Mempool Privacy via Batched Threshold Encryption: Attacks and Defenses

Arka Rai Choudhuri[1], Sanjam Garg[2], Julien Piet[2], and Guru-Vamsi Policharla[2]

[1]NTT Research
[2]University of California, Berkeley

Abstract

With the rising popularity of DeFi applications it is important to implement protections for regular users of these DeFi platforms against large parties with massive amounts of resources allowing them to engage in market manipulation strategies such as frontrunning/backrunning. Moreover, there are many situations (such as recovery of funds from vulnerable smart contracts) where a user may not want to reveal their transaction until it has been executed. As such, it is clear that preserving the privacy of transactions in the mempool is an important goal.

In this work we focus on achieving mempool transaction privacy through a new primitive that we term *batched-threshold encryption*, which is a variant of threshold encryption with strict efficiency requirements to better model the needs of resource constrained environments such as blockchains. Unlike the naive use of threshold encryption, which requires communication proportional to $O(nB)$ to decrypt B transactions with a committee of n parties, our batched-threshold encryption scheme only needs $O(n)$ communication. We additionally discuss pitfalls in prior approaches that use (vanilla) threshold encryption for mempool privacy.

To show that our scheme is concretely efficient, we implement our scheme and find that transactions can be encrypted in under 6 ms, *independent* of committee size, and the communication required to decrypt an entire batch of B transactions is 80 bytes per party, *independent* of the number of transactions B, making it an attractive choice when communication is very expensive. If deployed on Ethereum, which processes close to 500 transaction per block, it takes close to 2.8 s for each committee member to compute a partial decryption and under 3.5 s to decrypt all transactions for a block in single-threaded mode.

1 Introduction

A prominent application of cryptocurrencies and blockchain technologies is in Decentralized Finance (DeFi) which aims to provide financial services through a decentralized network of parties. A significant amount of capital has been invested in this space with a peak value peak of over $250 billion in December 2021.[1] This capital was drawn not just from retail investors but also from major institutional players.

The traditional financial system has well documented evidence [43] to prove that in the absence of regulations, market manipulation strategies will be employed by insiders who seek to financially exploit the average consumer. However, despite the large amounts of capital locked in the DeFi space, there is virtually no protections implemented for consumers against miners/builders who can not only see the details of all transactions submitted by clients to a *memory pool* (mempool) but also control the order in which transactions will be executed.

Although cryptocurrencies promise a fair and permissionless platform where all users are on a level playing field, the reality is far from this. Unsurprisingly, the DeFi space is ripe for exploitation by greedy (although arguably rational) miners who can frontrun/backrun/sandwich client trades to extract additional value from the asymmetric information available to them. This was first documented as "Miner Extractable Value" or MEV in [17]. Early work focused on devising the best strategies to implement MEV [1, 25, 60, 62, 64]. Many studies since have shown it to be a widespread problem in blockchains [14, 31, 36, 50, 59]. In particular, an estimated 200,000,000 USD were lost on Ethereum in 2021 alone, mostly benefiting miners [47]. In the case of trading on Decentralized Exchanges, the attacks mentioned above allow a miner to extract financial gains at the expense of the users.

Alternatively, consider the situation where a user discovers a bug in their smart contract that allows any party to drain all of its funds. A natural course of action is to *update* the smart contract with new code. But the diff between the old and new code could reveal the bug to any party with access to the mempool, allowing an attacker to exploit the bug before it is fixed. This class of attack is actually widespread, and has been

[1] https://cointelegraph.com/news/defi-needs-to-start-creating-real-world-value-if-it-wants-to-survive

observed and documented [51, 53]. In fact, with the recent advancements in Large Language Models, researchers have shown that fine-tuning can be carried out to identify bugs in smart contracts [19]. Using the same strategy, it is conceivable that an attacker could also automate this process. Finally, revealing the content of transactions allow malicious miners to selectively censor targeted groups of users or transactions.

Protecting the privacy of transactions in the mempool has been a subject of active discussion [5, 39, 45, 48, 52, 57] and *threshold public key encryption* [20, 22, 27] has emerged as the de facto solution to this problem. A threshold public key encryption system has a single public key, and distributes the private key among n decryption servers such that at least a threshold number (say k) are needed for decryption. At a high level, threshold encryption is used as follows. First, a committee - corresponding to the decryption servers - is set up such that the committe members are in possession of the (distributed) private key, and all users encrypt to the committee's public key. Miners can only see encrypted transactions in the mempool and create blocks containing ciphertexts. After the block has been confirmed, the committee jointly decrypts all transactions that have been included and announces them to all nodes in the network.

Existing solutions are insufficient to solve the problem of mempool privacy for two main reasons. First, the proposed solutions fail to account for a malleability attack, where an adversary could create a *related* transaction ciphertext, submit that instead, and learn information about a transaction which has not been included on chain. This is a very realistic attack as most users don't run their own nodes and instead rely on other untrusted nodes who submit transactions for them.

Second, the communication involved to decrypt ciphertexts is $O(nB)$ instead of $O(B)$ without privacy, where n is the committee size and B is the block size (number of transactions in a block). In a widely distributed network like blockchains, the additional latency introduced in an already time-critical system is untenable. We emphasize here that the metric of interest is not the bandwidth available on a node but the latency of the underlying broadcast/gossip network. In otherwords, it is not how fast a committee member can upload their partial decryptions to some server but how fast a message broadcast by a committee member can reach *all nodes* on the network, as all parties need to decrypt the ciphertexts. For concrete numbers, we refer to [21], which shows that each kilobyte over 20kB adds an 80ms delay to the time it takes for a message to reach a majority of nodes.

Deployed systems [57] have attempted to handle this by introducing a new encryption scheme similar to that of [23], where clients encrypt to an *epoch*, and the decryption key is the same for all ciphertexts in that epoch. However, this makes unacceptable compromises to privacy, as ciphertexts that are not included/withheld lose privacy despite not being confirmed transactions. We refer to this property as *pending transaction privacy*. We summarize a comparison against prior work in Table 1 and elaborate on other possible attack vectors in Section 3.

Our Model. To address the above two problems we introduce a new security notion of *batched-threshold-encryption* which protects the privacy of messages against adaptive chosen ciphertext attacks [7] and in addition demands that an entire batch of ciphertexts can be decrypted *non-interactively*, and with *sub-linear communication*. This translates to ensuring that (i) each member of the committee can locally compute partial decryption given only the batch of ciphertexts, and these can later be combined by any party (even outside the committee) to recover the messages; and (ii) the size of partial decryption is sub-linear in the batch size.

