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*.
[31] Pesech Feldman and Silvio Micali. 1997. An optimal probabilistic protocol for synchronous Byzantine agreement. *SIAM J. Comput.* (1997).
[32] Yair Frankel, Peter Gemmell, Philip D MacKenzie, and Moti Yung. 1997. Optimal-resilience proactive public-key cryptosystems. In *FOCS*.
[33] Yair Frankel, Peter Gemmell, Philip D MacKenzie, and Moti Yung. 1997. Proactive rsa. In *CRYPTO*.
[34] frontrun.me. [n.d.]. Visualizing Ethereum gas auctions. http://frontrun.me/.
[35] Georg Fuchsbauer. 2018. Subversion-zero-knowledge SNARKs. In *IACR International Workshop on Public Key Cryptography*. Springer, 315–347.
[36] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. 1996. Robust threshold DSS signatures. In *EUROCRYPT*.
[37] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. 1999. Secure distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT*.
[38] Rosario Gennaro, Michael O Rabin, and Tal Rabin. [n.d.]. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography.
[39] geth. [n.d.]. The maximum data size in a transaction is 32 KB. https://github.com/ethereum/go-ethereum/blob/6a3395473165866705646 6bf7573ed1c397f4750/core/tx_pool.go#L570.
[40] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *SOSP*.
[41] Jens Groth. 2010. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*.
[42] Amir Herzberg. 2009. Folklore, practice and theory of robust combiners. *Journal of Computer Security* 17, 2 (2009), 159–189.
[43] Amir Herzberg, Markus Jakobsson, Stanislław Jarecki, Hugo Krawczyk, and Moti Yung. 1997. Proactive public key and signature systems. In *ACM CCS*.
[44] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. 1995. Proactive secret sharing or: How to cope with perpetual leakage. In *CRYPTO*.
[45] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. 2010. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*.
[46] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO*.

[47] Eleftherios Kokoris-Kogias, Enis Ceyhun Alp, Sandra Deepthy Siby, Nicolas Gailly, Linus Gasser, Philipp Jovanovic, Ewa Syta, and Bryan Ford. 2018. CALYPSO: Auditable Sharing of Private Data over Blockchains. Cryptology ePrint Archive.
[48] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. 2016. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. In *IEEE S&P*.
[49] Haiyun Luo, Petros Zerfos, Jiejun Kong, Songwu Lu, and Lixia Zhang. 2002. Self-Securing Ad Hoc Wireless Networks. In *ISCC*.
[50] Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. 2019. CHURP: Dynamic-Committee Proactive Secret Sharing. https://eprint.iacr.org/2019/017.
[51] Ventzislav Nikov and Svetla Nikova. 2004. On proactive secret sharing schemes. In *International Workshop on Selected Areas in Cryptography*.
[52] John P. Njui. [n.d.]. Coinbase Custody Service Secures Major Institutional Investor Worth $20 Billion. *Ethereum World News* ([n. d.]).
[53] Mehrdad Nojoumian and Douglas R Stinson. 2013. On dealer-free dynamic threshold schemes. *Adv. in Math. of Comm.* (2013).
[54] Mehrdad Nojoumian, Douglas R Stinson, and Morgan Grainger. 2010. Unconditionally secure social secret sharing scheme. *IET information security* (2010).
[55] Rafail Ostrovsky and Moti Yung. 1991. How to withstand mobile virus attacks. In *ACM PODC*.
[56] Parity. [n.d.]. Transaction Queue. https://wiki.parity.io/Transactions-Queue.
[57] Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the blockchain protocol in asynchronous networks. In *EUROCRYPT*. Springer, 643–673.
[58] Rafael Pass and Elaine Shi. 2018. Thunderella: Blockchains with Optimistic Instant Confirmation. In *EUROCRYPT*.
[59] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*.
[60] Bartosz Przydatek and Reto Strobl. 2004. Asynchronous proactive cryptosystems without agreement. In *ASIACRYPT*. Springer, 152–169.
[61] Michael O Rabin. 1983. Randomized byzantine generals. In *FOCS*.
[62] Tal Rabin. 1998. A simplified approach to threshold and proactive RSA. In *CRYPTO*.
[63] Jeff John Roberts and Nicolas Rapp. 2017. Exclusive: Nearly 4 Million Bitcoins Lost Forever, New Study Says. http://fortune.com/2017/11/25/lost-bitcoins/
[64] Nitesh Saxena, Gene Tsudik, and Jeong Hyun Yi. 2005. Efficient node admission for short-lived mobile ad hoc networks. In *13th ICNP*.
[65] Berry Schoenmakers, Meilof Veeningen, and Niels de Vreede. 2016. Trinocchio: privacy-preserving outsourcing by distributed verifiable computation. In *ACNS*.
[66] David A Schultz, Barbara Liskov, and Moses Liskov. 2008. Mobile proactive secret sharing. In *ACM PODC*.
[67] Adi Shamir. 1979. How to share a secret. *Commun. ACM* (1979).
[68] Douglas R Stinson and Ruizhong Wei. 1999. Unconditionally secure proactive secret sharing scheme with combinatorial structures. In *SAC*.
[69] Paul Syverson, R Dingledine, and N Mathewson. 2004. Tor: The secondgeneration onion router. In *Usenix Security*.
[70] Tamir Tassa and Nira Dyn. 2009. Multipartite secret sharing by bivariate interpolation. *Journal of Cryptology* (2009).
[71] Theodore M Wong, Chenxi Wang, and Jeannette M Wing. 2002. Verifiable secret redistribution for archive systems. In *the 1st Security in Storage Workshop*.
[72] Fan Zhang, Philip Daian, Gabriel Kaptchuk, Iddo Bentov, Ian Miers, and Ari Juels. 2018. Paralysis Proofs: Secure Access-Structure Updates for Cryptocurrencies and More. Cryptology ePrint Archive, Report 2018/096.
[73] Lidong Zhou, Fred B Schneider, and Robbert Van Renesse. 2005. APSS: Proactive secret sharing in asynchronous systems. *ACM TISSEC* (2005).
[74] Guy Zyskind, Oz Nathan, et al. 2015. Decentralizing privacy: Using blockchain to protect personal data. In *Security and Privacy Workshops*.

