Practical Mempool Privacy via One-time Setup Batched Threshold Encryption

Arka Rai Choudhuri [*]
Nexus

Sanjam Garg [†]
UC Berkeley

Guru-Vamsi Policharla [‡]
UC Berkeley

Mingyuan Wang [§]
NYU Shanghai

Abstract

An important consideration with the growth of the DeFi ecosystem is the protection of clients who submit transactions to the system. As it currently stands, the public visibility of these transactions in the memory pool (mempool) makes them susceptible to market manipulations such as frontrunning and backrunning. More broadly, for various reasons—ranging from avoiding market manipulation to including time-sensitive information in their transactions—clients may want the contents of their transactions to remain private until they are executed, i.e. they have *pending transaction privacy*. Therefore, *mempool privacy* is becoming an increasingly important feature as DeFi applications continue to spread.

We construct the first *practical* mempool privacy scheme that uses a *one-time* DKG setup for n decryption servers. Our scheme ensures the strong privacy requirement by not only hiding the transactions until they are decrypted but also guaranteeing privacy for transactions that were not selected in the epoch (*pending transaction privacy*). For each epoch (or block), clients can encrypt their transactions so that, once B (encrypted) transactions are selected for the epoch, they can be decrypted by each decryption server while communicating only $O(1)$ information.

Our result improves upon the best-known prior works, which either: (i) require an expensive initial setup involving a (special purpose) multiparty computation protocol executed by the n decryption servers, along with an additional *per-epoch* setup; (ii) require each decryption server to communicate $O(B)$ information; or (iii) do not guarantee pending transaction privacy.

We implement our scheme and find that transactions can be encrypted in approximately 8.5 ms, independent of committee size, and the communication required to decrypt an entire batch of transactions is 48 bytes per party, independent of the number of transactions. If deployed on Ethereum, which processes close to 500 transactions per block, it takes close to 3.2 s for each committee member to compute a partial decryption and 3.0 s to decrypt all transactions for a block in single-threaded mode. Compared to prior work, which had an expensive setup phase per epoch, we incur $< 2\times$ overhead in the worst case. On some metrics such as partial decryptions size, we actually fare better.

[*] arkarai.choudhuri@gmail.com. Part of this work was done while the author was a postdoc at NTT Research.
[†] sanjamg@berkeley.edu
[‡] guruvamsip@berkeley.edu
[§] mingyuan.wang@nyu.edu. Part of this work was done while the author was a postdoc at UC Berkeley.

1 Introduction

Decentralized Finance or DeFi is a financial ecosystem that leverages blockchains to provide various financial services. The rapid growth of DeFi is best illustrated by the amount of capital that has made its way into the ecosystem, with a peak value of over $250 billion in 2021. As the ecosystem evolves and grows, it is important that the system implements protections for clients of this system to ensure they are not taken advantage of. Unfortunately, as we illustrate below, there are several instances where such systems fall short.

Consider a scenario of a client interacting with the system and simply wants to submit a transaction. For this transaction to be considered complete by the system, it must be "executed" by the blockchain. To facilitate this, clients submit their transaction to a common *memory pool*, commonly known as *mempool*. It is then the job of the miners (or builders) not only to select from the mempool the transactions that get executed, but also the order in which the transactions are executed. In an ideal world, one would like every transaction to be treated "fairly", i.e. the details of the transaction would not influence whether it is selected or the order in which it is executed. Ideally, miners/builders should be "blind" to the transaction details when processing them.

Unfortunately, the large financial incentives in these systems make reality far from the desired ideal world. Miners exploit the information asymmetry available to them to frontrun, backrun, or sandwich client transactions to extract additional value from them - termed MEV or *Miner Extractable Value* [DGK+20]. While the miners benefit financially from this asymmetry of information, it is at the expense of the clients. This has been well documented both from a miner perspective in ways to benefit the most from this system (perhaps rationally) [EMC19, ZQC+21, WCDW21, ZQT+21, AEC21], and also about the broad problems caused by the ability of the miners to exploit such a system [GKW+16, QZG21, TC+21, JSSW21, CJW22]. In the most extreme case, miners can effectively censor transactions from certain clients, an undesirable outcome for the system.

Beyond simple transactions, clients may also submit updates to their existing smart contracts within their "transaction." Such updates may be crucial if, for instance, a client discovers a bug in their smart contract that causes financial losses. In such a case, they would want to update the smart contract before others in the system become aware of the issue. However, such an update might expose the bug, allowing miners to exploit it before the transaction containing the update is executed. These vulnerabilities have led to active efforts [BO22, PNS23, MS22, KLJD23, RK23, Shu21] aimed at addressing privacy concerns surrounding transactions in the mempool.[1]

A natural cryptographic approach to reinstate the "blindness" of miners to the contents of the transactions is by having clients encrypt the contents of their transactions before they submit the transactions to the mempool. In this scenario, when the miners have to pick and order transactions, they must do so from an "encrypted" mempool, and thereby without knowledge of the contents of the transaction.[2] But for such a system to be of continued use, the transactions need to be executed *in the clear*. Requiring clients to return and decrypt their own transactions is undesirable for several reasons: (i) it requires the clients to be online more frequently for the system to function; and (ii) a misbehaving client may simply refuse to decrypt their transaction after it has learned the contents of the other transactions.

Instead, we want clients to be able to encrypt directly to the system, allowing the system to decrypt selected transactions without requiring further input from the clients. To distribute trust across the system, we can envision it as consisting of a set of decryption servers. An ideal system implementing this approach would proceed as follows: (1) The decryption servers run a one-time lightweight setup to generate a public key for the system; (2) Clients encrypt their transactions using the public key; and (3) Once B transactions for the block have been selected, each decryption server broadcasts a *partial decryption* of the ciphertexts, enabling anyone in the system to decrypt the B selected transactions. Decryption should be possible as long as a sufficient number of decryption servers broadcast their respective *partial decryptions*.

[1] We note that if the cost of transaction privacy is deemed too high, our system can be modified to support both encrypted and unencrypted transactions. But for this work, we will stick to all transactions that are encrypted.

[2] Note that we are not demanding any "fair" ordering of the encrypted transactions, only that the ordering does not consider the transaction contents.

For this scheme to be practical, it must be concretely efficient: the setup should be lightweight, and the size of each server's partial decryption should be constant, *independent of the number of ciphertexts* B. Intuitively, the ciphertexts should: (1) provide no information about their underlying contents until they are decrypted; and (2) continue to reveal no information if the ciphertext was *not included in the block*. The latter security property was formalized recently in [CGPP24] as *pending transaction privacy*. It is not hard to see that in the context of our earlier examples, it is critical that pending transaction privacy is maintained.

Given the importance of the problem, several works have made progress toward constructing a solution that achieves both mempool privacy and the stated efficiency goals [ElG86, CG99, BO22, DHMW23, Shu21, CGPP24]. However, as we shall discuss shortly, none of these works are able to simultaneously achieve all of the desired properties – some fail to meet certain efficiency goals, while others crucially fail to provide pending transaction privacy.

1.1 Our Contribution

In this work, we present the first *concretely efficient* construction that guarantees mempool privacy and simultaneously achieves the following efficiency requirement.

1. Only requires a *single* execution of a distributed key generation (DKG) protocol.
2. Uses $O(1)$ communication per decryption server to decrypt a batch of B transactions.
3. Ensures *pending transaction privacy*.
4. Has $O(1)$ sized ciphertexts.

We created a Rust crate for our scheme and benchmarked it. It takes < 8.5 ms to encrypt a ciphertext and adds ≈ 466 bytes to the transaction size. For an Ethereum deployment, which processes ≈ 500 transactions per block, it would take close to 3.2 s to compute a partial decryption and 3.0 s to reconstruct all messages in single-threaded mode. We refer the readers to Section 6 for more details on the concrete efficiency of our scheme.

1.2 Related Works

The de facto approach to the mempool privacy problem is by the use of *threshold public key encryption* [DF90, Fra90, DDFY94]. In such an encryption scheme there is a single public key, and the decryption servers hold shares of the decryption key such that a large fraction of them are required to decrypt any ciphertext encrypted to the public key. In the context of mempool privacy, clients can encrypt to the public key, and once the selected ciphertexts are determined, the decryption servers run the threshold decryption algorithm to only decrypt the selected ciphertexts. While the decryption servers only need to execute a DKG as a part of the setup, the total communication sent by the n decryption servers in order to decrypt B encrypted transactions is $O(nB)$.

In order to reduce the total communication during decryption, works such as [DHMW23, MGZ22] and deployed systems such as [Shu21] introduce fine-grained encryption schemes where the clients can encrypt to the epoch (or block) they want their transaction to be added to, and the total communication to decrypt *all* such ciphertexts is $O(n)$. In particular, this means that this information can be used to also decrypt ciphertexts that were not selected to be in the block, thereby violating the crucial requirement of *pending transaction privacy*.

Most closely related to our work is a recent scheme [CGPP24], which formalizes the security properties needed to obtain a meaningful notion of mempool privacy. They further define, and construct, a cryptographic primitive they term *batched threshold decryption* that suffices to obtain mempool privacy. While the scheme in [CGPP24] has total decryption communication $O(n)$ and guarantees pending transaction privacy, the biggest limitation is the (epoch) setup. In particular, the n decryption servers must run an initial MPC for setup, and then for every epoch (block), they must additionally run an epoch setup. Although one could

design an optimized, special purpose MPC protocol, [CGPP24] only provided estimates using generic techniques for honest majority with abort MPC protocols. In contrast our work only needs a *one-time* DKG setup at the beginning of the protocol which been extensively studied [Sch99, GHL22, Gro21, KMM+23, CD24, Kat24], simpler and concretely much more efficient than running a full-fledged MPC protocol. The above described comparisons are summarized in Table 1.

We note that while heavy cryptographic machinery like general purpose witness encryption [GGSW13] or indistinguishability obfuscation [BGI+01, ADM+24] can be used to derive a solution to mempool privacy, the inefficiency of known constructions limits their applicability towards constructing concretely efficient schemes. Likewise, timelock encryption [BBBF18] is also considered as a possible solution to mempool privacy, but does not satisfy pending transaction privacy in addition to requiring wasteful computation to perform decryption of the encrypted transactions.

Finally, while our focus in this work is more broadly mempool privacy, we note that for the specific case of reducing MEV, non-cryptographic approaches have also been considered [KDL+21, KZGJ20, Kur20, MDF22, HW22, CIMI+21, BDF21, ZQG21]. These broadly aim to either limit how transactions can be re-ordered, or aim to provide game-theoretic defenses with both approaches resulting in reduced MEV. We refer the reader to [CGPP24] for a more detailed exposition of these non-cryptographic approaches.

Scheme	Comm.	Pending Tx Privacy	Setup
[ElG86, BO22, PNS23]	$O(nB)$	✓	DKG
[DHMW23, MGZ22, Shu21]	$O(n)$	✗	DKG
[CGPP24]	$O(n)$	✓	MPC + Epoch Setup
This Work	$O(n)$	✓	DKG

Table 1: A comparison of this work against prior work in terms of communication to decrypt a batch of B transactions and whether the scheme preserves the privacy of transactions that were submitted but may not have been included in the batch. DKG refers to a distributed key generation protocol and MPC refers to a (more expensive) multi-party computation protocol used during the setup phase and Epoch Setup refers to a per-epoch setup that can be carried out well before the actual epoch takes place.

2 Technical Overview

This section presents the main ideas behind our construction. We start by recalling our objective.

Our Dream Goal. Assuming a lightweight one-time setup (e.g., standard DKG setup) among a committee, we wish to achieve the following. For each epoch, users should be able independently (from other users) sample a ciphertext encrypting their message. Among the ciphertexts from all users (say, $\mathsf{ct}_1, \ldots, \mathsf{ct}_m$) in a given epoch, once the committee agrees on a subset $\mathcal{S} \subseteq [m]$ of size $\leq B$ to decrypt, the committee sends a constant-size partial decryption. Given more than a threshold T number of partial decryptions, one may decrypt all the ciphertext in $\mathcal{S}$. However, the ciphertexts outside $\mathcal{S}$ should remain hidden. As shown in [CGPP24], constructing such a primitive suffices to obtain mempool privacy scheme. Thus, for the rest of this overview, we focus on the underlying ideas for building such a primitive. Additionally, as referenced in our discussion previously (Section 1.2), all prior works fail to achieve this dream goal in one way or another.