Scheme	Comm.	Pending Tx privacy	Setup
Thresh. Enc. [13, 24]	$O(nB)$	✓	DKG
Ferveo [5]	$O(nB)$	✓	DKG
McFly [23]	$O(n)$	✗	DKG
Shutter [57]	$O(n)$	✗	DKG
This Work	$O(n)$	✓	MPC + Epoch Setup

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 which were submitted but may not have been included in the batch. DKG refers to a distirbuted 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.

1.1 Our Contributions

In this work,

- We demonstrate the inefficacy of existing approaches in protecting the privacy of transactions in the mempool, highlighting the need for improved security definitions and constructions. In particular, we show that the Shutter Network [57] suffers from the issues highlighted in the prior section. (Section 3)

- We undertake a formal study of encrypted mempools and propose the notion of batched-threshold encryption, which we argue is the appropriate security notion of threshold encryption for mempool privacy. Additionally, our definitions demands efficiency properties that aim

to model the limited bandwidth available in large-scale distributed networks. (Section 6)

- We then provide the first *concretely* efficient construction satisfying the *batched-threshold encryption* definition. Importantly, to decrypt a batch of messages, the size of partial decryption is *independent* of the batch size; in fact, it is just one field element. The ciphertext size is 370 bytes for a 32 byte message, and only $\approx 3\times$ larger than the Cramer-Shoup threshold encryption scheme [12, 13], which does not satisfy necessary efficiency requirements. (Section 7). However, our scheme incurs a more expensive setup ($\approx 100\times$) along with a per epoch setup, which can be carried out well before the actual epoch. We believe that this can be a favorable trade-off when the number of transactions in a block is large, as our scheme reduces the communication during the time-sensitive phase (decryption) by several orders of magnitude. Concretely, for a block of 512 transactions and a committee size of 128, we reduce the communication by over $300\times$. Moreover, this number will only grow larger as the number of transactions in a block increases. (Section 7)

- We created a Rust crate for our batched-threshold encryption scheme. It takes < 6 ms to encrypt a ciphertext. For an Ethereum deployment, which processes ≈ 500 transactions per block, it would take close to 2.8 ms to compute a partial decryption and < 3.5 s to reconstruct all messages. We emphasize that our scheme can take advantage of multi-threaded architectures and Ethereum nodes are likely to already have machines with multiple cores. (Section 9)

2 Our Techniques

We now describe the main ideas underlying our work. Before we describe our construction, we note that a common theme when describing the shortcomings of prior approaches was the lack of clearly defined security properties. Further, the security needs to be defined while ensuring that any solution we propose is agnostic to the system (or blockchain) it is deployed on. To this end, we define a new primitive called *batched threshold public key encryption* (bTPKE) whose properties (to be defined shortly) will prove sufficient in achieving mempool transaction privacy. We elaborate below.

Batched Threshold Public Key Encryption At a high level, this primitive specifies a public key for which some parties that are designated as servers are given (threshold) shares of the corresponding secret key. Users, who are not necessarily servers, are then allowed to encrypt to the public key such that: (i) a batch of B ciphertext are chosen by the servers to be decrypted; (ii) each server broadcasts a short *batch decryption key* (of size independent of B) that enables decryption of all ciphertexts included in the batch; and (iii) the contents the ciphertexts *not* included in the batch remain hidden.

Given the properties described for a bTPKE, it follows in a straightforward manner how one can utilize the primitive to realize the the guarantees for mempool privacy. Specifically, in the mempool privacy setting, the miners correspond to the bTPKE servers, and users encrypt transactions using the bTPKE encryption scheme with the guarantee that encrypted transactions that are not selected by miners in the batch hide the underlying transaction. Further, the efficiency properties of bTPKE ensure that this is indeed a feasible solution to guarantee mempool transaction privacy. Note that, as desired, the properties of the bTPKE are agnostic to how the batch of ciphertexts to be decrypted are decided, and thus work for *any* set of selected transactions.

In Section 6 we formalize the above described properties via an ideal functionality $\mathcal{F}_{\mathsf{bTPKE}}$, which is motivated by the ideal functionality for threshold encryption schemes that are secure against chosen ciphertext attack [13]. The main difference in our setting is that our ideal functionality incorporates a notion of "epochs" such that users can encrypt messages to a specific epoch, and a batch of ciphertexts to be decrypted must all belong to the same epoch.

Construction For simplicity, in this overview, we will highlight the main ideas underlying our bTPKE construction in the setting that there is a *single server* that is honest. We defer the extension to the threshold setting with multiple servers to Section 7, where we use the threshold to emulate the single honest server. As a further simplification, we first describe a scheme that provides the weak form of security where privacy is guaranteed only against adversaries that only see ciphertexts (indistinguishability of chosen plaintext), but *not* their decryption. We will then describe the shortcomings of this construction, and then strengthen it to achieve the desired security guarantees.

Our construction is based on *polynomial-commitment schemes*, where such a scheme allows a user to generate a "commitment" com to a polynomial $\mathsf{p}(X)$ such that the user at a later point can provide a short *proof of opening* with respect to com to pairs (x, y) given the polynomial p with the guarantee that $\mathsf{p}(x) = y$. We now describe the template of our bTPKE construction: (i) the *public-key* will correspond to the commitment com of an unspecified degree B polynomial, i.e. the polynomial will be determined by the transactions and the reader can think of com as a commitment to an arbitrary polynomial that will be *explained* correctly later; (ii) each user will sample a pair (x_i, y_i) and "encrypt" their message m_i to only be decrypted using the proof of opening of (x_i, y_i) with respect to com; and (iii) finally, the server given B ciphertexts, each with their corresponding (x_i, y_i) pair, uses a *trapdoor* to explain com to be a commitment to a degree B polynomial $\mathsf{p}(X)$ such that for each $i \in [B]$ $\mathsf{p}(x_i) = y_i$ - since a degree B polynomial is fully determined by $B+1$ points, the server can simply broadcast a single pair $(0, \mathsf{p}(0))$ which allows all par-

ties to reconstruct $p(X)$ via polynomial interpolation using the points $\{(0,\mathsf{p}(0)),(x_1,y_1),\cdots,(x_B,y_B)\}$. Once $\mathsf{p}(X)$ is reconstructed, as discussed from the properties of the polynomial commitment scheme, anyone with access to the polynomial $\mathsf{p}(X)$ can compute the proof of openings to (x_i,y_i) and thus decrypt the ciphertexts. We now fill in the details of the template using the KZG polynomial commitment scheme [38]. We first briefly describe the commitment scheme below.