A SECURITY PROOF FOR Opt−CHURP

Recall that a protocol for dynamic-committee proactive secret sharing satisfies secrecy and integrity. We prove secrecy first.

Secrecy. We consider the handoff protocol of one epoch first. As described in Section 5.3, Opt-CHURP consists of three phases: Opt-ShareReduce, Opt-Proactivize and Opt-ShareDist. Other than the public inputs, the information obtained by the adversary $\mathcal{A}$ is:

Opt-ShareReduce:

- For all corrupt $\mathcal{U}_j$ in the previous handoff, reduced share $B(x,j)$.
- For all corrupt nodes C_i in the old committee, $\left\{B(i,j),W_{B(i,j)}\right\}_{j\in[2t+1]}$ (full share $B(i,y)$).
- For all corrupt $\mathcal{U}'_j$ in the new committee selected to participate in the handoff, $\left\{B(i,j),W_{B(i,j)}\right\}_{i\in[2t+1]}$ (reduced share $B(x,j)$).

Opt-Proactivize:

- For all corrupt nodes $\mathcal{U}'_j$, s_j and $Q(x,j)=R_j(x)$.
- For all corrupt nodes C'_i in the new committee, H_j and $\left\{g^{s_j},C_{Z_j},W_{Z_j(0)},C_{B'(x,j)}\right\}$.

Opt-ShareDist:

- For all corrupt C'_i in the new committee, $\left\{B'(i,j),W'_{B(i,j)}\right\}_{j\in[2t+1]}$.

The information above assumes the secrecy of our bivariate 0-sharing protocol, which we explained in the main body. In addition, note that the public information posted on chain are all commitments of the polynomials. By the hiding property of the commitment scheme based on the discrete log assumption, the PPT $\mathcal{A}$ learns no extra information from these commitments. To prove secrecy, we have the following lemmas.

Lemma 2. *If $\mathcal{A}$ corrupts no more than t nodes in the old committe node, and no more than t nodes in $\mathcal{U}'$, the information received by $\mathcal{A}$ in* Opt-ShareReduce *is random and independent of the secret s.*

Proof. This is implied by the degree of the bivariate polynomial $B(x,y)$. In the worst case when all t corrupted nodes are in $\mathcal{U}$ and $\mathcal{U}'$, $\mathcal{A}$ learns $2t$ reduced shares $B(x,j)$ and t full shares $B(i,y)$. For a $\langle t,2t\rangle$-bivariate polynomial, any t shares of $B(i,y)$ and $2t$ shares of $B(x,j)$ are random and independent of $s=B(0,0)$.