Witness Encryption for PPEs. To facilitate presenting our ideas, let us first introduce some minimal notations and a generic way of building witness encryption schemes [GGSW13] from pairing product equations

(PPEs).[3]

Suppose we have a pairing-friendly group with pairing operation $e(\cdot,\cdot)$ and generators g, h, g_{T}. Consider the following set of pairing product equations

$$e(g^A, h^x) = g_{\mathsf{T}}^b,^{[4]} \tag{1}$$

where A is a $u \times v$ matrix of field elements, and b and x are vectors of dimension u and v, respectively. Here, we think of g^A and g_{T}^b as *public* group elements and h^x as the *secret* group elements. Importantly this facilitates building a *witness encryption* scheme for the statement,

I hold the secret group elements h^x that will
pass the pairing product equations $e(g^A, \cdot) = g_{\mathsf{T}}^b$.

Specifically, to encrypt a message m for the above linear system, one samples a u-dimension random vector of field elements α and computes the ciphertext as

$$\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2) = \left(g^{\alpha \cdot A}, H\left(g_{\mathsf{T}}{}^{\langle \alpha, b \rangle}\right) \oplus m\right).^{[5]}$$

Here $H(\cdot)$ is a random oracle. Note that the public group elements g^A and g_{T}^b are treated as the statement, and anyone in possession of the witness (secret group elements) h^x can decrypt by computing m as $\mathsf{ct}_2 \oplus H\left(e(\mathsf{ct}_1, h^x)\right)$. For security, one can prove that the ciphertext is computationally hidden as long as the witness h^x is hard to compute (see Theorem 2).

Several works (e.g., [KMW23, CGPP24, GKPW24]) show how WE for PPEs can be used to realize powerful applications. In these works, as in ours, the core contribution lies in casting the task at hand in the form of appropriate PPEs satisfying the constraints above. This requires "special gymnastics" to get the problem in the right format. We next explain the challenges involved in this for our work.

First Attempt. Let us start with a simple idea by applying WE for PPEs to BLS signature [BLS01]. Recall that, in BLS signature, the public key is $\mathsf{pk} = g^{\mathsf{sk}}$ for a signing key sk, and the signature σ for a tag tg is computed as $\sigma = H(\mathsf{tg})^{\mathsf{sk}}$. The signature verification check is

$$e(\sigma, h) = e(H(\mathsf{tg}), \mathsf{pk})$$

conforming to a PPE system as described in (1). Henceforth, one may apply WE for PPEs to build an encryption scheme.[6] To extend this to multiple users while ensuring succinct batch decryption, one naturally asks each user to sample their respective ciphertext under the *same tag* tg.[7] In this way, the committee could compute a (threshold) signature σ of tg as the witness to enable batch decryption of *all* ciphertext encrypted under tg. This is essentially the scheme of McFly [DHMW23].

However, this simple approach above does not provide *pending transaction privacy*. Indeed, the (threshold) BLS allows for decryption of all ciphertext encrypted under tg, not necessarily the B selected transaction. Therefore, the challenge remains how one designs a PPE system that allows succinct decryption of *only the selected transactions*.

[3] This general tool was observed in the literature under different names. For example, [BC16] observed this in the hash proof system context; [GKPW24] observed this from the witness encryption perspective (which is what we present here). Moreover, this tool is implicitly used in a less general manner in many existing works (e.g., [BF01, GS18, BL18, KMW23]).

[4] For a matrix/vector A of field elements, we use g^A for a matrix/vector of group elements defined in a natural way. Moreover, for two matrices A, B of group elements, $e(A, B)$ stands for the standard matrix multiplication with multiplication operation replaced by pairing operation.

[5] Note that ct_1 is a *vector* of group elements.

[6] This is exactly the Boneh-Franklin IBE scheme [BF01].

[7] For example, this tag could be the epoch ID.

Polynomial Encoding of a Set. To get around this challenge, [CGPP24] proposed using *KZG polynomial commitment* [KZG10]. Let us recall KZG polynomial commitment first. To enable committing to a degree $(B-1)$ polynomial, the KZG common reference string consists of the tuple $(g, g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^{B-1}})$, and the commitment to a polynomial p is $\mathsf{com} = g^{\mathsf{p}(\tau)}$.[8] Given a pair $(x, \mathsf{tg}) \in \mathbb{F} \times \mathbb{F}$ such that $\mathsf{tg} = \mathsf{p}(x)$, the proof of opening $\pi = g^{\mathsf{q}(\tau)}$ where $\mathsf{q}(X)$ is a polynomial such that $\mathsf{q}(X) = (\mathsf{p}(X) - \mathsf{tg})/(X - x)$. The verification equation is thus given by

$$e(\pi, h^{\tau - x}) = e(\mathsf{com}/g^{\mathsf{tg}}, h).$$

An observant reader may note that the above verification equation is again compatible with (1).[9] Indeed, in [CGPP24] the authors was able to leverage this to build a batch threshold decryption in the following way. For each epoch, a random group element $\mathsf{com} = g^y$ will be sampled as the commitment and the committee shall hold secret shares of y. Now, each user will sample its own (x_i, tg_i) and compute a witness encryption for the KZG opening verification check for the aforementioned commitment com,

$$e(\pi_i, h^{\tau - x_i}) = e(\mathsf{com}/g^{\mathsf{tg}_i}, h).$$

Now, suppose a set S of transaction is selected to be decrypted. For succinct batch decryption, instead of asking the committee to compute a opening proof π_i for each ciphertext in S, the committee shall leverage the *equivocal* property of the commitment com[10] to explain it as the polynomial commitment of the polynomial $f(x)$ that interpolates all $\{(x_i, \mathsf{tg}_i)\}_{i \in S}$. Note that if $f(x)$ becomes public, any one can generate the opening proof and, henceforth, decrypt the ciphertexts in S. The crucial point here is that to communicate $f(x)$, the committee does not need to send the entire $f(x)$, but only to publish an arbitrary point $(x^*, f(x^*))$ on $f(x)$.[11]

While this idea gives an elegant solution, it unfortunately comes with several undesirable features.

- It necessitates a *per-epoch* setup — the committee needs to collectively sample a random group element com for every epoch. In other words, com is not a *reusable* setup.
- Equivocating $\mathsf{com} = g^y$ to a chosen polynomial requires the committee to hold the trapdoor τ. In fact, the computation required for equivocation is a high-degree computation in τ. To enable a light-weight decryption process, [CGPP24] asks for an expensive initial setup such that committee holds secret shares of not only τ, but also $\tau^2, \tau^3, \ldots$ This setup requires an expensive tailor-made MPC protocol. For instance, a simple DKG would not have sufficed.

Therefore, the question remains

Can we rely on the idea of polynomial commitment for batch decryption,
while only requiring a simple one-time *setup?*

First Core Idea: Commitment as Witness. The per-epoch setup is necessary in prior works due to the sampling of the (equivocable) commitment com. To eliminate this requirement, we must allow the users to do witness encryption without such a freshly sampled com. Towards this, we observe that: the KZG opening check still conforms to the PPEs system even if *the commitment is part of the witness*. Namely,

$$e(\pi_i, h^{\tau - x_i}) = e(\mathsf{com}/g^{\mathsf{tg}_i}, h).$$

In other words, if the users witness encrypts with respect to the above PPE system, the committee no longer needs to sample a random commitment com per epoch — effectively removing the per-epoch setup. While this idea works nicely, putting the commitment as a part of the witness, however, comes with significant challenges as we explain next.

[8] This can be computed from the common reference string without knowledge of τ, which is a secret trapdoor for the scheme.

[9] Indeed, this property of the KZG commitment verification has been used recently in many works to construct witness encryption for polynomial opening [CGPP24, FHAS24].

[10] In [CGPP24], it was shown that to achieve equivocation, one must use a randomized version of the KZG commitment scheme. For the purposes of keeping this overview simple, we do not explain here the randomized version.

[11] Indeed, this enables a public reconstruction of $f(x)$ by applying Lagrange interpolation on $\{(x_i, \mathsf{tg}_i)\}_{i \in S} \cup \{(x^*, f(x^*))\}$.

The "Intractability" Challenge. The core issue is the following. Since the committee does not sample the commitment, it is now *computationally easy* to find a satisfying witness. Indeed, the decryptor can pick any polynomial that interpolates (x_i, tg_i) as its witness and breaks the semantics security of the i-th ciphertext. Therefore, the challenge becomes

How can we restore the critical property that
the witness is intractable *to find?*

One may naturally wonder if we can fix this issue by asking the committee to produce a signature on com to authenticate the commitment com. That is, each user computes a witness encryption to the following PPEs system,

$$e(\sigma, h) \cdot e(\mathsf{com}, \mathsf{pk})^{-1} = g_{\mathsf{T}}^{0},$$
$$e(\pi, h^{\tau - x_i}) \cdot e(\mathsf{com}, h)^{-1} = e(g^{\mathsf{tg}_i}, h)^{-1}.$$

Although this idea appears to make the witness hard to compute, subtle attacks still persist. For instance, consider the adversary that wants to break the i-th ciphertext. He may still use a *constant polynomial* as the polynomial that interpolates (x_i, tg_i). In this case, the signature on com is still easy to compute since the dLog of com is picked by the adversary.

Second Core Idea: Shifted BLS. We resolve the "intractability" challenge by using what we call a *shift BLS* signature. Namely, we ask the committee to produce a signature, not on the commitment com itself, but on com *shifted by a public random group elements*. Note that, unlike [CGPP24], the committee does not need to hold any trapdoor related to this random group element and, hence, this random group element will not require a per-epoch setup.

To elaborate, users will now encrypt to an epoch id eid[12] that is publicly known ahead of time. We require the committee to produce a signature on $H(\mathsf{eid})/\mathsf{com}$. Specifically, the final PPEs that the users apply witness encryption to is,

$$\begin{aligned} e(\sigma, h) \cdot e(\mathsf{com}, \mathsf{pk}) &= e(H(\mathsf{eid}), \mathsf{pk}), \\ e(\pi, h^{\tau - x_i}) \cdot e(\mathsf{com}, h)^{-1} &= e(g^{\mathsf{tg}}, h)^{-1}. \end{aligned} \tag{2}$$

In other words, the witness that decrypts the above ciphertext (from user i) states

I have a signature σ on a $H(\mathsf{eid})/\mathsf{com}$,
where com has a proof of opening π at x_i to tg_i.

Intuitively, an adversary either needs to break the BLS signature scheme to generate a signature of its choice for any commitment com′ of its choosing, or it needs to break the KZG polynomial commitment opening to open a fixed com to any (x', tg') of its choosing. We formalize this intuition with a new assumption (Definition 1). In the technical section, we show our new assumption reduces to more standard assumptions in the algebraic group model [FKL18]. Intuitively, we show that the aforementioned two cases are the only possible ways that an adversary can break the system. This allows us to argue that every ciphertext not included in the B ciphertexts will never be decrypted.

Security Aspect. Finally, we note that we have ignored malleability issues so far for our ciphertext. These issues arise since an adversarial user may maul an honestly generated ciphertext such that if the mauled ciphertext is among the B ciphertexts selected, the witness for such a ciphertext can be converted to a witness for the honest ciphertext that *was not* among the B selected. For instance, if an adversary were to simply "copy" (x, tg) from an honest ciphertext ct when generating its own ciphertext ct′, it is easy to see that the witness to decrypt ct′ also serves as a witness to decrypt ct. Our construction handles such non-malleability issues in an identical manner to [CGPP24]. Specifically, the users must provide a non-interactive

[12]Each epoch will select B ciphertexts to decrypt, and the users whose ciphertexts are not chosen can encrypt to the next epoch, and so on.

proof of knowledge of the correct computation of the ciphertext. Further, to avoid the aforementioned copy attack, tg is generated by each user as $\mathsf{tg} = H(g^s)$, where the user outputs S and uses the non-interactive proof of knowledge to prove knowledge of s such that $S = g^s$. We refer the readers to the technical section for details on this part.

Put Everything Together. To conclude, when a user wants to encrypt a message for an epoch eid, it applies the witness encryption scheme (see (1)) to the PPEs system (2) to generate the ciphertext. Additionally, it sends a non-interactive (zero-knowledge) proof of knowledge to attest to the correct computation of the ciphertext. Once the B ciphertexts are chosen, the committee uses the (x_i, tg_i) from the selected ciphertexts to interpolate and generate a polynomial commitment com. The committee then signs $H(\mathsf{eid})/\mathsf{com}$ and broadcasts the signature in a threshold manner. Any user can now decrypt the B selected ciphertexts by redoing the computation of com, and, in turn, computing the required polynomial commitment openings. They finally use the broadcast signature to obtain the witnesses needed for decryption. Since the committee only sends a single signature, it satisfies our efficiency requirements from the dream goal. Further, the committee only needs to run a DKG at the start of the protocol to get Shamir's sharing of the BLS signing key.