In the KZG scheme, the commitment to a degree B polynomial $\mathsf{p}(X)$ given the common reference string $\mathsf{crs} = (g, g^{\tau}, g^{\tau^2}, \cdots, g^{\tau^B})$ is simply the *deterministic* value $\mathsf{com} = g^{\mathsf{p}(\tau)}$. Here g is the generator of a multiplicative group $\mathbb{G}$, and the committer is able to compute $g^{\mathsf{p}(\tau)}$ given the crs, without knowledge of τ - which is the trapdoor hidden from the committer. Given a pair (x,y) such that $y = \mathsf{p}(x)$, the *proof of opening* is a group element $\pi = g^{\mathsf{q}(\tau)}$ that is a commitment to q, which is a polynomial such that $\mathsf{q}(X) = (\mathsf{p}(X) - y)/(X - x)$. If the group is appropriately equipped with a pairing operation $e(\cdot,\cdot)$ (see Section 5 for details on groups and pairing operations), verification of the *proof of opening* is done by checking the equality $e(\mathsf{com}/g^y, g) = e(\pi, g^{\tau}/g^x)$. The binding property of the scheme guarantees that if τ is hidden from the committer, for any pair (x',y') such that $\mathsf{p}(x') \neq y'$, one *cannot* find a group element π' that satisfies the aforementioned verification check.

Given the discussed properties from the KZG polynomial commitment scheme, specifically the verification check, we use ideas present in the construction of the Boneh-Franklin Identity Based Encrytion (IBE) scheme [11] (also recently used in the context of encryption under signature verification [23, 44]). Specifically, the public key is the CRS $(g, g^{\tau}, g^{\tau^2}, \cdots, g^{\tau^B})$ and a random group element $\mathsf{com} = g^{\alpha}$ treated as a "commitment" to a polynomial yet to be determined. The server is in possession of the trapdoors α and τ. To encrypt a message M with respect to the public key com, a user first samples a pair (x,y), and then computes the ciphertext as

$$\mathsf{ct} = (x, y,\ (g^{\tau}/g^x)^{\rho},\ H(e(\mathsf{com}/g^y, g))^{\rho} \oplus M)$$

where H is a hash function (modeled as a random oracle in our security proofs), and ρ is a randomly sampled value. Given B ciphertexts $\{(x_i, y_i, c_{i,1}, c_{i,2})\}_{i\in[B]}$[2], the server explains the commitment com by interpolating the *unique* degree B polynomial $\mathsf{p}(X)$ using the $B+1$ points $(x_1,y_1),\cdots,(x_B,y_B)$ and (τ,α) (where we view $\alpha = \mathsf{p}(\tau)$ by treating com as a KZG commitment). The server then simply broadcasts the batch decryption key $(0,\mathsf{p}(0))$.

To decrypt, any user in possession of the B ciphertexts $\{(x_i, y_i, c_{i,1}, c_{i,2})\}_{i\in[B]}$ *and* $(0,\mathsf{p}(0))$, performs the following steps in order: (i) reconstructs the polynomial $\mathsf{p}(X)$ using the $B+1$ points in its possession; for each $i \in [B]$, (ii) computes the proof of opening π_i with respect to com and the pair (x_i, y_i); and (iii) compute m_i as $H(e(\pi, c_{i,2})) \oplus c_{i,1}$. From the correctness of the KZG verification, it is easy to see that the decryption procedure does indeed reconstruct the B encrypted messages. Further, the server was only required to send 2 elements to enable the decryption of a batch of B ciphertexts.

Any ciphertext with $(\widetilde{x},\widetilde{y})$ not selected as a part of the B ciphertexts will (with high probability) be such that $\mathsf{p}(\widetilde{x}) \neq \widetilde{y}$ if $(\widetilde{x},\widetilde{y})$ is sampled randomly. Thus, by the binding property of the KZG commitment scheme, com, given $\mathsf{p}(X)$, cannot be opened to the pair $(\widetilde{x},\widetilde{y})$ ensuring that the ciphertext encrypted using $(\widetilde{x},\widetilde{y})$ remains hidden.

First, an observant reader may note that in the use cases envisioned, the security argued above is insufficient since the adversary can "maul" ciphertexts to break any privacy guarantees upon obtaining the batch decryption key. For instance, an adversary can simply flip bits in the $c_{i,2}$ component of ct, and perhaps more devastatingly, since the batch decryption key solely depends on the pair (x,y), the adversary can execute a "copy attack" by copying the pair (x,y) from an honest ciphertext to generate a new ciphertext such that given the batch decryption key, the adversary can decrypt the honest ciphertext even if was not included in the batch of ciphertexts to be decrypted. As we will describe shortly in Section 3, prior works often fail to address these pitfalls.

Second, note that while the binding of the KZG scheme guarantees that ciphertexts with pairs $(\widetilde{x},\widetilde{y})$ that *won't* be included in the B ciphertexts cannot be decrypted, our use case requires a stronger property - namely that even given a ciphertext with a pair (x,y) that *will* be included in the B ciphertexts, an adversarial user should *not* be able to decrypt the ciphertext (i.e. compute opening proof) until the server broadcasts the batch decryption key. This latter requirement requires the use of a variant of KZG that is randomized, and a subsequent strengthening of the 'KZG assumption' to argue security.

In the remainder of this section, we describe how we handle the first type of malleability issues, and defer details regarding the "randomized KZG" to the technical sections.

Adding Non-Malleability Adding non-malleability to ciphertexts corresponds closely to securing encryption scheme against chosen ciphertext attacks, and in this work we shall follow the same approach. To prevent an adversarial mauling of the $c_{i,2}$ component, we use the folkore approach used in the context of adding non-malleability to El-Gamal encryption (see [56]) by additionally attaching a non-interactive proof of knowledge to our ciphertext ct proving knowledge of the encryption randomness ρ in the exponent. By assuming that the proof of knowledge extractor is an *online extractor* (see remark below), this approach prevents the malleability of the $(c_{i,1}, c_{i,2})$ component of ct. This unfortunately doesn't protect the aforementioned "copy attack" since the adversary can replace $(c_{i,1}, ct_{i,2})$ with a fresh ciphertext while keeping (x,y) the same. We handle this in a similar fashion by changing the way y is generated, and having the prover explain the

[2]For simplicity, we assume that the B x_i are all distinct.

generation of y. Specifically, the user still samples x at random, but for y the user first samples a random exponent s, computes $y = H(S)$ where $S = g^s$ and then provide a non-interactive proof of knowledge of s such that $S = g^s$. Thus our final ciphertext is of the form

$$\mathsf{ct} = (x, S, y,\ (g^{\tau}/g^{x})^{\rho},\ H(e(\mathsf{com}/g^{y}, g)^{\rho}) \oplus M, \Pi_1, \Pi_2),$$

where Π_1 and Π_2 correspond to the non-interactive proofs of knowledge discussed above. Intuitively, for any ciphertext such that Π_1 and Π_2 verify, the adversary is able to explain the y that is used, and the message that is encrypted. It turns out that this intuition can be formalized, and is sufficient to provide the security guarantees of our primitive. Note that the above description was intended to provide only the key ideas in our construction, and we refer the reader to Section 7 for the full details.