Moreover, based on the discrete-log assumption, the proofs $W_{B(i,j)}$ are computationally zero-knowledge by the KZG scheme, and the PPT adversary cannot learn additional information from them. □

Lemma 3. *Given a bivariate 0-sharing scheme with secrecy and integrity, if at least one node is honest in* Opt-Proactivize, *$Q(x,y)$ is randomly generated.*

Proof. Any $2t+1$ degree t univariate polynomials $Q(x,j)$ uniquely define a $\langle t,2t\rangle$-bivariate polynomial. Therefore, as long as one node is honest and generates a random degree t polynomial, $Q(x,y)$ is randomly generated to mask $B(x,y)$.

Similar to the proof above, the hashes and commitments do not leak additional information to a PPT adversary $\mathcal{A}$. □

Lemma 4. *If $\mathcal{A}$ corrupts no more than t nodes in the new committee C', the information received by $\mathcal{A}$ in* Opt-ShareDist *is random and independent of the secret s.*

Proof. By Lemma 2, $Q(x,y)$ is randomly generated, thus $B'(x,y)=B(x,y)+Q(x,y)$ is independent of $B(x,y)$. Regardless of the number of nodes corrupted by $\mathcal{A}$ in $\mathcal{U}'$, $\mathcal{A}$ receives no more than t out of n' shares of $B'(i,y)$ in Opt-ShareDist. As the degree of $B'(x,y)$ is $\langle t,2t\rangle$ and is independent of $B(x,y)$, these shares are random and independent of s. Again, the proofs in the second part do not leak additional information. □

By Lemma 2, 3 and 4, $\mathcal{A}$ does not learn any information about s in consecutive epochs. The secrecy of the scheme follows by induction.

Integrity. For integrity, we have the following lemmas.

Lemma 5. *After* Opt-ShareReduce, *at least $t+1$ honest nodes $\mathcal{U}'_j$ can successfully reconstruct $B(x,j)$.*

KZG **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.

Figure 8: Protocols of KZG commitment scheme.

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

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

2: **Output:** Each node $\mathcal{U}_j$ outputs a share $s_j = P(j)$ for randomly

3: generated degree-$2t$ polynomial $P(y)$ with $P(0) = 0$

4: node $\mathcal{U}_j$

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

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

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

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

Figure 9: $(2t, 2t+1)$-UnivariateZeroShare **between** $2t+1$ **nodes. A 0-hole univariate polynomial** P **of degree-**$2t$ **is generated.**

Proof. As the number of nodes in the old committee $n \geq 2t+1$, each node $\mathcal{U}'_j$ receives at least $t+1$ correct shares of $B(i,j)$. As the degree on the first variable of $B(x,y)$ is t, $\mathcal{U}'_j$ can reconstruct $B(x,j)$ successfully. Finally, as the number of nodes in $\mathcal{U}'$ is $2t+1$, there are at least $t+1$ honest nodes. □

Lemma 6. *Assuming the correctness of the bivariate 0-sharing scheme, after* Opt-Proactivize*, either honest nodes* $\mathcal{U}'_j$ *hold the correct shares of* $B'(x,j)$ *such that* $B'(0,0) = B(0,0) = s$ *and their commitments* $C_{B'(x,j)}$ *are on-chain, or at least* $t+1$ *honest nodes in* $\mathcal{C}'$ *output fail.*

Proof. By line 15 in Figure 12, $\{g^{s_j}, C_{Z_j}, W_{Z_j(0)}, C_{B'(x,j)}\}$ is consistent with the hash H_j posted on chain by $\mathcal{U}'_j$. If C_{Z_j} is not a univariate polynomial with constant term 0, by line 16, VerifyEval outputs false and C'_i outputs fail by the soundness of KZG. Otherwise, by the second check of line 16, $C_{B'(x,j)}$ is the commitment of a polynomial $B'(x,j)$ with constant term $B(x,j) + s_j$. Finally, by the check of line 17, by the discrete-log assumption, $\sum_{j=1}^{2t+1} s_j \lambda_j^{2t} = 0$. Therefore, $B'(0,0) = B(0,0)$ because of the property of Lagrange coefficients. □

By Lemma 5 and 6, if Opt-ShareReduce and Opt-Proactivize do not fail, all nodes $\mathcal{U}'_j$ hold the correct shares of $B'(x,j)$ such that $B'(0,0) = B(0,0) = s$ and their commitments $C_{B'(x,j)}$ are on the chain. In Opt-ShareDist, each node C'_i receives $2t+1$ shares of $B'(i,j)$ from all $\mathcal{U}'_j$s. By the soundness of the KZG scheme, if any of these shares is corrupt, VerifyEval rejects, and honest nodes in $\mathcal{C}'$ output fail. Otherwise, with $2t+1$ correct shares of $B'(i,j)$, C'_i can successfully reconstruct $B'(i,y)$, which completes the proof of integrity.