3 Preliminaries

We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. The security parameter is denoted by $\lambda \in \mathbb{N}$. A function $f : \mathbb{N} \to \mathbb{N}$ is said to be polynomial if there exists a constant c such that $f(n) \leq n^c$ for all $n \in \mathbb{N}$, and we write $\mathsf{poly}(\cdot)$ to denote such a function. A function $f : \mathbb{N} \to [0, 1]$ is said to be negligible if for every $c \in \mathbb{N}$, there exists $N \in \mathbb{N}$ such that for all $n > N$, $f(n) < n^{-c}$, and we write $\mathsf{negl}(\cdot)$ to denote such a function. A probability is *noticeable* if it is not negligible, and *overwhelming* if it is equal to $1 - \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\lambda)$. For a set $\mathcal{S}$, we write $s \leftarrow \mathcal{S}$ to indicate that s is sampled uniformly at random from $\mathcal{S}$. For a random variable $\mathcal{D}$, we write $d \leftarrow \mathcal{D}$ to indicate that d is sampled according to $\mathcal{D}$. An algorithm $\mathcal{A}$ is PPT (probabilistic polynomial-time) if its running time is bounded by some polynomial in the size of its input. For two ensembles of random variables $\{\mathcal{D}_{0,\lambda}\}_{\lambda\in\mathbb{N}}$, $\{\mathcal{D}_{1,\lambda}\}_{\lambda\in\mathbb{N}}$, we write $\mathcal{D}_0 \approx_c \mathcal{D}_1$ to indicate that for all PPT $\mathcal{A}$, it holds that

$$\left| \Pr_{d\leftarrow\mathcal{D}_{0,\lambda}}[\mathcal{A}(d) = 1] - \Pr_{d\leftarrow\mathcal{D}_{1,\lambda}}[\mathcal{A}(d) = 1] \right| \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

We use e to denote an efficiently computable non-degenerate bilinear pairing from the groups $(\mathbb{G}_1, \mathbb{G}_2)$ to a target group $\mathbb{G}_T$ and use $\mathbb{F}$ to denote the associated field. The generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ are fixed to be g and h, respectively. We will represent the Shamir secret sharing of an element x as $[x]$, with i-th share represented as $[x]_i$. We note that while we overload the previously discussed notation $[n]$, the notation for secret shares will be clear from the context.

The Random Oracle Model. In the random oracle model (ROM), parties are given oracle access to some function H that is sampled uniformly at random from the space of all functions $H : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are finite non-empty sets. That is, parties can query their oracle on an input $x \in X$ and receive in return $H(x) \in \mathcal{Y}$. When proving the security of a protocol in the random oracle model, the simulator Sim is able to "control" the oracle, observing queries made by the adversary and simulating responses.

Lagrange Polynomials. We will use $L_i(x) = \Pi_{j=0,j\neq i}^{d-1} \frac{(x-x_j)}{(x_i-x_j)}$ to denote the i-th lagrange polynomial for polynomials of degree $\leq d$ over a sufficiently large field, corresponding to a domain $\Omega = \{x_0, x_1, \ldots, x_d\}$. These polynomials have the property that $f(\tau) = \sum_{i=0}^{d} L_i(\tau) f(x_i)$ for any point $\tau \notin \Omega$ in the field.

Non-interactive Zero-Knowledge proofs. Let $\mathcal{L}$ be an NP-language and $\mathcal{R}$ the corresponding NP-relation. A Simulation Extractable-NIZK for $\mathcal{R}$ consists of the following algorithms:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{crs}, \mathsf{td})$: Takes as input a security parameter, and outputs a common reference string crs and trapdoor td.

- $\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$: Takes as input crs and any pair $(x, w) \in \mathcal{R}$ and outputs a proof π for the statement $x \in \mathcal{L}$.
- $\mathsf{Verify}(\mathsf{crs}, x, \pi) \to b$: Takes as input crs, statement x and proof π and outputs a bit b indicating whether verification has passed or failed.
- $\mathsf{SimProve}(\mathsf{crs}, \mathsf{td}, x) \to \pi$: Takes as input crs, trapdoor td and statement x and outputs a *simulated* proof π.

We will drop the crs term wherever implicit. A SE-NIZK satisfies the following properties:

- *Perfect Completeness.* An SE-NIZK satisfies perfect completeness if for any $(x, w) \in \mathcal{R}$, we have

$$\Pr\left[\begin{array}{c}(\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)\end{array} : \mathsf{Verify}(x,\pi) = 1\right] = 1.$$

- *Proof of knowledge (and soundness).* For every PPT adversary $\mathcal{A}$, there is a PPT extractor Ext such that

$$\Pr\left[\begin{array}{c}(\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (x,\pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ w \leftarrow \mathsf{Ext}(\mathsf{crs})\end{array} : \begin{array}{c}\mathsf{Verify}(x,\pi) = 1 \\ (x,w) \notin \mathcal{R}\end{array}\right] \leq \mathsf{negl}(\lambda).$$

- *Computational Zero-knowledge.* There exists a simulator Sim such that for all stateful distinguishers $\mathcal{A}$ the following probabilities are equal:

$$\left|\Pr\left[\begin{array}{c}(\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (x,w) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)\end{array} : \begin{array}{c}(x,w) \in \mathcal{R} \\ \mathcal{A}(\pi) = 1\end{array}\right] - \Pr\left[\begin{array}{c}(\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (x,w) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \pi \leftarrow \mathsf{SimProve}(\mathsf{crs}, \mathsf{td}, x)\end{array} : \begin{array}{c}(x,w) \in \mathcal{R} \\ \mathcal{A}(\pi) = 1\end{array}\right]\right| \leq \mathsf{negl}(\lambda).$$

- *Weak Simulation-Extractability.* For every PPT adversary $\mathcal{A}$, there exists a PPT extractor Ext such that

$$\Pr\left[\begin{array}{c}(\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (x,\pi) \leftarrow \mathcal{A}^{\mathsf{SimProve}} \\ w \leftarrow \mathsf{Ext}\end{array} : \begin{array}{c}\mathsf{Verify}(x,\pi) = 1 \\ \wedge\, (x,w) \notin \mathcal{R} \\ \wedge\, x \notin Q\end{array}\right] \leq \mathsf{negl}(\lambda).$$

  where $\mathcal{A}$ has oracle access to $\mathsf{SimProve}(\mathsf{crs}, \mathsf{td}, \cdot)$, and Q is a list of queries made by the adversary.

We will also demand that the proof is straight-line extractable. This means that the extractor can, on input a valid proof and (potentially) the list of random-oracle queries made by the adversary (prover), extract a witness with overwhelming probability without rewinding the prover.

Algebraic Group Model. We prove security in the algebraic group model [FKL18]. In this model, adversaries are restricted to performing algebraic manipulation of group elements that they have seen so far. When an adversary outputs a value $Y \in \mathbb{G}$, it also produces a *linear combination* $(e_1, \ldots, e_m)$ of elements $(X_1, \ldots, X_m)$ that it has seen so far, such that $Y = \prod X_i^{e_i}$.

4 Model and Definitions

We use essentially the same security and efficiency requirements of batched-threshold encryption as defined in [CGPP24], with one main difference — our definition does not involve an epoch setup.

- $\mathsf{Setup}(1^\lambda, n, t, B) \to \{\mathsf{pk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)\}$: On input the threshold t for n parties and a batch size B, outputs the public key pk along with secret keys for each party.

- $\mathsf{Enc}(\mathsf{pk}, \mathsf{eid}, m) \to \mathsf{ct}$: On input a message m, an epoch id eid, and a public key pk, outputs a ciphertext ct.
- $\mathsf{BatchDec}((\mathsf{ct}_1, \ldots, \mathsf{ct}_B), \mathsf{sk}_i) \to \mathsf{pd}_i$: On input B ciphertexts $(\mathsf{ct}_1, \ldots, \mathsf{ct}_B)$ and a secret key shares sk_i, outputs a partial decryption pd_i or $\perp$.
- $\mathsf{Combine}(\mathsf{pk}, \mathsf{eid}, (\mathsf{ct}_1, \ldots, \mathsf{ct}_B), \{\mathsf{pd}_i\}_{i \in S}) \to (m_1, \ldots, m_B)$ Takes as input the public key pk, epoch id eid and $t+1$ partial decryptions where $S \subseteq [n]$ and $|S| = t+1$, and outputs messages $(m_1, \ldots, m_B)$ or $\perp$.

Efficiency. We impose various efficiency constraints on the above algorithms. Setup must run in time $\mathsf{poly}(n, t, B, \lambda)$. Enc must run in $O(1)$ time. BatchDec must be non-interactive i.e., each party computes it's partial decryption independently. It must also run in time $\mathsf{poly}(B, \lambda)$, and the size of each partial decryption must be sub-linear in the batch size $(o(B))$ and *independent* of the number of parties. Finally, Combine must run in time $\mathsf{poly}(B, n, \lambda)$.

The ideal functionality. We now describe the ideal functionality (Fig. 1). Note that even though there is no interactive epoch setup, ciphertexts are still encrypted to a particular epoch and the parties must agree on a batch of ciphertexts to decrypt in each epoch. Again, our ideal functionality is identical to that of [CGPP24] except for the omission of EpochSetup. However, the encrypt and decrypt methods still make use of an epoch id eid.

$\mathcal{F}_{\mathsf{bTPKE}}$

Parties: Clients who encrypt messages $(U_1, \ldots, U_m)$ and Servers $(S_1, \ldots, S_n)$.
Parameters: Space of receipts $\mathcal{C} \subset \{0,1\}^*$, number of servers n, threshold $0 < t \leq n$, and batch size B.

Setup. If all servers send $(\mathsf{Setup}, (S_1, \ldots, S_n))$ then notify all servers that setup was successful, else, send $\perp$ to all servers. Adversary specifies a distribution Γ over $\mathcal{C}$.

If the setup was successful:

Encryption. When a client U sends $(\mathsf{Enc}, m, \mathsf{eid})$, sample a receipt $c \leftarrow \Gamma$ and store (c, m, eid). Announce (c, eid) to all parties.

BatchDecryption. When $t+1$ servers send $(\mathsf{Dec}, (c_1, \ldots, c_B), \mathsf{eid})$, if tuples of the form $\{(c_i, m_i, \mathsf{eid})\}_{i \in [B]}$ have been previously stored, and $\mathsf{eid} \in \mathcal{L}_r$, then send the corresponding messages $\{m_i\}_{i \in [B]}$ to the servers. Remove eid from $\mathcal{L}_r$ and add it to $\mathcal{L}_c$.

Figure 1: Ideal Functionality for a Batched Threshold Public-Key Encryption scheme.

5 Construction

We now present our construction of batched-threshold encryption. Note that we present an optimized version of the protocol which uses a roots of unity domain to interpolate tags across ciphertexts. This allows us to compute openings proofs of a KZG commitment at all points in the domain in $O(B \log B)$ time. However, this also means that if two clients chose the same $\hat{x}$, then only one of these ciphertexts can be included in the block. This can be avoided by letting clients choose an arbitrary $\hat{x} \in \mathbb{F}$ at the cost of a more expensive combine step. We prove that the latter securely emulates the ideal functionality Fig. 1.

Setup: Our setup is significantly simpler than that of [CGPP24]. We only require a *single* execution of a distributed key generation protocol to establish shamir shares of a secret key $\mathsf{sk} \in \mathbb{F}$ among n parties. The public key is $\mathsf{pk} = h^{\mathsf{sk}}$ along with $\{h^{[\mathsf{sk}]_j}\}_{j \in [n]}$. We also require a powers-of-tau CRS – $\{g, g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^{B-1}}\}$

– which can be sampled in a distributed manner as outlined in [NRBB24]. [13] Finally, the parties also run the setup for a NIZK proof system (in a distributed manner if necessary) and publish the public parameters (if any). In our concrete implementation, we will use sigma protocols, which are secure in the random oracle model and do not require an interactive setup.

Encrypt: Clients sample a random value $\hat{x} \in \Omega = \{1, \omega, \omega^2, \ldots, \omega^{B-1}\}$, where ω is a root of unity in $\mathbb{F}$ of order B. Informally, they create a witness encryption to the following relation, using the compiler from [BC16, GKPW24]:

> I know an opening proof π for some KZG commitment [KZG10] com, at the point $\hat{x}$ to the value tg, and I know a BLS signature σ on $\delta = H(\mathsf{eid})/\mathsf{com}$,

where variables in red are part of the witness and those in blue are part of the statement. More formally the set of PPE's that need to be satisfied can be written as:

$$e(H(\mathsf{eid}) \cdot g^{-\mathsf{tg}}, h) = e(\pi, h^{(\tau - \hat{x})}) \cdot e(\delta, h)$$

$$e(\delta, \mathsf{pk}) = e(\sigma, h)$$

Or equivalently,

$$e(\mathsf{com} \cdot g^{-\mathsf{tg}}, h) = e(\pi, h^{(\tau - \hat{x})})$$

$$e(H(\mathsf{eid}) \cdot \mathsf{com}^{-1}, \mathsf{pk}) = e(\sigma, h)$$

where terms in red denote the witness.