Remark 1. *We note that for our implemented solution to be efficient, we use the non-interactive version of Schnorr protocol [54] to prove knowledge of discrete log. Unfortunately, it was observed in [56] that proving security of the folklore El-Gamal construction, even in the random oracle model, is problematic using the Schnorr protocol. At a very high level, this stems from the fact that the Schnorr protocol necessitates rewinding for extraction, which is incompatible with the decryption queries of an adversary mounting a chosen ciphertext attack (see [56] for details). This is in fact more broadly an issue with* any *proof of knowledge protocol that requires rewinding, and thus in our construction we assume the existence of a 'straight-line extractor' to circumvent this issue. We justify our instantiation of the Schnorr protocol by noting that in the Random Oracle Model in conjunction with the Algebraic Group Mode (AGM), Schnorr does indeed have straight line extraction and can be used to prove that the folklore 'Schnorr signed El Gamal' is chosen ciphertext secure [28]. The ideas in that work directly extend to our setting as well.*

3 Pitfalls in prior work

In this section we describe a flaw in the Shutter Network [57] that allows an adversary to compromise transactions privacy by exploiting the malleability of the encryption scheme. We believe that this not only necessitates the need for a formal security model, but also informing the various choices in our construction.

The Shutter Network [57] uses it's own construction of threshold encryption for mempool privacy. Indeed it is more efficient than other proposals in the space (such as Ferveo [5] or MEVade [49]), and has been integrated to work on top of Ethereum without requiring changes to the base protocol. However, we found a few flaws in its design and implementation that render it insecure in its current form.

We begin by describing shutter's encryption scheme. We note that at the time of writing this paper, we were not able to find a whitepaper containing a specification of their scheme and had to infer details of their protocol from their GitHub repository. [3]

Shutter works in epochs, each corresponding to a set of encrypted transactions. At the beginning of the protocol, an eon-secret key $sk_e \in \mathbb{F}$ is secret shared amongst the committee and they publish an eon-public key $pk_e = h^{sk_e}$. For each epoch, a unique epoch id $\mathsf{eid} \in \mathbb{G}_1$ is broadcast to users. Then, users encrypt transactions as follows:

- $C_1 = h^r$ for $r \leftarrow \mathbb{F}$
- $C_2 \leftarrow H(e(\mathsf{eid}, \mathsf{pk}_e)^r) \oplus M$. Here, M could be replaced with an encapsulation key which is in turn used to encrypt a long message.

The user then sends (C_1, C_2) as the encryption of their transaction. In order to decrypt, the committee members each release shares of a private epoch key $\mathsf{sk} = \mathsf{eid}^{\mathsf{sk}_e}$. Then, anyone can decrypt as $M = H(e(\mathsf{sk}, C_1)) \oplus C_2$. We now describe some flaws with this design.

- First observe that ciphertexts can easily be mauled by flipping some bits, and they remain valid ciphertexts. Any party who intercepts a user's transaction can modify the ciphertexts, have it be included on chain and learn the transaction without the transaction being included on chain. Thus, this already violates the IND-CCA2 requirements.
- Next, Shutter's efficiency gains compared to Ferveo and MEVade come from sharing a decryption key for all transactions submitted during the same epoch: every transaction encrypted to that epoch can be decrypted once the key is released, regardless of whether the transaction was actually included on chain. This means transactions that are stuck in the mempool and don't make it on chain before the end of the epoch will be revealed even though they won't be executed. For the optimistic roll-up version of Shutter [58], a malicious collator (the entity in charge of selecting a batch of transactions) could withhold transactions from the batch, and still be able to decrypt them once the key is published.

3.1 Disclosure

Upon disclosing our findings to the Shutter team and the Cryptographers who designed Shutter's protocol – Stefan Dziembowski (University of Warsaw and IDEAS NCBR) and Sebastian Faust (TU Darmstadt) – we received the following responses:

[3] https://github.com/shutter-network/shutter/blob/42b2338ed4970bc0df95fa6bdf6b752b57d8ce2f/shlib/shcrypto/encryption.go#L50

From the Shutter team:

> We thank the authors for finding and disclosing the malleability vulnerability in our implementation, which we have fixed thereupon. We are aware of the lack of pending transaction privacy in our protocol, but consider it prohibitively difficult to exploit in the practical scenarios we are focusing on. This is because the fact that transactions are encrypted largely prevents targeted attacks, because modern blockchains and rollups usually include transactions very quickly,[4] and because leaked transactions are prohibited from being included in future blocks. In one variant of our protocol intended for a different setting,[5] we achieve pending transaction privacy, but at the cost of a communication overhead linear in the number of transactions. Nevertheless, we agree with the authors that achieving pending transaction privacy in an efficient manner is very valuable.

From the Cryptographers:

> Thanks for looking at Shutter's implementation and informing us about its issues! We address them below. About the "stuck in the mempool" problem - you are right - Shutter assumes that the transactions make it to the blockchain before the decryption key is published (how big of a problem it is depends on the concrete application).
>
> About malleability: the formal cryptographic description of the Shutter protocol we prepared in 2021 did not suffer from this problem. In fact, we were well aware of the non-malleability requirement, so we based Shutter on non-malleable ID-based threshold encryption. The bug you found was introduced during Shutter programming. We take it as a lesson on the importance of interaction between the cryptographers and software developers during the entire system development process. The development team has fixed the issue in the meantime. We will also publish the Shutter description (as it was designed in 2021) in a paper that we will put on the Cryptology Eprint archive soon. Thanks again!

4 Related Work

Trusted-Execution. One could explore Trusted-Execution based solutions where the enclave contains the secret key of a public-key encryption scheme and clients can encrypt their transactions to this public key. The TEE then decrypts and publishes a block of transactions, which gets included on-chain. Given that over $1 billion USD of MEV has been extracted,[6] and that TEEs have been repeatedly shown to be exploitable,[7] it is not reasonable to expect that this approach is secure in the long-term.