B APPLICATIONS IN DECENTRALIZED SYSTEMS

Secret sharing finds use in innumerable applications involving cryptographic secrets, including secure multi-party computation (MPC) [14], threshold cryptography [23], Byzantine agreement [61], and cryptocurrency custody [9].

(t, n)-BivariateZeroShare

1: **Input:** t, n, set of nodes $\{C_i\}_{i\in[n]}$ $(2t < n)$

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

3: Order $\{C_i\}_{i\in[n]}$ based on lexicographic order of their public keys

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

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

6: node $\mathcal{U}_j$:

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

8: Send a point $R_j(i)$ to node C_i for each $i \in [n]$

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

10: node C_i:

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

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

13: Output share $Q(i,y)$

Figure 10: (t, n)-BivariateZeroShare **between** n **nodes. A 0-hole bivariate polynomial** Q **of degree-**$\langle t, 2t\rangle$ **is generated.**

Opt-ShareReduce

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

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

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

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

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

6: node 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 $\mathsf{VerifyEval}(C_{B(x,j)}, i, B(i,j), W_{B(i,j)})$

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

Figure 11: Opt-ShareReduce **between the committees** $\mathcal{C}$ **and** $\mathcal{C}'$**.**

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 [63] or centralized key management [9, 52] that defeats the main purpose of blockchain systems.

Second, *blockchain objects cannot keep private state.* This fact limits the 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 using CHURP:

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 [8], 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.

Decentralized identity. Initiatives such as the Decentralized Identity Foundation [6] and uPort [7] envision an ecosystem in which

Opt-Proactivize

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

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

Output: $\mathcal{U}'_j$ outputs success and $B'(x,j)$ for a degree-$\langle t, 2t\rangle$ bivariate polynomial $B'(x,y)$ with $B'(0,0) = B(0,0)$ (or) fail

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

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 $\left\{g^{s_j}, C_{Z_j}, W_{Z_j(0)}, C_{B'(x,j)}\right\}$ off-chain to all nodes in 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 C'_i:

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

Figure 12: Opt-Proactivize **updates the reduced shares.**

Opt-ShareDist

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

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

Output: $\forall i \in [n']$, 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 $\left\{B'(i,j), W'_{B(i,j)}\right\}$ to C'_i where $W'_{B(i,j)} = \mathsf{CreateWitness}(B'(x,j), i)$

node C'_i:

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

  $\forall j \in [2t+1]$, 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

Figure 13: Opt-ShareDist **uses the updated reduced shares to distribute full shares in C'.**

users control their identities and data by means of private keys. The same techniques used in the cryptocurrency case for private-key management would similarly apply to assets such as identities.

Smart-contract attestations. Committee management of smart-contract private keys could also enable *digital signing* by smart contracts. Committee members would 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.

Simplified Committee-based consensus for light clients. A number of consensus schemes, e.g., proof-of-stake protocols [21, 46], aim to achieve scalability by delegating consensus to committees. As these committees change over time, verifying the blocks they sign requires awareness of their identities. By instead maintaining its key pair, a committee could make it easier for light clients to verify blockchains.

Secure multiparty computation (MPC) for smart contracts. More generally, dynamic-committee secret sharing would enable decentralized secure MPC by smart contracts, effectively endowing them with confidential storage and computation functionalities, as envisioned in, e.g., [19, 74].

C CHURP PESSIMISTIC PATHS

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

C.1 Exp-CHURP-A

This path is invoked when a failure occurs in Opt-CHURP. As mentioned before, the pessimistic paths use on-chain communication only. The first phase is the same as Opt-ShareReduce, and is not re-executed if Opt-ShareReduce ends successfully.