Adding CCA2 Security. To achieve security CCA2 security for our encryption scheme, we must a) prove that ciphertext is well-formed and b) prevent an adversary from choosing the same tg as an honest party.

We require the former to ensure correctness for ciphertexts created by the adversary, which is crucially used by the simulator to extract the underlying message as part of the security proof. The latter is necessary to prevent a concrete attack that can launched by the adversary. Namely, the adversary first looks at an honest party's ciphertext encrypted to some string tg^* and creates another "dummy" ciphertext which is encrypted to the same statement – $(\hat{x}, \mathsf{eid}, \mathsf{tg}^*)$. The adversary then gets the dummy ciphertext decrypted by including it in the next batch, and the partial decryptions will allow the adversary to not only decrypt the dummy ciphertext but also the honest party's ciphertext as they require the *same* witness. To prevent this, we will choose tg^* as the output of a one-way function f on a randomly chosen input s and demand a proof of knowledge of s. Concretely, we use $f(s) \coloneqq g^s$ as it allows us to use very efficient sigma protocols with small proofs.

Batch Decryption: When choosing a batch of ciphertexts to decrypt, the committee must ensure that the $\hat{x}$ values across ciphertexts are distinct. The committee members first compute $\mathsf{com} = g^{p(\tau)}$, where $p(X)$ is the degree-$(B-1)$ polynomial interpolating the tags across the B ciphertexts. The j-th party then publishes $\sigma_j = (H(\mathsf{eid})/\mathsf{com})^{[\mathsf{sk}]_j}$ as their partial decryption.

Combine: We first recover the commitment $\mathsf{com} = g^{p(\tau)}$, as done in the Batch Decryption step, and compute opening proofs for all points in Ω using [FK23]. Given t partial signatures on $H(\mathsf{eid})/\mathsf{com}$, we can interpolate in the exponent to recover σ such that $e(H(\mathsf{eid})/\mathsf{com}, \mathsf{pk}) = e(\sigma, h)$. Finally, we simply run the witness encryption's decrypt algorithm from [BC16, GKPW24] to obtain the underlying messages.

[13] We note that one can also use a powers of tau CRS that has been previously setup in a distributed manner. For instance, https://ceremony.ethereum.org has 141,416 contributions. Soundness holds as long as at least one party deleted the randomness they used as part of their contribution.

Batched-Threshold Encryption

Parameters: A pairing friendly group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$, and random oracles $H : \{0,1\}^* \to \{0,1\}^*$, $H_F : \{0,1\}^* \to \mathbb{F}$, and $H_G : \{0,1\}^* \to \mathbb{G}_1$.

- $\underline{\mathsf{Setup}(1^\lambda, n, t, B)}$: A trusted dealer provides the committee with (t, n)-shares of a random field element $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{F}$ and publishes the committee's public key $\mathsf{pk} = h^{\mathsf{sk}}$ along with $\{h^{[\mathsf{sk}]_j}\}_{j\in[n]}$ with $\mathsf{crs}_{\mathsf{KZG}} = (g, g^\tau, \ldots, g^{\tau^{B-1}}, h, h^\tau)$. They also run the setup for NIZK and publish $\mathsf{crs}_{\mathsf{NIZK}}$.
- $\underline{\mathsf{Enc}(\mathsf{pk}, \mathsf{eid}, m)}$: Clients first sample a random value $\hat{x} \in \Omega = \{1, \omega, \omega^2, \ldots, \omega^{B-1}\}$, where ω is a root of unity in $\mathbb{F}$ of order B. They also sample $s \leftarrow\!\!\$\ \mathbb{F}$ and compute $S \leftarrow g^s$, and $\mathsf{tg} \leftarrow H_F(S)$. The ciphertext is set to be:
  - $\mathsf{ct}^{(1)} \coloneqq H(e(H_G(\mathsf{eid}) \cdot g^{-\mathsf{tg}}, h)^\alpha) \oplus M$
  - $\mathsf{ct}^{(2)} \coloneqq h^{\alpha(\tau - \hat{x})}$
  - $\mathsf{ct}^{(3)} \coloneqq h^\alpha \mathsf{pk}^\beta$
  - $\mathsf{ct}^{(4)} \coloneqq h^\beta$
  - S
  - $\hat{x}$
  - $\Phi = \mathsf{NIZK.Prove}\{\alpha, \beta, s \mid S = g^s \wedge \mathsf{ct}^{(1)} \wedge \mathsf{ct}^{(2)} = h^{\alpha(\tau-\hat{x})} \wedge \mathsf{ct}^{(3)} = h^\alpha \mathsf{pk}^\beta \wedge \mathsf{ct}^{(4)} = h^\beta\}$
- $\underline{\mathsf{BatchDec}((\mathsf{ct}_0, \ldots, \mathsf{ct}_{B-1}), \mathsf{sk}_i)}$: A threshold number of parties agree on a batch of ciphertexts $\{\mathsf{ct}_0, \ldots, \mathsf{ct}_{B-1}\}$ such that every ciphertext has a valid proof Φ_i and $\{\mathsf{ct}_0.\hat{x}, \ldots, \mathsf{ct}_{B-1}.\hat{x}\} = \Omega$. Compute $\mathsf{com} = g^{p(\tau)}$ using $\mathsf{crs}_{\mathsf{KZG}}$, where $p(X)$ is a degree-$(B-1)$ polynomial such that $\{p(\mathsf{ct}_i.\hat{x}) = \mathsf{tg}_i\}_{i\in[B]}$, where $\mathsf{tg}_i = H(S_i)$. The j-th party then publishes
$$\sigma_j = (H_G(\mathsf{eid})/\mathsf{com})^{[\mathsf{sk}]_j}$$
as the partial decryption.
- $\underline{\mathsf{Combine}(\mathsf{pk}, (\mathsf{ct}_1, \ldots, \mathsf{ct}_B), \{\sigma_i\}_{i\in\mathcal{S}})}$: Recover the commitment $\mathsf{com} = g^{p(\tau)}$, where $\{p(\mathsf{ct}.\hat{x}_i) = \mathsf{tg}_i\}_{i\in[B]}$ and compute opening proofs $\{\pi_i = g^{q_i(\tau)}\}_{i\in[B]}$, where $q_i(X) = (p(X) - \mathsf{tg}_i)/(X - \mathsf{ct}.\hat{x}_i)$ and $\mathsf{tg}_i = H_F(\mathsf{ct}_i.S)$. Given t partial decryptions $\{\sigma_j \mid e(H_G(\mathsf{eid})/\mathsf{com}, \mathsf{pk}_j) = e(\sigma_j, h)\}_{j\in\mathcal{S}}$, where $\mathcal{S} \subseteq [n]$ and $|\mathcal{S}| > t$, interpolate in the exponent to recover σ such that $e(H_G(\mathsf{eid})/\mathsf{com}, \mathsf{pk}) = e(\sigma, h)$. Output
$$\{\mathsf{ct}_i^{(1)} \oplus H(e(\pi_i, \mathsf{ct}_i^{(2)}) \cdot e(\delta, \mathsf{ct}_i^{(3)}) \cdot e(\sigma^{-1}, \mathsf{ct}_i^{(4)}))\}_{i\in[B]}.$$

Figure 2: Protocol for Batched-Threshold Encryption

6 Implementation and Evaluation

To evaluate the concrete performance of our scheme, and to ensure a fair comparison against prior work, we implemented a performant version of our scheme in Rust using the same libraries. We use `arkworks` [ac22] for implementations of pairing-friendly curves and their associated algebra, and the `merlin` [dcc20] library to handle the Fiat-Shamir transform. Our implementation is available at https://github.com/guruvamsi-policharla/batched-threshold-pp

Setup. All of our experiments were run on a 2019 MacBook Pro with a 2.4 GHz Intel Core i9 processor and 16 GB of DDR4 RAM in single-threaded mode.

Algebra. We use BLS12-381 as our pairing friendly curve and the BLAKE3 hash function as our random oracle.

Dealer. We assume a trusted dealer for the distributed key generation during the setup phase but this can be emulated using any verifiable secret sharing scheme [Sch99, GHL22, Gro21, KMM+23, CD24, Kat24].

Evaluation. We now evaluate the performance of our scheme. We answer the following questions:

- How long does it take to encrypt a message? What is the size of the corresponding ciphertexts?
- How long does it take for a committee member to compute partial decryptions, and how does it vary with batch/committee size?
- How long does it take to recover all messages given partial decryptions, and how does it vary with batch size?

Batch size	Partial decrypt (ms)	Reconstruct (ms)
8	49.2	60.5
32	199.8	168.1
128	809.9	646.1
512	3203.9	3026.5

Table 2: Time taken to compute partial decryptions and to reconstruct messages with varying batch size.

Parameter	[BO22]	[MGZ22, Shu21]	[CGPP24]	This Work
Increase in Ciphertext size	$\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_2\rvert$	$\lvert\mathbb{G}_2\rvert$	$2\lvert\mathbb{G}_2\rvert + \lvert\mathbb{G}_1\rvert + 3\lvert\mathbb{F}\rvert + 2$	$3\lvert\mathbb{G}_2\rvert + \lvert\mathbb{G}_1\rvert + 4\lvert\mathbb{F}\rvert + 2$
|Partial Decryptions|	$nB\lvert\mathbb{G}_1\rvert$ (3 MB)	$n\lvert\mathbb{G}_1\rvert$ (6 KB)	$n(\lvert\mathbb{G}_1\rvert + \lvert\mathbb{F}\rvert)$ (10 KB)	$n(\lvert\mathbb{G}_1\rvert)(6\ KB)$
|Partial Decryptions|/|Block Size|	500%	1%	2%	1%

Table 3: Ciphertext size and total decryption broadcast size for n committee members, and B transactions per block. Concrete numbers were obtained with the BLS12-381 curve, using the average transaction and block size on Ethereum in 2023 for a committee size of 128 with a block of 512 transactions. We note that the numbers of [MGZ22, Shu21] are for CPA-secure ciphertexts. Adding CCA security would increase ciphertext size as shown in [DHMW23]. We compare against the most efficient schemes with weakest security guarantees to highlight that despite the handicap given to prior work, our scheme is competitive.

Encryption. The ciphertext contains one $\mathbb{G}_1$ element, three $\mathbb{G}_2$ elements, 4 $\mathbb{F}$ elements, a two byte description of $\hat{x}$, and a string proportional to the message length (32 bytes), resulting in a total size of 498 bytes. Creating a ciphertext takes approximately 8.5ms, and is independent of committee/batch size. Our encryption time is $\approx 40\%$ slower and ciphertexts are $\approx 35\%$ larger than prior work [CGPP24]. In contrast, we

only need a very simple *one-time* setup, whereas [CGPP24] required a complicated multi-party computation-based setup and per-epoch setup. We view this as a clearly favorable trade-off for the setting of mempool privacy.

Batch Decryption. Before issuing a partial decryption, each party must check that all ciphertexts in the batch are valid by verifying the accompanying zero-knowledge proof. We find that this is the most expensive step in the protocol, encompassing $\approx 99\%$ of the times reported in Table 2. Each partial decryption is one $\mathbb{G}_1$ element (48 bytes with compression), and 40% smaller than the partial decryptions in [CGPP24] but computing them is $\approx 20\%$ slower.

Reconstruct. Given sufficiently many partial decryptions, any party can recover the messages from the entire batch of ciphertexts that were decrypted in that epoch. In the process, we need to produce KZG opening proofs at all points in the domain Ω. Using the techniques from [FK23], we can produce all openings using $O(B \log B)$ group operations and $O(B)$ pairings. In Table 2, we provide numbers assuming that all parties respond with partial decryptions, and each partial decryption is verified to be correct. We note here an important distinction when comparing numbers reported in [CGPP24] with those in Table 2. [CGPP24] reports "optimistic" timings and size, where they assume that the entire committee responds and that all partial decryptions are correct. If an inconsistency is detected, they demand proofs that the partial decryption was computed correctly. Hence, they exclude the time taken to generate these proofs and sizes of the these proofs from the reported numbers. Our partial decryptions are publicly testable using a pairing check, and hence malformed partial decryptions can be detected without incurring any additional communication. At a batch size of 512, we are faster than [CGPP24] as we use an optimized implementation to compute pairing product equation where the miller loop is evaluated only once instead of for every pairing. We also note that increasing committee size does not have a noticeable impact on the reconstruction time as computing KZG opening proofs takes a majority of the time – which is independent of committee size.