Cryptographic. Here, clients use cryptography to *hide* the content of their transaction until it is included in a block i.e. execution is confirmed and ordering has been fixed. The three main approaches considered are timelock encryption [10], threshold encryption [5, 39, 45, 48, 57] and witness encryption [29].

In timelock encryption, transactions lose privacy if they are not included within a given window of time and also need *wasteful* computation to be carried out leading to additional delays.

Threshold encryption with adaptive chosen ciphertext security [12, 13], provides the required privacy guarantees but the communication is proportional to the number of transactions times the committee size for decryption, which is a bottleneck for scaling these systems.

Finally, witness encryption is typically considered to be impractical but a recent work by Döttling et al. [23] showed that you can build signature based witness encryption even for aggregate and threshold BLS signatures where a signature on some public message is the decryption key. Now users can encrypt transactions to a predictable block number and committee members can sign this block number. Thus it is guaranteed that a key to decrypt the ciphertext will appear (in the form of a signature on the block number) at some point in the future. Although this allows for encryption *to the future*, it is not clear how to ensure that transactions which are not included on-chain will not be decrypted.

In contrast, our work achieves the best of threshold encryption and signature based witness encryption as the communication to decrypt an entire block of transactions is independent of the block size and privacy of transactions that have not been included is still preserved.

Order-fairness. An alternate approach to reducing MEV involves attaining consensus amongst miners on the order of incoming transactions to the mempool. This method aims to limit transaction reordering, thus limiting a miner's potential for extracting MEV. Themis [40] and Aequitas [41] illustrate the principle of decentralized order fairness. These methods, however, can only guarantee a weak form of order-fairness, dubbed *batch order-fairness*, which does not entirely eliminate MEV. A protocol suite for maintaining order fairness based on linearizability has been proposed in Wendy [42], although it only ensures coarse order-fairness and does not wholly rule out all possible reorderings.

[4] https://arxiv.org/pdf/2201.05574.pdf

[5] https://github.com/gnosischain/specs/blob/6e454e5ebb0655495e2584c355f81609cc2d7c11/shutter/low-level.md

[6] See https://explore.flashbots.net for flasbhots statistics.

[7] https://sgx.fail

MEV-resistant DEX. Several strategies for redefining DEXs transaction processing model have been proposed. McMenamin et al. [46] and Heimbach et al. [35] offer game-theoretic defenses against MEV extraction through frequent batch auctions and setting slippage tolerance, respectively. Off-chain communication [16] and multi-party computation [4] have been suggested as means to eliminate arbitrage. A unified AMM to mitigate attacks is proposed by A2MM [63]. Despite improved guarantees, these models are more complex and potentially more vulnerable to smart contract bugs. Furthermore, they do not restrict MEV across different DEXs.

5 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. 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 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. An SE-NIZK satisfies satisfies the following properties:

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

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

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

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

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

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

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

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

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.

6 Model and Definitions

To define the security and efficiency requirements of our batched-threshold encryption scheme we extend the IND-CCA2 threshold encryption definition from [13]. First we specify the syntax for batched-threshold encryption.

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

- $\mathsf{ESetup}(1^\lambda, n, t, B) \rightarrow \{\mathsf{epk}, \mathsf{com}, (\mathsf{td}_1, \ldots, \mathsf{td}_n)\}$: An optional, per epoch setup, which has the same inputs as Setup and outputs strings $\mathsf{epk}, \mathsf{com} \in \{0,1\}^*$ which is published and trapdoor information $(\mathsf{td}_1, \ldots, \mathsf{td}_n)$ for each member of the committee.

- $\mathsf{Enc}(\mathsf{pk}, \mathsf{epk}, \mathsf{com}, m) \rightarrow \mathsf{ct}$: Takes as input a message m, epoch setup $\mathsf{epk}, \mathsf{com}$, and a public key pk and encrypts m to output a ciphertext ct.

- $\mathsf{BatchDec}((\mathsf{ct}_1, \ldots, \mathsf{ct}_B), \mathsf{sk}_i, \mathsf{td}_i) \rightarrow d_i$: Takes as input B ciphertexts $(\mathsf{ct}_1, \ldots, \mathsf{ct}_B)$, and a secret key sk_i and outputs a partial decryption d_i or outputs $\perp$.

- $\mathsf{Combine}(\mathsf{pk}, \mathsf{epk}, \mathsf{com}(\mathsf{ct}_1, \ldots, \mathsf{ct}_B), \{d_i\}_{i \in S}) \rightarrow (m_1, \ldots, m_B)$: Takes as input the public key pk, epoch setup $\mathsf{epk}, \mathsf{com}$ and $t+1$ partial decryptions where $S \subseteq [n]$ and $|S| = t+1$, and output messages $(m_1, \ldots, m_B)$ or $\perp$.

Efficiency. We impose various efficiency constraints on the above algorithms. Setup and ESetup must run in time $\mathsf{poly}(n, t, B, \lambda)$. Enc must run in $O(1)$ time. $\mathsf{BatchDec}$ 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 number of parties. Finally, $\mathsf{Combine}$ must run in time $\mathsf{poly}(B, n, \lambda)$.

The ideal functionality. We now describe an ideal functionality (Fig. 1) similar to [13] which models an idealized encryption service. Note that to closely model applications we have in mind, the interaction with the ideal functionality proceeds in epochs. Further, note that the batch of ciphertexts to be decrypted can be determined in an arbitrary manner.[8]

Note that the ideal functionality already incorporates the progression of the protocol through epochs by specifying epoch ids (eid). Any protocol that securely emulates this ideal functionality, does not leak information about the ciphertexts not decrypted in the epoch. The definition further ensures that there is no leakage of any long term secret information in the protocol across epochs. Indeed, the protocol we describe will be shown to securely emulate the ideal functionality.

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

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 setup was successful:

EpochSetup. Maintain two lists of epoch ids $\mathcal{L}_r$ and $\mathcal{L}_c$. If any $t+1$ parties send $(\mathsf{ESetup}, \mathsf{eid})$, and $\mathsf{eid} \notin \mathcal{L}_r \cup \mathcal{L}_c$, then add eid to $\mathcal{L}_r$.

Encryption. When a client U sends $(\mathsf{Enc}, m, \mathsf{eid})$, sample a receipt $c \leftarrow \Gamma$ and store (c, m, eid). Announce c 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 Threshold Public-Key Encryption scheme.