In Exp-Proactivize, we use a different zero-sharing protocol, allowing honest parties to avoid re-execution of the protocol in case of corruption — they can simply discard the shares generated by the adversarial nodes. Messages are encrypted under the receiver's public key and posted on-chain, so that a verifiable accusation can be performed in case of a corruption.

If any adversary in $\mathcal{U}'$ is expelled in this phase, we ask members in the old committee to publish the shares and witnesses sent to the adversarial nodes during Opt-ShareReduce on-chain. Thus, all honest parties have access to reduced shares that belong to adversarial nodes, which allows them to reconstruct the full shares in the next phase.

In Exp-ShareDist, to allow identification of malicious nodes, members post all messages on-chain in contrast to the optimistic path. Exp-Proactivize and Exp-ShareDist are presented in Figure 14 and 15. The overall on-chain complexity of Exp-CHURP-A is $O(n^2)$.

C.2 State Verification Details

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 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 $\{g^{B'(i,j)}\}$ $\forall j \in [2t+1]$, and their witness $w'_{j,i}$ to the KZG polynomial commitments $C_{B'(x,j)}$ on-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: $\mathsf{VerifyEvalExp}(C_{B'(x,j)}, i, g^{B'(i,j)}, W_{j,i})$ for $j \in [2t+1]$. (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: $\{\mathsf{True}, \mathsf{False}\} \leftarrow \mathsf{VerifyEvalExp}(C_\phi, i, g^{\phi(i)}, W_i)$.)

Exp-Proactivize

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

Input: Set of $2t+1$ nodes $\{\mathcal{U}'_j\}_{j\in[2t+1]}$. Each node $\mathcal{U}'_j$ is given $B(x,j)$

Output: $\mathcal{U}'_j$ outputs $B'(x,j)$ for a degree-$\langle t,2t\rangle$ bivariate polynomial $B'(x,y)$ with $B'(0,0)=B(0,0)$

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

node $\mathcal{U}'_i$:

Generate $\{s_{ij}\}_{j\in[2t+1]}$ that form a 0-sharing i.e., $\sum_{j=1}^{2t+1}\lambda_j^{2t}s_{ij}=0$.

Publish $\{g^{s_{ij}}\}_{j\in[2t+1]}$, $\{\mathsf{Enc}_{\mathsf{pk}_j}[s_{ij}]\}_{j\in[2t+1]}$ and zk proofs of correctness of the encryptions on-chain.

node $\mathcal{U}'_j$:

Decrypt $\{\mathsf{Enc}_{\mathsf{pk}_j}[s_{ij}]\}$ from node i and verify s_{ij} using $g^{s_{ij}}$ on-chain.

node $\mathcal{U}'_j$:

If any adversarial node i is detected in step 9, add it to $\mathcal{U}'_{corrupted}$, and publish s_{ji}.

Set $s_j=\sum_{i\in\mathcal{U}'\setminus\mathcal{U}'_{corrupted}}s_{ij}$.

Execute step 7-9, 11-12 of Opt-Proactivize in Figure 12, with messages posted on the chain in step 12.

node C'_i

Execute step 16 of Opt-Proactivize in Figure 12. If it outputs `fail`, add j to $\mathcal{U}'_{corrupted}$. Nodes in $\mathcal{U}'$ discard shares by executing step 12 again.

node C_i:

For all malicious nodes j detected in step 9 and 15, publish point and witness $\{B(i,j),w_{i,j}\}$ on-chain.

Figure 14: Exp-Proactivize **protocol.**

If the checks above pass, all members validate Z_i, Z_i^{rnd} as: $Z_i=\prod_{j=1}^{2t+1}(g^{B'(i,j)})^{\lambda_j^{2t}}$, $Z_i^{rnd}=\prod_{j=1}^{2t+1}(g^{B'(i,j)})^{r_j\lambda_j^{2t}}$.

Exp-ShareDist

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

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

Output: $\forall i\in[n']$, 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 $w'_{i,j}=\mathsf{CreateWitness}(B'(x,j),i)$. Also publish zk proofs of correctness of the encryption.

node 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 fail, add j to $\mathcal{U}'_{corrupted}$

node 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 and discard shares by executing step 12 in Fig. 14.

node 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 $B'(i,y)$

Figure 15: Exp-ShareDist **protocol.**

If any of the checks above fail, 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.3 Exp-CHURP-B

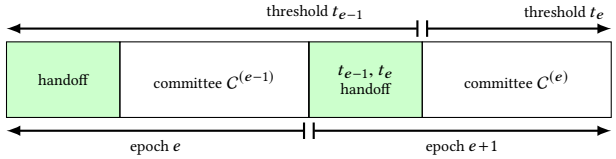
This pessimistic path is taken only after a detection of breach in the underlying assumptions of the KZG scheme.