Committee Churn. Parties may leave/join the protocol for a variety of reasons. In such situations, we can ensure the protocol can continue without interruption using techniques from the line of work on Proactive Secret Sharing [HJKY95, CKLS02, DJ97, WWW02, ZSVR05, SLL08, MZW+19, GKM+22, VAFB22]. From the benchmarks in [CGPP24], this could be completed in under 10 seconds for a committee size of 64 in a simulated WAN on the Google Cloud Platform using the `NetEm` package with a delay time of 200 ms, jitter time of 20 ms and a rate of 10 mbit per second.

7 Security Proof

We prove security with guaranteed output delivery against a fully malicious adversary corrupting up to $t < n/2$ members of the committee and any number of users creating ciphertexts. In the process, we introduce a new interactive assumption (Definition 1) and justify its hardness. We then prove Theorem 5 which shows that the protocol in Fig. 2 securely emulates the ideal functionality in Fig. 1.

We start by describing our new interactive assumption, that we term $i - kzg$, below.

Definition 1 (i-kzg Game)**.** *Let g and, h be the generators of $\mathbb{G}_1$, and $\mathbb{G}_2$, respectively. We define an oracle $\mathsf{O}^{a,t,\tilde{x},\{b_i\}_{i\in[m]}}(j, f(X))$ parametrized by $t, a, \tilde{x}, \{b_i\}_{i\in[m]} \in \mathbb{F}$. It takes as input a degree-$(B-1)$ polynomial $f(X)$ and an index j, and if $f(\tilde{x}) \neq 0$ it returns $(g^{b_j} - g^{f(t+\tilde{x})})^a$, else it returns $\perp$. Consider the following game for $B, m = \mathsf{poly}(\lambda)$, where $\mathcal{A}$ is a non-uniform probabilistic polynomial time algorithm:*

1. *Sample a, t, u, v, $\tilde{x}$, and $\{b_i\}_{i\in[m]}$ from $\mathbb{F}$ and send the following to $\mathcal{A}$:*

$$(\tilde{x}, g, g^t, g^{t^2}, \ldots, g^{t^{B-1}}, \{g^{b_i}\}_{i\in[m]}, h, h^t, h^a, h^v, h^{tu}, h^{u+av})$$

2. *For each $j \in [m]$, $\mathcal{A}$ can make* one *query of the form*

$$\mathsf{O}^{a,t,\tilde{x},\{b_i\}_{i\in[m]}}(j, f_j(X)),$$

for any degree-$(B-1)$ polynomial $f_j(X)$ of its choice.

3. *$\mathcal{A}$ wins if it outputs $e(g,h)^{b_j u}$ for some $j \in [m]$.*

The i-kzg game is said to be hard if for all non-uniform probabilistic polynomial time algorithms $\mathcal{A}$, the probability that $\mathcal{A}$ wins the game is negligible.

While the above interactive assumption closely resembles the security properties required from our scheme, we show below that the above assumption can be reduced to the standard (non-interactive) k-discrete log assumption in the algebraic group model (AGM). To do so, we follow the analysis of algebraic adversaries as presented in [BC16, GKPW24].

We first prove a master theorem for the compilers in [BC16, GKPW24] that will help simplify our analysis. We use the following notation for convenience. Let $\mathbf{X} \in \mathbb{G}^{u \times v}$, for some group $\mathbb{G}$ and $e \in \mathbb{F}^n$. We will denote the i-th row by $\mathbf{X}_i$ and $X_{i,j}$ to denote the entry in the j-th column of the i-th row. We will write $\mathbf{X}_i^e$ to denote $\prod_{j \in [u]} X_{i,j}^{e_i}$. We will also use $\circ$ to denote matrix multiplication between two matrices with elements in $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, where the multiplication operation is replaced with a pairing and addition is replaced with the group operation. Similarly, we will use $\cdot$ to denote multiplication between a matrix of field elements and a matrix of group elements where the multiplication operation is replaced with group exponentiation and addition is replaced by the group operation.

Theorem 2 (Master Theorem)**.** *Let $\mathbf{L} \in \mathbb{G}_1^{u \times v_1}$, $\mathbf{R} \in \mathbb{G}_2^{v_2 \times u}$ and $\mathbf{b} \in \mathbb{G}_T^u$, define a system of pairing product equations, that are satisfied by $\mathbf{w}_L \in \mathbb{G}_2^{v_1}$, and $\mathbf{w}_R \in \mathbb{G}_1^{v_2}$ iff $\mathbf{L}_j \circ \mathbf{w}_L \cdot (\mathbf{R}^\intercal)_j \circ \mathbf{w}_R = \mathbf{b}_j$ for all $j \in [u]$. For all non-uniform PPT algebraic adversaries $\mathcal{A}$ such that*

$$\Pr[\mathcal{A}(\mathbf{L}, \mathbf{R}, \mathbf{b}, \{(\mathbf{L}^\intercal)_i^r\}_{i \in [v_1]}, \{\mathbf{R}_i^r\}_{i \in [v_2]}) \to \mathbf{b}^r \mid r \leftarrow\!\$\, \mathbb{F}^u] = \varepsilon,$$

there exists a uniform PPT algebraic adversary $\mathcal{A}'^{\mathcal{A}}$ which outputs $\mathbf{w}_L \in \mathbb{G}_2^{v_1}$, and $\mathbf{w}_R \in \mathbb{G}_1^{v_2}$ such that $\mathbf{L}_j \circ \mathbf{w}_L \cdot (\mathbf{R}^\intercal)_j \circ \mathbf{w}_R = \mathbf{b}_j$ for all $j \in [u]$, along with corresponding representations, with probability ε.

Proof. Let $\mathcal{T}$ denote the completion set of target group elements viz. all possible pairing combinations of elements in $\mathbb{G}_1$ and $\mathbb{G}_2$ seen by $\mathcal{A}$ so far. Since $\mathcal{A}$ is an algebraic adversary, for any group element in the target group that it outputs, it also attaches the specific linear combination $\alpha \in \mathbb{F}^{|\mathcal{T}|}$ of elements in $\mathcal{T}$ that it used to produce the output. The terms in $\mathcal{T}$ can be categorized into three different types:

- Degree-0: Terms obtained by pairing an element from $\mathbf{L}$ with and element from $\mathbf{R}$ or a term from $\mathbf{b}$.
- Degree-1: Terms obtained by pairing an element from $\mathbf{L}$ with an element from $\{\mathbf{R}_i^r\}_{i \in [v_2]}$ or an element from $\{(\mathbf{L}^\intercal)_i^r\}_{i \in [v_1]}$ with an element from $\mathbf{R}$.
- Degree-2: Terms obtained by pairing an element from $\{\mathbf{L}_i^r\}_{i \in [v]}$ with an element from $\{\mathbf{R}_i^r\}_{i \in [v]}$.

Observe that the coefficients of Degree-0 and Degree-2 terms must be zero in the linear combination used to produce $\mathbf{b}^r$. If not, observe that we can solve for at least one of the r_i's and $\mathcal{A}$ can then be used to solve for r^* given $(g^{r^*}, h^{r^*}) \in \mathbb{G}_1 \times \mathbb{G}_2$ – a variant of the discrete logarithm problem.

We now describe the strategy of $\mathcal{A}'^{\mathcal{A}}$. Let $\mathbf{R}_j$ denote the j-th row of $\mathbf{R}$. We can then define $S_j = \{s \mid e(s, \mathbf{R}_j^r) \in \mathcal{T}\}$, to be the terms where some element in $\mathbb{G}_1$ is paired with $\mathbf{R}_j^r$. We also define $\alpha_j \subseteq \alpha$ to be the corresponding coefficients of the elements in the set $\{e(s, \mathbf{R}_j^r) \mid s \in S_j\}$, in the linear combination used to produce $\mathbf{b}^r$. It is now easy to see that $\mathcal{A}'^{\mathcal{A}}$ can produce a satisfying $\mathbf{w}_R$ by setting the j-th element of $\mathbf{w}_R$ to be $\mathbf{w}_{R,j} \leftarrow S_j^{\alpha_j}$. Analogously for $\mathbf{w}_L$. □

Definition 3 ((k)-dlog Assumption)**.** *Let g, and h be the generations of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. Then for all non-uniform PPT algorithm $\mathcal{A}$, it holds that*

$$\Pr[\mathcal{A}(g, g^a, g^{a^2}, \ldots, g^{a^k}, h, h^k) \to k \mid a \leftarrow\!\$\, \mathbb{F}] \leq \mathsf{negl}(\lambda).$$

Lemma 4. *The i-kzg game is hard in the algebraic group model provided the $(B-1)$-dlog assumption holds.*

Proof. First observe that we can take any algebraic adversary $\mathcal{A}$ which produces $e(g,h)^{b_j u}$ for some $j \in [m]$ and construct an adversary $\mathcal{A}'$ which produces δ, π, and σ (and corresponding linear combinations of elements in $\mathbb{G}_1$) such that:

$$e(g^{b_j}, h) = e(\pi, h^t) \cdot e(\delta, h)$$
$$e(\delta, h^a) = e(\sigma, h)$$

This can be achieved by applying the master theorem with

$$\mathbf{R} = \begin{bmatrix} h & h^a \\ h^t & 0 \\ 0 & h \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ e(g,h)^{b_j u} \end{bmatrix},$$

where a satisfying witness is of the form $\mathbf{w}_R = (\delta, \pi, \sigma^{-1})$. We can then redefine $\mathsf{com} = g^{b_j} - \delta$ to obtain the triple $(\mathsf{com}, \pi, \sigma)$ which satisfies:

$$e(\mathsf{com}, h) = e(\pi, h^t)$$
$$e(g^{b_j}, \mathsf{pk}) = e(\sigma, h) \cdot e(\mathsf{com}, \mathsf{pk})$$

Suppose there existed an adversary $\mathcal{A}$ which succeeded in the i-kzg game with non-negligible probability. Then we will construct an adversary $\mathcal{A}'$ which can solve the $(B-1)$-dlog problem with non-negligible probability. We begin by describing $m+2$ types of strategies, and $\mathcal{A}'$ will run one of these, chosen uniformly at random. We will then show that for all non-uniform PPT machines $\mathcal{A}$, at least one of these strategies will succeed, and hence $\mathcal{A}'$ will solve the $(B-1)$-dlog problem with non-negligible probability. Let the $(B-1)$-dlog challenge be $(g, g^z, g^{z^2}, \ldots, g^{z^{B-1}}, h, h^z)$.

1. Set $t \coloneqq z$ and sample $a, \tilde{x}, u, v$, and b_i's uniformly at random. $\mathcal{A}'$ then runs $\mathcal{A}$ with the challenge instance
$$(\tilde{x}, g, g^z, g^{z^2}, \ldots, g^{z^{B-1}}, \{g^{b_i}\}_{i\in[m]}, h, h^z, h^a, h^v, h^{zu}, h^{u+av})$$
and can simulate responses from the oracle using knowledge of a.

2. Similarly, Set $a \coloneqq z$, and sample remaining variables uniformly at random. Responses to oracle queries can be simulated using knowledge of b_i's and t.

3. We describe a strategy for each $j \in [m]$: Set $b_j \coloneqq z$ and sample remaining variables uniformly at random. Responses to oracle queries can be simulated using knowledge of a.

Since all witness variables are in $\mathbb{G}_1$ and hence expressed as a linear combination over

$$(g, g^t, g^{t^{B-1}}, \{g^{b_i}, (g^{b_i} - g^{f_i(t+\tilde{x})})^a\}_{i\in[m]}).$$

Let $Q = \{(i, f_i, (g^{b_i} - g^{f_i(t+\tilde{x})})^a)\}_{i\in[m]}$ denote the query-response pairs for queries made by the adversary to the oracle in the i-kzg game and $\mathcal{A}$ outputs $(\mathsf{com}, \pi, \sigma)$, along with their representation. Consider the following events:

1. We denote by E_1, the event that com is expressed as a linear combination of powers-of-t – $(g, g^t, \ldots, g^{t^{B-1}})$ – and no other elements.