7 Construction

We build a concretely efficient scheme for batch-threshold encryption where the communication to decrypt an entire block of B ciphertexts is independent of the number of transactions and only requires each member of the committee to send one field element. Both our ciphertexts and public key have constant size and do not grow with the batch size nor the number of parties in the committee. We present our construction assuming a trusted dealer that runs the Setup and

[8] In secure computation, this is often explicitly modeled by the environment. Since we do not discuss composability in this work, we refrain from formally defining the environment for improved readability, and implicitly use the fact that the environment determines the batch of ciphertexts to be decrypted.

ESetup algorithms, and distributes necessary information to the committee. In Section 8 we briefly describe how one can use (specialized) multiparty computation (MPC) to emulate the trusted dealer using the committee.

Before we describe of our construction we first recall the KZG polynomial commitment scheme [38]. Specifically, we use the randomized version of the KZG scheme, with the randomizing polynomial degree set to be 1. Intuitively, we do not require a higher degree randomizing polynomial because we use the polynomial commitment scheme in an 'all or nothing' manner - we only require the underlying polynomial in the commitment be hiding until a *single* opening proof is provided, which in our case will correspond to revealing the entire polynomial.

KZG Commitment Scheme [38]. The setup consists of a $\mathsf{crs} = (g, g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^B}, \tilde{g}, h, h^{\tau}, \tilde{h})$ for randomly sampled generators $g \neq \tilde{g} \in \mathbb{G}_1$, $h \neq \tilde{h} \in \mathbb{G}_2$ and $\mathsf{dlog}_g \tilde{g} = \mathsf{dlog}_h \tilde{h}$. The commitment to a polynomial of degree $\leq B$ is computed as $\mathsf{com} = \tilde{g}^r g^{p(\tau)}$, where $r \leftarrow\$ \mathbb{F}$. To open the polynomial at a point x^* to the value y^*, the prover publishes $\pi = (\pi^1 = g^{q(\tau)}, \pi^2 = g^r)$, where $q(x) = (p(x) - y^*)/(x - x^*)$. The proof is verified by checking that

$$e(\mathsf{com}/g^{y^*}, h) = e(\pi^1, h^{\tau}/h^{x^*}) \cdot e(\pi^2, \tilde{h}). \tag{1}$$

For convenience of notation we will separate the crs into two separate $\mathsf{crs}_1 = (g, g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^B}, h, h^{\tau})$ and $\mathsf{crs}_2 = (\tilde{g}, \tilde{h})$. We note that from the above description, to compute the opening to the polynomial it suffices to be in possession of the polynomial p, and g^r, i.e. it is not required to know r to compute π. We will utilize this observation in our construction.

Setup. Now to build the desired batched-threshold encryption scheme we use the KZG commitment scheme as follows. The committee runs a setup procedure for a batch size B, and publishes $\mathsf{crs}_1 = (g, g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^B}, h, h^{\tau})$. The committee also holds Shamir secret shares of the *trapdoor* – Lagrange coefficients $(L_0(\gamma), L_1(\gamma), \ldots, L_B(\gamma))$ on the domain $\Omega_L = \{x_0, x_1, \ldots, x_B\}$, for some $\gamma \notin \Omega_L$. Note that the batch size B is an upper limit and dummy ciphertexts can always be inserted to appropriately pad any batch of size $< B$.

For our concrete instantiation, whenever an FFT is to be carried out, we use domains that form a smooth multiplicative subgroup such as the roots of unity ($\Omega = \{1, \omega, \omega^2, \ldots, \omega^{B-1}\}$) in a prime field. However, for the Lagrange polynomials we use $\Omega_L = \{1, \omega, \omega^2, \ldots, \omega^{B-1}, \tau\}$. Looking ahead, we have to interpolate a polynomial over the points $\Omega \cup \{\gamma\}$ in order to produce KZG opening proofs over all points in Ω [26], but this can still be done in $O(B \log B)$ time as the domain is *almost* smooth. The idea is to first interpolate a quotient polynomial $q(x)$ over the smooth subset Ω. Then we observe that the polynomial we are interested can be computed using $O(B)$ multiplications as $f(x) = q(x)(x - \gamma) + f(\gamma)$.

EpochSetup. For each epoch, the committee samples a fresh $\mathsf{crs}_2 = (\tilde{g}, \tilde{h})$ such that $\mathsf{dlog}_g \tilde{g} = \mathsf{dlog}_h \tilde{h}$ and publishes a random group element $\mathsf{com} = g^{\alpha} \tilde{g}^r$ for which they hold shares of $\alpha L_B(\gamma)$, and r.

Remark 2. *We note that to compute $\alpha L_B(\gamma)$, EpochSetup requires the "secret state" $L_B(\gamma)$ (dependent on τ sampled in Setup). When emulating the trusted dealer via an MPC, at the end of Setup, the committee members will have also possess shares of $L_B(\gamma)$ which will allow for the subsequent computation of the shares of $\alpha L_B(\gamma)$ for each instance of EpochSetup via an MPC. This ensures* distributed *transfer of the secret state $L_B(\gamma)$ between the two setups without affecting efficiency. In our trusted dealer description, we assume this state is maintained between the two setups.*

Encrypt. Now to encrypt a message we rely on the fact that the opening proof π is hard to compute. Taking a closer look at the KZG verification equation, we observe that if the encryptor exponentiates the left hand side of Eq. (1) by ρ for any $x^* \in \Omega, y^* \in \mathbb{F}$, then this is a uniformly random element that can be used to mask the message. However, the committee must still be able to decrypt the ciphertext and we do this by providing *hints* – $(h^{\tau}/h^{x^*})^{\rho}$ and $\tilde{h}^{\rho}$ as part of the ciphertext.

Decrypt. To decrypt, it is sufficient to produce a proof π of opening the commitment to y^* at x^*. Here we note that the shares of the powers of τ, and α actually form a trapdoor that allows the committee to *equivocate* the commitment, and hence open it to any point (x^*, y^*) of their choice. As described in *Fig.* 2, the committee will choose a batch of transactions and decrypts them by equivocating the commitment according to the ciphertexts.