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 14 and step 5,8,12 in Figure 15). Exp-CHURP-B incurs $O(n^2)$ on-chain cost, assuming $n>3t$. Due to the space limit, we omit the full protocol of Exp-CHURP-B.

D CHANGING THE THRESHOLD

D.1 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 $C^{(e-1)}$ and $C^{(e)}$, 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 [66], increasing the threshold takes two steps: first, a handoff is executed between $C^{(e-1)}$ and $C^{(e)}$ 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.



Specifically, 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, i.e., given a commitment C_ϕ, it can be publicly verified that ϕ is at most d-degree. 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 a KZG commitment can guarantee that hidden polynomials are of degree $\le t$, which is critical to the correctness of shares. To support dynamic thresholds up to $t_{\max}$, we

KZG extended with degree verification

1) (sk, pk) ← Keygen(1^λ, q): Select a bilinear group $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ ← BilGen(1^λ), $q+1$ group elements $\{\alpha_i\}_{i\in[q]}$ and s randomly in $\mathbb{Z}_p^*$. Set 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) $C_\phi \leftarrow$ Commit($\phi(x)$, pk): Let $d = \deg(\phi)$. Compute $C_\phi = (d, g^{\phi(s)}, g^{\alpha_d \phi(s)})$ using pk_0 and pk_d.

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

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

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

Figure 16: KZG [45] **extended with degree verification.**

extend KZG as specified in Fig. 16 and run the trusted setup with $D = t_{\max}$. Our extension relies on the q-PKE [41] assumption.

D.2 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 $C = C^{(e-1)}$ and $C' = C^{(e)} \cup \mathcal{V}$, 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 C' incorporate $\mathcal{V}$'s state and restore the invariants. The handoff protocol is outlined as follows.

Decreasing the threshold

1) Choose a subset $\mathcal{U} \subseteq C'$ 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, 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 $j \in C'$ a point $RS_i^{(e)}(j)$. The full share of each node $j \in C'$ 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, 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.

E POINT-TO-POINT TECHNIQUE DETAILS

E.0.1 Choosing overwrite rate k. An optimal strategy is to overwrite as many times as possible. Ethereum, though, imposes a constraint on overwriting: the sender must raise the transaction fee by at least a minimum fraction ρ. (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. In the worst case, where $\rho = 12.5\%$, the optimal strategy is to overwrite $k = 7$ times, yielding average cost $0.29\times\frac{f}{|m|}$ per byte, about 70% less than without overwriting. Moreover, if the first $k-1$ transactions have $|m|$ bytes of data and the last one empty, the average cost is down to $\frac{f_0\times\rho^k}{|m|\times k}$ per byte.

E.0.2 Experiments. We validate our ideas experimentally on the Ethereum mainnet. The sender and receiver are full nodes connected to the Ethereum P2P network and 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.

In our experiments, we rewrite $k = 7$ times. Each of the first 7 transactions contains 31KB of data and the 8th is empty. A total of ≈ 100MB data is transmitted in 4,200 transactions, in about 1 hour. Table 3 summarizes the results, which we now discuss.

Bandwidth: DoS prevention measures and network latency in Ethereum cause overly frequent overwritten transactions to drop. 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 [39]. While this suffices for CHURP, we belived more engineering would yield higher bandwidth. Studies of blockchain arbitrage [34] show that arbitrageurs can overwrite transactions in hundreds of milliseconds.

Message-transmission cost: Transaction costs for message delivery are extremely low: $0.06 per MB 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.

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 [56], 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 ≈ 9% loss rate.

E.0.3 Comparison to on-chain communication. For comparison, we estimate the same metrics for on-chain communication, i.e. using the Ethereum blockchain. The results are summarized in Table 3.

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 MB is estimated as that of sending 32 transactions with 32KB data in each, assuming an exchange rate of 1ETH = $200. The latency depends on the gas price and the network condition. A lower latency requires a higher gas price and thus a higher transmission cost. We used [1] for our estimation. The tradeoff between latency and message transmission cost is shown in Fig. 7.