2. We denote by E_2, the event that there exists $(j, f_j, \sigma) \in Q$ such that $\mathsf{com} = g^{f_j(t)}$.

Case $\neg E_1$: Suppose $\mathcal{A}$ returned a representation of com where the coefficients of $\{g^{b_i}, (g^{b_i} - g^{f_i(t+\tilde{x})})^a\}_{i\in[m]}$ are non-zero. Then we have coefficients $p_i, \beta_i, \tilde{\beta}_i, \gamma_i, \tilde{\gamma}_i$ such that:

$$\sum_{i=0}^{B-1} p_i \cdot t^i + \sum_{i=1}^{m} \beta_i \cdot b_i + \sum_{i=1}^{m} \tilde{\beta}_i \cdot b_i t + \sum_{i=1}^{m} \gamma_i \cdot (b_i - f_i(t+\tilde{x}))a + \sum_{i=1}^{m} \tilde{\gamma}_i \cdot (b_i - f_i(t+\tilde{x}))at = 0.$$

We will now use these coefficients to solve the $(B-1)$-dlog problem. Suppose

$$\sum_{i=1}^{m} \gamma_i \cdot (b_i - f_i(t+\tilde{x}))a + \sum_{i=1}^{m} \tilde{\gamma}_i \cdot (b_i - f_i(t+\tilde{x}))at \neq 0$$

and. In the case where we set $a \coloneqq z$, we can recover z as

$$z = -\frac{\sum_{i=0}^{B-1} p_i \cdot t^i + \sum_{i=1}^{m} \beta_i \cdot b_i + \sum_{i=1}^{m} \tilde{\beta}_i \cdot b_i t}{\sum_{i=1}^{m} \gamma_i \cdot (b_i - f_i(t+\tilde{x})) + \sum_{i=1}^{m} \tilde{\gamma}_i \cdot (b_i - f_i(t+\tilde{x}))t}$$

Thus, the following equations must hold:

$$\sum_{i=0}^{B-1} p_i \cdot t^i + \sum_{i=1}^{m} (\beta_i + \tilde{\beta}_i \cdot t) \cdot b_i = 0$$

$$\sum_{i=1}^{m} \gamma_i \cdot (b_i - f_i(t+\tilde{x}))a + \sum_{i=1}^{m} \tilde{\gamma}_i \cdot (b_i - f_i(t+\tilde{x}))at = 0$$

Note that since E_1 is false, at least one of $\beta_i, \tilde{\beta}_i, \gamma_i, \tilde{\gamma}_i$ must be non-zero. There are now two cases two analyze:

1. If one of $\beta_i, \tilde{\beta}_i$ is non-zero, and $\sum_{i=1}^{m} (\beta_i + \tilde{\beta}_i \cdot t) \cdot b_i \neq 0$, then there must exist some $i^* \in [m]$ such that $(\beta_{i^*} + \tilde{\beta}_{i^*} \cdot t) \cdot b_{i^*} \neq 0$. We can then guess this i^*, uniformly at random, and set $b_{i^*} \coloneqq z$. With non-negligible probability we will be correct, and we can solve for z as:

   $$z = -\frac{\sum_{i=0}^{B-1} p_i \cdot t^i + \sum_{i \neq i^*} (\beta_i + \tilde{\beta}_i \cdot t) \cdot b_i}{(\beta_{i^*} + \tilde{\beta}_{i^*} \cdot t)}.$$

   If instead $\sum_{i=1}^{m} (\beta_i + \tilde{\beta}_i \cdot t) \cdot b_i = 0$, but one of the $\beta_i, \tilde{\beta}_i$ is non-zero, then we can set $t \coloneqq z$, and recover z as

   $$z = -\frac{\sum_{i=1}^{m} \beta_i \cdot b_i}{\sum_{i=1}^{m} \tilde{\beta}_i \cdot b_i}$$

   Hence, it must be true that all the $\beta_i, \tilde{\beta}_i$ are zero.

2. If one of $\gamma_i, \tilde{\gamma}_i$ is non-zero, and

   $$\sum_{i=1}^{m} \gamma_i \cdot (b_i - f_i(t+\tilde{x}))a + \sum_{i=1}^{m} \tilde{\gamma}_i \cdot (b_i - f_i(t+\tilde{x}))at = 0,$$

   then we set $t \coloneqq z$ and solve for z by finding the roots of

   $$\sum_{i=1}^{m} (\gamma_i + \tilde{\gamma}_i \cdot z) \cdot (b_i - f_i(z+\tilde{x})) = 0.$$

   We can then test all roots to see which one is consistent with the challenge instance. Hence, it must be true that all the $\gamma_i, \tilde{\gamma}_i$ are zero.

Thus, E_1 must be true.

Case $E_1 \wedge \neg E_2$**:** Since E_1 is true, com is a linear combination of $(g, g^t, \ldots, g^{t^{B-1}})$. Let f' be the degree-$(B-1)$ polynomial that $\mathcal{A}$ provided as a representation of com in the powers-of-t basis viz. com $= g^{f'(t)}$. We then define f to be the degree-$(B-1)$ polynomial such that $f'(t) = f(t+\tilde{x})$, effectively giving us a commitment in the powers-of-$(t+\tilde{x})$ basis. This implies $\sigma = g^{(b_j - f(t+\tilde{x}))a}$. Furthermore, since E_2 is false, $(j, f, \cdot) \notin Q$.

Suppose the j-th query was never made by the adversary viz. $(j, \cdot, \cdot) \notin Q$, we then have a representation of σ in the basis of $(g, g^t, \ldots, g^{t^{B-1}}, \{g^{b_i}\}_{i\in[m]}, \{g^{(b_i - f_i(t+\tilde{x}))}\}_{i\in[m];i\neq j})$ which implies we have p_i's, β_i's and γ_i's such that

$$\sum_{i=0}^{B-1} p_i \cdot t^i + \sum_{i=1}^{m} \beta_i \cdot b_i + \sum_{i=1, i\neq j}^{m} \gamma_i \cdot (b_i - f_i(t+\tilde{x}))a = (b_j - f(t+\tilde{x}))a.$$

If $\sum_{i\neq j}^{m} \gamma_i \cdot (b_i - f_i(t+\tilde{x})) - (b_j - f(t+\tilde{x})) \neq 0$, then in the case where we set $a \coloneqq z$, we can solve for z as

$$z = \frac{\sum_{i=0}^{B-1} p_i \cdot t^i + \sum_{i=1}^{m} \beta_i \cdot b_i}{(b_j - f(t+\tilde{x})) - \sum_{i=1, i\neq j}^{m} \gamma_i \cdot (b_i - f_i(t+\tilde{x}))}$$

We are then left with the following two equations:

$$\sum_{i\neq j}^{m} \gamma_i \cdot (b_i - f_i(t+\tilde{x})) - (b_j - f(t+\tilde{x})) = 0$$

$$\sum_{i=0}^{B-1} p_i \cdot t^i + \sum_{i=1}^{m} \beta_i \cdot b_i = 0$$

In this case, if we had set $b_i \coloneqq z$, then we can solve for z as

$$z = \sum_{i\neq j}^{m} \gamma_i \cdot (b_i - f_i(t+\tilde{x})) + f(t+\tilde{x}).$$

Therefore, $\mathcal{A}$ cannot win if it didn't make the j-th query and there exists $(j, f_j, \sigma_j) \in Q$. There are now two cases:

1. $f(t+\tilde{x}) = f_j(t+\tilde{x})$. Corresponding to the case where $\sigma = \sigma_j$, but recall that there does not exist $(j, f, \cdot) \in Q$, and hence $f \neq f_j$. In the case where we set $t \coloneqq z$, we can solve for z by finding the roots of
$$f(z+\tilde{x}) - f_j(z+\tilde{x}) = 0.$$

2. $f(t+\tilde{x}) \neq f_j(t+\tilde{x})$. Corresponding to the case $\sigma \neq \sigma_j$. We have a representation of $g^{(b_j - f(t+\tilde{x}))a}$ in the basis of $(g, g^t, \ldots, g^{t^{B-1}}, \{g^{b_i}\}_{i\in[m]}, \{g^{(b_i - f_i(t+\tilde{x})a)}\}_{i\in[m]})$.

$$\sum_{i=0}^{B-1} p_i \cdot t^i + \sum_{i=1}^{m} \beta_i \cdot b_i + \sum_{i=1}^{m} \gamma_i \cdot (b_i - f_i(t+\tilde{x}))a = (b_j - f(t+\tilde{x}))a.$$

If $\sum_{i=1}^{m} \gamma_i \cdot (b_i - f_i(t+\tilde{x})) - (b_j - f(t+\tilde{x})) \neq 0$, then in the case where we set $a := z$, we can solve for z as

$$z = \frac{\sum_{i=0}^{B-1} p_i \cdot t^i + \sum_{i=1}^{m} \beta_i \cdot b_i}{(b_j - f(t+\tilde{x})) - \sum_{i=1}^{m} \gamma_i \cdot (b_i - f_i(t+\tilde{x}))}.$$

On the other hand, suppose $\sum_{i=1}^{m} \gamma_i \cdot (b_i - f_i(t+\tilde{x})) - (b_j - f(t+\tilde{x})) = 0$. Note that the case where all γ_i's are zero except for $\gamma_j = 1$ cannot occur as this would imply $f(t+\tilde{x}) = f_j(t+\tilde{x})$. In all other cases we could have set some $b_{i^*} := z$, where $\gamma_{i^*} \neq 0$, solved for z.

Hence, E_2 must be true if E_1 is true $\implies$ E_2 is true.

Case $E_1 \wedge E_2$**:** Here $\mathcal{A}$ outputs a $\mathsf{com} = g^{f_j(t+\tilde{x})}$ that has the representation of a polynomial $f_j(X)$ that was queried, and used the same σ that it received. If we redefine $t = \tau + \tilde{x}$, we see that com is a KZG [KZG10] polynomial commitment to f_j and the adversary output a proof π that corresponds to the opening of the commitment com at the point $\tilde{x}$ to the value 0. Note that querying such a polynomial $f_j(X)$ is expressly forbidden by the oracle. Hence, $\mathcal{A}$ violated the KZG binding property, which can then be reduced to solving $(B-1)$-dlog as shown in [CHM+20]. □

Theorem 5. *The protocol in Fig. 2 securely emulates the* $\mathcal{F}_{\mathsf{bTPKE}}$ *ideal functionality Fig. 1 in the dealer model against any static PPT adversary A corrupting up to* $t < n/2$ *parties, given any weakly simulation-extractable NIZK, and provided the i-kzg game (Definition 1) is hard in the programmable random oracle model.*

We begin by describing the simulator Sim, who simulated all interactions between the adversary and the honest parties using only the ideal functionality.

Sim remembers all queries made to the Random Oracle. Unless specified, if the query has not been previously made, it lazily samples the response and remembers it. If the query has been made previously, Sim returns the same response.

During the Setup phase, Sim plays the role of the trusted dealer as (i) uses the NIZK simulator to sample $\mathsf{crs}_{\mathsf{NIZK}}$; (ii) samples elements of $\mathsf{crs}_{\mathsf{KZG}}$ honestly; and (iii) samples random field elements as the corrupt parties' shares of the secret key and then sets sk to be a random value. This fully determines the honest parties' shares of the secret key, and hence $h^{[\mathsf{sk}]}_{j\in[n]}$, which Sim publishes. Here, Sim also specifies Γ to be the distribution of an encryption of a random message, under this public key.

For convenience, we will maintain a list of previously seen *receipts* for honest party ciphertexts $\mathcal{L}$. Every time the simulator sees a receipt ct from an honest party, it updates the set as $\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathsf{ct}\}$. When simulating the Encryption phase, there are two cases:

- Every time an honest party encrypts a message, Sim receives a receipt ct (say) from the ideal functionality. If $\mathsf{ct} \in \mathcal{L}$, Sim aborts. Else, Sim adds ct to $\mathcal{L}$ and forwards ct to the adversary.
- For any ciphertext $\mathsf{ct} = (\mathsf{ct}^{(1)}, \mathsf{ct}^{(2)}, \mathsf{ct}^{(3)}, \mathsf{ct}^{(4)}, S, \hat{x}, \Phi)$ created by corrupt parties, Sim verifies the NIZK proof Φ and discards the ciphertext if it fails. Next, it checks whether there exists some $\mathsf{ct}' \in \mathcal{L}$ such that $H(\mathsf{ct}.S) = H(\mathsf{ct}'.S)$ and aborts if so. Otherwise, Sim runs the extractor and receives the randomness (α, β) used by the adversary to create the ciphertext. Sim then extracts the message as $M = \mathsf{ct}^{(1)} \oplus H(e(H(\mathsf{eid}) \cdot g^{-\mathsf{tg}}, h)^{\alpha})$.

Sim receives the entire batch of messages $\{M_1, \ldots, M_B\}$ corresponding to the batch of ciphertexts $\{\mathsf{ct}_1, \ldots, \mathsf{ct}_B\}$ that are decrypted in epoch number eid. Now, Sim computes the degree-$(B-1)$ polynomial $p(X)$ such that $\{p(i) = \mathsf{tg}_i\}_{i\in[B]}$, where $\mathsf{tg}_i := H_F(\mathsf{ct}_i.S)$. It then computes $\mathsf{com} = g^{p(\tau)}$ using $\mathsf{crs}_{\mathsf{kzg}}$ and sets $\delta = H(\mathsf{eid})/\mathsf{com}$. Note that the partial decryptions are fully determined as $\{\sigma_i := \delta^{[\mathsf{sk}]_i}\}_{i\in[B]}$. Sim can compute partial decryptions that should have been sent by honest parties using the shares of the secret key it create during Setup. It then recovers $\sigma = \delta^{\mathsf{sk}}$.