In more detail, a threshold number of parties in the committee agree on a batch of ciphertexts $\{\mathsf{ct}_1, \ldots, \mathsf{ct}_B\}$ to be decrypted for some particular epoch. The committee then finds the unique degree-B polynomial $p(x)$ satisfying the following constraints, where tg_i corresponds to the y value sampled during Encrypt:

- $\{p(\omega^i) = \mathsf{tg}_i\}_{i=0}^{B-1}$
- $p(\tau) = \mathsf{dlog}(\mathsf{com})$

and output $p(\gamma)$, where com is that epoch's setup and $\gamma \notin \Omega_L$. Given $p(\gamma)$, the polynomial $p(x)$ can be recovered, allowing anyone to generate the first term of opening proof (π^1). We note that this can be computed non-interactively as $p(\gamma) = \sum_{i=0}^{B} p(x_i) L_i(\gamma)$ and the parties have shares of $L_B(\gamma) p(\tau)$ and $\{L_i(\gamma)\}_{i=0}^{B-1}$ along with $\{p(x_i)\}_{i=0}^{B-1}$ in the clear. Since shamir secret sharing is linearly homomorphic, the committee can compute shares of $p(\gamma)$ locally as

$$[p(\gamma)] = \sum_{i=0}^{B-1} p(x_i)[L_i(\gamma)] + [L_B(\gamma) p(\tau)].$$

In addition, each member of the committee also sends shares of $\tilde{g}^r$ computed as $\tilde{g}^{[r]}$, which allows π^2 to be recovered. Thus, reconstruction only requires one field element and one group element to be sent by each party. For simplicity of exposition, we use a trusted dealer in Fig. 2, to handle the Setup and EpochSetup phases. In Section 8 we discuss how the dealer can be emulated using secure multi-party computation.

CCA2 security. It is easy to see that simply masking the message with randomness leaves it susceptible to man-in-the-middle attacks. We handle this by making y^* the output of a one way function and having the encryptor attach a simulation-extractable NIZK proving knowledge of the pre-image of the one way function and the randomness ρ used, and the ciphertext is a tag in the NIZK. The one way function we use is based on the discrete log problem and the NIZK is a very efficient sigma protocol (proof of knowledge of discrete logarithm).

8 Emulating the Dealer

In this section we discuss how the committee can use secure multi-party computation to emulate the trusted dealer.

8.1 Setup

The Setup requires the committee to secret share the Lagrange coefficients $-L_0(\gamma), L_1(\gamma), \ldots, L_{B-1}(\gamma)$ defined on the domain $\Omega_L = \{1, \omega, \ldots, \omega^{B-1}, \tau\}$, for some $\gamma \notin \Omega_L$. In addition they also publish $\mathsf{crs}_1 = (g, g^\tau, g^{\tau^2}, \ldots, g^{\tau^B}, h, h^\tau)$. The MPC first samples a secret sharing of a random value τ and computes it's powers $\tau^2, \ldots, \tau^{B-1}$ using standard techniques [18]. In fact, all terms can be computed in constant rounds using [3]. Given shares of the powers of τ, the parties can locally exponentiate their share and announce $\{g^{[\tau^i]}\}_{i\in[B-1]}$. The crs can now be computed by carrying out Lagrange interpolation in the exponent.

We now describe an MPC protocol to distribute shares of the i-th Lagrange coefficient $L_i(\gamma)$. Recall that for a domain $\Omega = \{x_0, x_1, \ldots, x_B\}$, $L_i(\gamma) = \Pi_{j\neq i} \frac{\gamma - x_j}{x_i - x_j}$. We separate this into two cases:

- $(x_i = \omega^i)$ Here, a *secret* value (τ) only shows up once in the numerator and once in the denominator. Thus, this only requires computing one inversion and one multiplication of a secret shared value. Again, these can be done using standard techniques.
- $(x_i = \tau)$ Although this is not needed in the setup phase, it will be needed later in the epoch setup phase. This case is slightly trickier as (τ) appears multiple times in the denominator. However, upon expanding the denominator in symbolic manner, the denominator is just a linear combination of the powers of τ. Since the parties already have a secret sharing of the powers of τ, they can locally

Batched-Threshold Encryption

Parameters: A pairing friendly group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$.

- $\mathsf{Setup}(1^\lambda, n, t, B)$: A trusted dealer runs the setup for NIZK and the commitment scheme and sets $\mathsf{pk} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{ck}, \mathsf{crs}_1 = (g, g^\tau, g^{\tau^2}, \ldots, g^{\tau^B}, h, h^\tau))$, and distributes shares of $(L_0(\gamma), L_1(\gamma), \ldots, L_{B-1}(\gamma))$ to the committee, setting for each $i \in [n]$, $\mathsf{sk}_i = \{[L_j(\gamma)]_i\}_{j=0}^{B-1}$. Output $(\mathsf{pk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n))$.
- $\mathsf{ESetup}(1^\lambda, n, t, B)$: The trusted dealer samples $\alpha, \delta, r \leftarrow\$ \mathbb{F}$ and sets $\mathsf{crs}_2 = (\tilde{g} = g^\delta, \tilde{h} = h^\delta)$ and $\mathsf{com} = g^\alpha \tilde{g}^r$. They also secret share $\alpha L_B(\gamma)$ (see Remark 2) and g^r with the committee members. Output $(\mathsf{epk}, \mathsf{td}_1, \ldots, \mathsf{td}_n)$ where $\mathsf{epk} = \mathsf{crs}_2$ and for each $i \in [n]$, $\mathsf{td}_i = ([\alpha L_B(\gamma)]_i, [r]_i)$. Additionally commitments $\{c_i = \mathsf{Com}([r]_i; \eta_i)\}_{i\in[n]}$ to $\{[r]_i\}_{i\in[n]}$ are published and the opening η_i to c_i is given to the i-th committee member.
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{epk}, \mathsf{com}, m)$: To encrypt a message M for a particular epoch with com as the epoch setup, sample $s \leftarrow\$ \mathbb{F}$ and compute $S \leftarrow g^s$ and $\mathsf{tg} \leftarrow H(S)$ where $\mathsf{tg} \in \mathbb{F}$. Finally, sample a random value $\hat{x} \in \Omega$ and output the following as ct:
  - $\mathsf{ct}^{(1)} = H(e(\mathsf{com}/g^{\mathsf{tg}}, h)^\rho) \oplus M$
  - $\mathsf{ct}^{(2)} = (h^\tau / h^{\hat{x}})^\rho$
  - $\mathsf{ct}^{(3)} = \tilde{h}^\rho$
  - S
  - $\hat{x}$
  - $\phi = \mathsf{NIZK}\{s, \rho \mid g^s = S \wedge \mathsf{ct}^{(2)} = h^{\rho(\tau - \hat{x})} \wedge \mathsf{ct}^{(3)} \wedge \mathsf{ct}^{(1)} \wedge \hat{x}\}$
- $\mathsf{BatchDec}((\mathsf{ct}_0, \ldots, \mathsf{ct}_{B-1}), \mathsf{sk}_i, \mathsf{td}_i)$: A threshold number of parties in the committee agree on a batch of ciphertexts $\{\mathsf{ct}_0, \ldots, \mathsf{ct}_{B-1}\}$ such that every ciphertext has a valid proof ϕ_i, $e(\tilde{g}, \mathsf{ct}_i^{(2)}) = e(g^{\tau - \hat{x}_i}, \mathsf{ct}_i^{(3)})$, and $\{\hat{x}_0, \ldots, \hat{x}_{B-1}\} = \Omega \setminus \{\gamma\}$, where $\mathsf{ct}_i = (\mathsf{ct}_i^{(1)}, \mathsf{ct}_i^{(2)}, S_i, \hat{x}_i, \phi_i)$. The i-th party then outputs
$$[p(\gamma)]_i = \sum_{j=0}^{B-1} p(x_i)[L_j(\gamma)]_i + [L_B(\gamma)p(\tau)]_i,$$
and $g^{[r]_i}$ along with a proof $\phi' = \mathsf{NIZK}\{\eta_i, [r]_i \mid c_i = \mathsf{Com}([r]_i; \eta_i) \wedge g^{[r]_i}\}$. Shares with ϕ' not verifying are discarded.
- $\mathsf{Combine}(\mathsf{pk}, (\mathsf{ct}_1, \ldots, \mathsf{ct}_B), \{d_i\}_{i\in S})$: Recover the polynomial $p(x)$ and g^r (using error-correction for $p(x)$ if necessary) and produce KZG proofs of opening $\{\pi_i = (g^{q_i(\tau)}, g^r)\}_{i=0}^{B-1}$ for all points in Ω, where $q_i(x) = (p(x) - \mathsf{tg}_i)/(x - x_i)$ and $\mathsf{tg}_i = H(S_i)$. Output
$$\{M_i = \mathsf{ct}_i^{(1)} \oplus H(e(\pi_i^1, \mathsf{ct}_i^{(2)}) \cdot e(\pi_i^2, \mathsf{ct}_i^{(3)}))\}_{i=0}^{B-1}.$$