Finally, the simulator must program the random oracle to ensure that the simulated honest party ciphertexts remain consistent with the messages obtained from the ideal functionality. For every *decrypted*

ciphertext ct_i that came as the receipt for an honest party's ciphertext, Sim computes the KZG opening proof π_i as described in $Fig.$ 2 and computes $e(\pi, \mathsf{ct}_i^{(2)}) \cdot e(\delta, \mathsf{ct}_i^{(3)}) \cdot e(\sigma^{-1}, \mathsf{ct}_i^{(4)})$. If this point has been previously queried to the random oracle, Sim aborts. Otherwise, it programs the random oracle output to be $M_i \oplus \mathsf{ct}_i^{(1)}$.

We now describe a sequence of hybrids that are indistinguishable from each other starting from the batch-decryption protocol in Fig. 2 in the real world and ending in the ideal world where a simulator handles all interaction between the parties and the ideal functionality.

$\mathbf{H}_0$: The real world where the adversary interacts with the honest parties who execute the protocol as described in Fig. 2.

$\mathbf{H}_1$: The simulator emulates the trusted dealer and executes the Setup sub protocol faithfully. For all Random Oracle queries made by the adversary, Sim queries the random oracle defined in the protocol and forwards all responses. This hybrid has an identical distribution to the previous hybrid.

$\mathbf{H}_2$: This hybrid is identical to the previous hybrid, except for one change. When a corrupt party creates a ciphertext, Sim first verifies the NIZK proof and discards the ciphertext if it fails. Additionally, it maintains a list $\mathcal{L}$ of ciphertexts created by honest parties and checks if there exists some $\mathsf{ct}' \in \mathcal{L}$ such that $H(\mathsf{ct}.S) = H(\mathsf{ct}'.S)$ and aborts if so. If both the above checks pass, then Sim runs the extractor to obtain the randomness (α, β) used by the adversary to create the ciphertext. Sim can then extract the message as $M = \mathsf{ct}^{(1)} \oplus H(e(H_G(\mathsf{eid}) \cdot g^{-\mathsf{tg}}, h)^{\alpha})$.

We will now argue that Sim aborts with negligible probability. From the knowledge soundness property of the NIZK proof, if the proof verifies, then the statement must be true, and the adversary knows s such that $\mathsf{ct}.S = g^s$, except with negligible probability. Furthermore, we have that $H_F(\mathsf{ct}.S) = H_F(\mathsf{ct}'.S)$. This implies that either (i) $\mathsf{ct}.S = \mathsf{ct}'.S$, or (ii) the adversary has found a collision for the random oracle, which occurs with negligible probability by definition. In the former case, we can use this adversary to solve the discrete logarithm problem.

The reduction embeds the challenge instance from the discrete logarithm game as the S component of some ciphertext created by honest parties. To generate the proof for this ciphertext, we can run SimProve. Since S is a randomly chosen group element in the batch-threshold encryption protocol, the distributions are computationally indistinguishable. With noticeable probability, the adversary will create a ciphertext with the same S as the challenge instance. By running the extractor, we can compute $s = \mathsf{dlog}(S)$ and hence solve the discrete logarithm problem. Hence, $\mathsf{ct}.S \neq \mathsf{ct}'.S$ with overwhelming probability which implies Sim aborts with negligible probability. Thus, this hybrid is computationally indistinguishable from the previous hybrid.

$\mathbf{H}_3$: In this hybrid, Sim begins to program the output of the random oracle as follows. For each eid, Sim samples a random field element b_{eid} and sets $H_G(\mathsf{eid}) = g^{b_{\mathsf{eid}}}$. For all other queries, Sim lazily samples a response to the random oracle query. This hybrid has an identical distribution to the previous hybrid.

$\mathbf{H}_4$: This hybrid is identical to the previous hybrid, except Sim aborts when $\mathcal{A}$ queries the random oracle on a *bad* point which allows it to break the semantic security of honest party ciphertexts.

More formally, for each ciphertext ct created by an honest party for epoch eid, let (α, β) denote the randomness used to create the ciphertext and define $\mathcal{P}_{\mathsf{ct}} = e(H_G(\mathsf{eid}) \cdot g^{-\mathsf{tg}}, h)^{\alpha}$, where $\mathsf{tg} = H_F(\mathsf{ct}.S)$. Let $\mathcal{B}_{\mathsf{eid}} = \{\mathsf{ct}_1, \ldots, \mathsf{ct}_B\}$ denote the batch of ciphertexts that are decrypted in epoch number eid. If a batch decryption for that particular epoch has not occurred yet, then we set $\mathcal{B}_{\mathsf{eid}} = \emptyset$. We can then define the set $\mathcal{P}_{\mathsf{eid}} = \{\mathcal{P}_{\mathsf{ct}} \mid \mathsf{ct} \in \mathcal{L}_{\mathsf{eid}} \setminus \mathcal{B}_{\mathsf{eid}}\}$, where $\mathcal{L}_{\mathsf{eid}}$ is the set of honest party ciphertexts encrypted to epoch eid. If $\mathcal{A}$ queries the random oracle at any point in $\mathcal{P}_{\mathsf{eid}}$, Sim aborts. We will now argue that Sim aborts with negligible probability.

Claim 6. *If the i-kzg game (Definition 1) is hard, then for all PPT adversaries $\mathcal{A}$, and any epoch* eid, $\Pr[\mathcal{A} \to p \mid p \in \mathcal{P}_{\mathsf{eid}}] \leq \mathsf{negl}(\lambda)$.

Proof. Suppose there exists an adversary $\mathcal{A}$ corrupting up to $t < n/2$ committee members and an arbitrary number of users. If $\mathcal{A}$ queries a point $p \in \mathcal{P}_{\mathsf{eid}}$, for *any* epoch eid with non-negligible probability, then we can use $\mathcal{A}$ to construct Sim' that wins the i-kzg game with non-negligible probability. The high-level idea is to embed the instance from the i-kzg game in a randomly chosen ciphertext $\tilde{\mathsf{ct}}$ created by an honest party. With some non-negligible probability, $\mathcal{A}$ will query the point $\mathcal{P}_{\tilde{\mathsf{ct}}}$ corresponding to $\tilde{\mathsf{ct}}$, as there are a polynomial (in security parameter) number of epochs and ciphertexts. We will then use $\mathcal{P}_{\tilde{\mathsf{ct}}}$ to produce a winning output in the i-kzg game. We now describe Sim'.

Let the challenge instance in the i-kzg game be

$$(\tilde{x}, g, g^t, g^{t^2}, \ldots, g^{t^B}, \{g_i^b\}_{i\in[m]}, h, h^t, h^a, h^v, h^{tu}, h^{u+av}),$$

with access to an oracle $\mathsf{O}^{a,t,\tilde{x},\{b_i\}_{i\in[m]}}$ and let $\mathcal{C}$ be the set of committee members corrupted by the adversary.

During setup, Sim' sets $\tau = t + \tilde{x}$. It can then compute $\mathsf{crs}_{\mathsf{kzg}} = (g, g^\tau, \ldots, g^{\tau^B}, h, h^\tau)$ using linear operations over $(g, g^t, g^{t^2}, \ldots, g^{t^B}, h, h^t)$. Sim' sets the public key of the committee to be h^a, and samples random values $\{[a]_i\}_{i\in\mathcal{C}}$ for the shares of corrupt parties. Also note that h^a and $\{h^{[a]_i}\}_{i\in\mathcal{C}}$, can be used to compute $\{h^{[a]_i}\}_{i\in[n]\setminus\mathcal{C}}$ using linear interpolation such that $\{[a]_i\}_{i\in[n]}$ forms a (t, n)-secret sharing of a. Finally, Sim' runs the setup for NIZK and publishes $\{\mathsf{crs}_{\mathsf{KZG}}, \mathsf{crs}_{\mathsf{NIZK}}, \{h^{[a]_i}\}_{i\in[n]}\}$ and sends $\{[a]_i\}_{i\in\mathcal{C}}$ to the corrupt parties.

Let $\mathcal{M}$ be the space of messages. Sim' samples $s \leftarrow\!\!\$\ \mathbb{F}$ and computes $S \leftarrow g^s$. It then sets $\tilde{\mathsf{ct}}$ to be:

- $\tilde{\mathsf{ct}}^{(1)} \leftarrow\!\!\$\ \{0,1\}^{|\mathcal{M}|}$
- $\tilde{\mathsf{ct}}^{(2)} \coloneqq h^{tu}$
- $\tilde{\mathsf{ct}}^{(3)} \coloneqq h^{u+av}$
- $\tilde{\mathsf{ct}}^{(4)} \coloneqq h^{v}$
- S
- $\tilde{x}$
- $\tilde{\Phi} = \mathsf{NIZK.SimProve}\{\alpha, \beta, s \mid S = g^s \wedge \mathsf{ct}^{(1)} \wedge \mathsf{ct}^{(2)} = h^{\alpha(\tau-\tilde{x})} \wedge \mathsf{ct}^{(3)} = h^{\alpha}\mathsf{pk}^{\beta} \wedge \mathsf{ct}^{(4)} = h^{\beta}\}$

By defining $\alpha \coloneqq u$, $\beta \coloneqq v$, it is easy to see that the statement Φ attests to is indeed in the language, but Sim' does not know the witness values α, β and hence simulates the proof.

Let $C(\lambda)$ be an upper bound on the number of honest party ciphertexts the adversary sees, which depends on the run time of the adversary. Sim' samples $j \leftarrow\!\!\$\ [C(\lambda)]$ and replaces the j-th ciphertext created by an honest party with $\tilde{ct}$.

For random oracle queries of the form $H_G(\mathsf{eid})$, where eid is the epoch id, Sim' sets $H_G(\mathsf{eid}) = g^{b_{\mathsf{eid}}+\tilde{\mathsf{tg}}}$, where $\tilde{\mathsf{tg}} \leftarrow H_F(\tilde{ct}.S)$. We choose m large enough, based on the run time of the adversary, to ensure Sim' can respond to all queries.

For each batch decryption, Sim' simulates partial decryptions as follows. Let $\mathcal{B}_{\mathsf{eid}} = \{\mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_B\}$ be the set of ciphertexts that are decrypted in epoch eid. If $\tilde{\mathsf{ct}} \in \mathcal{B}_{\mathsf{eid}}$, then Sim' aborts and restarts $\mathcal{A}$. Now, Sim' queries the oracle as $\mathsf{O}^{a,t,\tilde{x},\{b_i\}_{i\in[m]}}(\mathsf{eid}, f_{\mathsf{eid}}(X) - \tilde{\mathsf{tg}})$, where $f_{\mathsf{eid}}(X)$ is the degree-$(B-1)$ polynomial such that $f_{\mathsf{eid}}(\mathsf{ct}_i.\hat{x}) = \mathsf{tg}_i$, where $\mathsf{tg}_i = H_F(\mathsf{ct}_i.S)$. In response, it receives $\sigma \coloneqq (H(\mathsf{eid})/g^{f_{\mathsf{eid}}(t+\tilde{x})})^a = (g^{b_{\mathsf{eid}}+\tilde{\mathsf{tg}}-f_{\mathsf{eid}}(\tau)})^a$. Note that it could have also received $\perp$ if $f_{\mathsf{eid}}(\tilde{x}) = \tilde{\mathsf{tg}}$, but this can only happen in two cases i) $\tilde{\mathsf{ct}}$ was included in $\mathcal{B}_{\mathsf{eid}}$ or ii) the adversary created a ciphertext ct^* such that $H_F(\mathsf{ct}^*.S) = H_F(\tilde{\mathsf{ct}}.S)$. In the former case, Sim' would have already aborted and restarted $\mathcal{A}$. The latter case occurs with negligible probability, as argued in earlier hybrids. Hence, Sim' does not receive $\perp$, except with a negligible probability. Sim' can now simulate all partial decryptions in a straightforward manner using $\{[a]_i\}_{i\in\mathcal{C}}$.

If $\mathcal{A}$ terminates before Sim' has replaced a ciphertext, then Sim' restarts $\mathcal{A}$ and tries again. If Sim' manages to replace a ciphertext before $\mathcal{A}$ terminates, then Sim' randomly samples one of the queries made by $\mathcal{A}$ to the random oracle and outputs it as the response to the i-kzg game. This is a winning output with non-negligible probability, as the adversary has queried a point in $\mathcal{P}_{\mathsf{eid}}$ with non-negligible probability. Thus, Sim' wins the i-kzg game with non-negligible probability, which contradicts the hardness of the i-kzg game. □

From Claim 6 Sim aborts with negligible probability provided the i-kzg game is hard. Thus, this hybrid is computationally indistinguishable from the previous hybrid.

$\mathbf{H}_5$: This hybrid is identical to the previous hybrid, except Sim now simulates *all* honest party ciphertexts as described in the previous hybrid. When an honest party's ciphertext $\tilde{\mathsf{ct}}$ is included in the batch of ciphertexts decrypted in that epoch, Sim programs the random oracle output to be $\tilde{M} \oplus \tilde{\mathsf{ct}}.\mathsf{ct}^{(1)}$, where $\tilde{M}$ is the message the honest party received as input. The distribution of *simulated* honest party ciphertexts $\tilde{\mathsf{ct}}$ is identical to the distribution of *honestly* generated ciphertexts, provided the point $\mathcal{P}_{\tilde{\mathsf{ct}}}$ is never queried by the adversary. Using a similar argument as in the previous hybrid, $\mathcal{A}$ queries such points with negligible probability. Thus, this hybrid is computationally indistinguishable from the previous hybrid.

$\mathbf{H}_6$: This hybrid is identical to the previous hybrid, except Sim computes the point to program using a different strategy. Recall that in previous hybrids Sim controlled honest parties and hence has access to the randomness used to create ciphertexts. In this hybrid, Sim uses its knowledge of the secret key to simulate partial decryptions by faithfully following the protocol. This hybrid has an identical distribution to the previous hybrid.

$\mathbf{H}_7$: In the final hybrid, Sim does not control any honest parties and only handles interaction between the ideal functionality and the adversary. During setup Sim runs the setup sub-protocol from Fig. 2 faithfully to, and specifies the distribution Γ to be the encryption of a randomly chosen message under the resulting public key. Instead of simulating honest party ciphertexts as in previous hybrids, it simply forwards the receipts from the ideal functionality to the adversary. During batch decryption, Sim receives the messages corresponding to the batch of ciphertexts. Here it programs the random oracle *after* receiving the messages as described in previous hybrids.

8 Acknowledgements

This work is supported in part by AFOSR Award FA9550-24-1-0156, and research grants by Bakar Fund, J. P. Morgan Faculty Research Award, Supra Inc., Visa Inc, and the Stellar Development Foundation. Guru-Vamsi Policharla is supported in part by a Berkeley Center for Responsible, Decentralized Intelligence (RDI) Fellowship. Mingyuan Wang is supported in part by an NYU startup fund.

References

[ac22] arkworks contributors. `arkworks` zksnark ecosystem. https://arkworks.rs, 2022. 13

[ADM+24] Gennaro Avitabile, Nico Döttling, Bernardo Magri, Christos Sakkas, and Stella Wohnig. Signature-based witness encryption with compact ciphertext. Cryptology ePrint Archive, Paper 2024/1477, 2024. 4

[AEC21] Guillermo Angeris, Alex Evans, and Tarun Chitra. A note on bundle profit maximization. *Stanford University*, 2021. 2

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Cham, August 2018. 4

[BC16] Olivier Blazy and Céline Chevalier. Structure-preserving smooth projective hashing. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 339–369. Springer, Berlin, Heidelberg, December 2016. 5, 11, 15

[BDF21] Carsten Baum, Bernardo David, and Tore Kasper Frederiksen. P2DEX: Privacy-preserving decentralized cryptocurrency exchange. In Kazue Sako and Nils Ole Tippenhauer, editors, *ACNS 21International Conference on Applied Cryptography and Network Security, Part I*, volume 12726 of *LNCS*, pages 163–194. Springer, Cham, June 2021. 4

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Berlin, Heidelberg, August 2001. 5

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Berlin, Heidelberg, August 2001. 4

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 500–532. Springer, Cham, April / May 2018. 5

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Berlin, Heidelberg, December 2001. 5

[BO22] Joseph Bebel and Dev Ojha. Ferveo: Threshold decryption for mempool privacy in BFT networks. Cryptology ePrint Archive, Report 2022/898, 2022. 2, 3, 4, 13

[CD24] Ignacio Cascudo and Bernardo David. Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 216–248. Springer, Cham, May 2024. 4, 13

[CG99] Ran Canetti and Shafi Goldwasser. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 90–106. Springer, Berlin, Heidelberg, May 1999. 3

[CGPP24] Arka Rai Choudhuri, Sanjam Garg, Julien Piet, and Guru-Vamsi Policharla. Mempool privacy via batched threshold encryption: Attacks and defenses. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024. 3, 4, 5, 6, 7, 9, 10, 13, 14

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Cham, May 2020. 19

[CIMI+21] Michele Ciampi, Muhammad Ishaq, Malik Magdon-Ismail, Rafail Ostrovsky, and Vassilis Zikas. Fairmm: A fast and frontrunning-resistant crypto market-maker. *IACR Cryptology ePrint Archive*, 2021. 4

[CJW22] Agostino Capponi, Ruizhe Jia, and Ye Wang. The evolution of blockchain: from lit to dark. *arXiv preprint*, 2022. 2

[CKLS02] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 88–97. ACM Press, November 2002. 14

[dcc20] dalek-cryptography contributors. merlin. https://github.com/dalek-cryptography/merlin, 2020. 13

[DDFY94] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *26th ACM STOC*, pages 522–533. ACM Press, May 1994. 3

[DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, New York, August 1990. 3

[DGK+20] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy*, pages 910–927. IEEE Computer Society Press, May 2020. 2

[DHMW23] Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wohnig. McFly: Verifiable encryption to the future made practical. In Foteini Baldimtsi and Christian Cachin, editors, *FC 2023, Part I*, volume 13950 of *LNCS*, pages 252–269. Springer, Cham, May 2023. 3, 4, 5, 13

[DJ97] Yvo Desmedt and Sushil Jajodia. Redistributing secret shares to new access structures and its applications. Technical report, Citeseer, 1997. 14

[ElG86] Taher ElGamal. On computing logarithms over finite fields. In Hugh C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, pages 396–402. Springer, Berlin, Heidelberg, August 1986. 3, 4

[EMC19] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: front-running attacks on blockchain. In *International Conference on Financial Cryptography and Data Security*, 2019. 2

[FHAS24] Nils Fleischhacker, Mathias Hall-Andersen, and Mark Simkin. Extractable witness encryption for KZG commitments and efficient laconic OT. Cryptology ePrint Archive, Report 2024/264, 2024. 6

[FK23] Dankrad Feist and Dmitry Khovratovich. Fast amortized KZG proofs. Cryptology ePrint Archive, Report 2023/033, 2023. 11, 14

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Cham, August 2018. 7, 9

[Fra90] Yair Frankel. A practical protocol for large group oriented networks. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 56–61. Springer, Berlin, Heidelberg, April 1990. 3

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013. 4

[GHL22] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. Practical non-interactive publicly verifiable secret sharing with thousands of parties. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 458–487. Springer, Cham, May / June 2022. 4, 13

[GKM+22] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 252–282. Springer, Cham, March 2022. 14

[GKPW24] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold encryption with silent setup. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024*, pages 352–386, Cham, 2024. Springer Nature Switzerland. 5, 11, 15

[GKW+16] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016. 2

[Gro21] Jens Groth. Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Report 2021/339, 2021. 4, 13

[GS18] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 468–499. Springer, Cham, April / May 2018. 5

[HJKY95] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 339–352. Springer, Berlin, Heidelberg, August 1995. 14

[HW22] Lioba Heimbach and Roger Wattenhofer. Eliminating sandwich attacks with the help of game theory. *arXiv preprint*, 2022. 4

[JSSW21] Aljosha Judmayer, Nicholas Stifter, Philipp Schindler, and Edgar R. Weippl. Estimating (miner) extractable value is hard, let's go shopping! *IACR Cryptology ePrint Archive*, 2021. 2

[Kat24] Jonathan Katz. Round-optimal, fully secure distributed key generation. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024*, pages 285–316, Cham, 2024. Springer Nature Switzerland. 4, 13

[KDL+21] Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. Themis: Fast, strong order-fairness in byzantine consensus. *IACR Cryptology ePrint Archive*, 2021. 4

[KLJD23] Alireza Kavousi, Duc V. Le, Philipp Jovanovic, and George Danezis. Blindperm: Efficient mev mitigation with an encrypted mempool and permutation. Cryptology ePrint Archive, Paper 2023/1061, 2023. https://eprint.iacr.org/2023/1061. 2

[KMM+23] Aniket Kate, Easwar Vivek Mangipudi, Pratyay Mukherjee, Hamza Saleem, and Sri Aravinda Krishnan Thyagarajan. Non-interactive VSS using class groups and application to DKG. Cryptology ePrint Archive, Report 2023/451, 2023. 4, 13

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part V*, volume 14442 of *LNCS*, pages 407–441. Springer, Singapore, December 2023. 5

[Kur20] Klaus Kursawe. Wendy, the good little fairness widget: Achieving order fairness for blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, 2020. 4

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Berlin, Heidelberg, December 2010. 6, 11, 19

[KZGJ20] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *International Cryptology Conference*, 2020. 4

[MDF22] Conor McMenamin, Vanesa Daza, and Matthias Fitzi. Fairtradex: A decentralised exchange preventing value extraction. *arXiv preprint*, 2022. 4

[MGZ22] Peyman Momeni, Sergey Gorbunov, and Bohan Zhang. Fairblock: Preventing blockchain front-running with minimal overheads. In *SecureComm*, volume 462 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 250–271. Springer, 2022. 3, 4, 13

[MS22] Dahlia Malkhi and Pawel Szalachowski. Maximal extractable value (mev) protection on a dag, 2022. 2

[MZW+19] Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. CHURP: Dynamic-committee proactive secret sharing. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2369–2386. ACM Press, November 2019. 14

[NRBB24] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. Powers-of-tau to the people: Decentralizing setup ceremonies. In Christina Pöpper and Lejla Batina, editors, *Applied Cryptography and Network Security*, pages 105–134, Cham, 2024. Springer Nature Switzerland. 11

[PNS23] Julien Piet, Vivek Nair, and Sanjay Subramanian. Mevade: An mev-resistant blockchain design. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9, 2023. 2, 4

[QZG21] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? *arXiv preprint*, 2021. 2

[RK23] Antoine Rondelet and Quintus Kilbourn. Threshold encrypted mempools: Limitations and considerations, 2023. 2

[Sch99] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 148–164. Springer, Berlin, Heidelberg, August 1999. 4, 13

[Shu21] Shutter Network contributors. The shutter network. `https://shutter.network`, 2021. 2, 3, 4, 13

[SLL08] David A. Schultz, Barbara Liskov, and Moses Liskov. Mobile proactive secret sharing. In Rida A. Bazzi and Boaz Patt-Shamir, editors, *27th ACM PODC*, page 458. ACM, August 2008. 14

[TC+21] Christof Ferreira Torres, Ramiro Camino, et al. Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain. In *30th USENIX Security Symposium*, 2021. 2

[VAFB22] Robin Vassantlal, Eduardo Alchieri, Bernardo Ferreira, and Alysson Bessani. COBRA: Dynamic proactive secret sharing for confidential BFT services. In *2022 IEEE Symposium on Security and Privacy*, pages 1335–1353. IEEE Computer Society Press, May 2022. 14

[WCDW21] Ye Wang, Yan Chen, Shuiguang Deng, and Roger Wattenhofer. Cyclic arbitrage in decentralized exchange markets. *Available at SSRN 3834535*, 2021. 2

[WWW02] T.M. Wong, Chenxi Wang, and J.M. Wing. Verifiable secret redistribution for archive systems. In *First International IEEE Security in Storage Workshop, 2002. Proceedings.*, pages 94–105, 2002. 14

[ZQC+21] Liyi Zhou, Kaihua Qin, Antoine Cully, Benjamin Livshits, and Arthur Gervais. On the just-in-time discovery of profit-generating transactions in defi protocols. In *2021 IEEE Symposium on Security and Privacy (SP)*, 2021. 2

[ZQG21] Liyi Zhou, Kaihua Qin, and Arthur Gervais. A2mm: Mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges. *arXiv preprint*, 2021. 4

[ZQT+21] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. In *2021 IEEE Symposium on Security and Privacy (SP)*, 2021. 2

[ZSVR05] Lidong Zhou, Fred B. Schneider, and Robbert Van Renesse. Apss: proactive secret sharing in asynchronous systems. *ACM Trans. Inf. Syst. Secur.*, 8(3):259–286, aug 2005. 14