Figure 2: Protocol for batched-threshold encryption

compute shares of the denominator, effectively reducing to the case above.

8.2 Epoch Setup

The epoch setup is much more simple and can be carried out for multiple epochs simultaneously, which also reduces the communication [6]. The parties only need to establish sharings of random values α, δ, r, after which shares can be locally exponentiated with appropriate group elements and announced to recover $\mathsf{com}, \tilde{g}, \tilde{h}$. Finally, they carry out one multiplication to obtain shares of $\alpha L_B(\gamma)$. To ensure that the correct shares of g^r are revealed during partial decryption, we publish commitments to shares of r, using a publicly verifiable secret sharing protocol [30, 34, 37, 55]. During batch decryption, we provide a proof that $[g^r]$ was computed correctly (if cheating is detected).

Although the MPC protocols described above are a straightforward application of prior work, we expect them to be reasonably efficient. We also note that an important direction for future work is to optimize these protocols by exploiting the structure of the function being evaluated.

9 Implementation and Evaluation

To evaluate the concrete performance of our batched-threshold encryption scheme we implemented a performant version of our scheme in Rust. We use the `arkworks` [2] library for implementations of pairing-friendly curves and associated algebra and the `merlin` library to handle the Fiat-Shamir transform. Our code can be found at https://github.com/guruvamsi-policharla/batch-threshold-encryption.

Setup. All of our experiments for performance benchmarks Table 2 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. Next, for the committee churn experiments, we adapted the implementation from [32] to our setting and simulated a 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. For the committee nodes we used an `e2-small` instance with 2 GB 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 implement the Setup and EpochSetup protocols using a trusted dealer for a committee of size 16. However, these can be replaced with efficient secure multi-party computation protocols. We note that the Setup protocol is run only once per committee and the EpochSetup can be carried out by the committee during downtime where they are not issuing partial decryptions.

Evaluation. We now evaluate the performance of our batched-threshold encryption scheme. In particular, we aim to answer three main questions and provide insights about bottlenecks in different parts of the protocol.

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

Encryption. As is evident from the protocol description Fig. 2, a constant number of operations are required to encrypt a message, independent of committee/batch size. Our experiments show that it takes less than 6 ms to encrypt a 32 byte message. The ciphertext consists of 1 $\mathbb{G}_1$ element, 2 $\mathbb{G}_2$ elements, 3 $\mathbb{F}$ elements, a 2 byte description of $\hat{x}$, and a string proportional to the message length (32 bytes), resulting in a total size of 370 bytes with group element compression. This is $\approx 3\times$ larger than the threshold Cramer-Shoup CCA2 secure encryption scheme [12, 13] which consists of 4 group elements in a non-pairing friendly group (128 bytes) such as [9].

Partial Decryption. Here, each party in the committee checks a zero-knowledge proof, and then computes it's partial decryption. We observe that over 99% of the cost comes from verifying the proofs of knowledge of discrete log and pairing checks needed for CCA2 security. This warrants the investigation of zero-knowledge proofs that can be verified faster in batches. Table 2 confirms that the time taken grows linearly in the size of the batch of transactions being decrypted. If our scheme was deployed on Ethereum, which processes roughly 500 transactions per block, it would take approximately 2.8 s to verify and compute a partial decryption. Each partial decryption is 80 bytes (one $\mathbb{F}$ element and one $\mathbb{G}_1$ element).

Batch size	Partial decrypt (ms)	Reconstruct (ms)
8	41.5	41.9
32	173.4	165.0
128	678.11	781.4
512	2818.6	3472.2

Table 2: Benchmarks of time taken to compute partial decryptions and to reconstruct messages as the batch size grows.

Reconstruct. Here, any party (without secrets) can recover the messages given all the partial decryptions. The bottleneck here is the $O(B \log B)$ group operations, and $O(B)$ pairings

needed to produce all KZG opening proofs using [26]. We provide timings in Table 2. Again, if our scheme were to be deployed on Ethereum, it would cost less than 3.5 s to decrypt 512 transactions in single-threaded mode and this can be fully parallelized. We note that we implement an *optimistic* decryption protocol where all the shares received from the committee are valid. If certain shares are invalid, the reconstruction protocol will need to use a more expensive error correction protocol such as [61].

Varying Committee Size. Given that the only component that scales with committee size is the size of the Fast Fourier transform to reconstruct $p(\gamma)$ and g^r in *Fig.* 2, increasing the committee size when batch size is large (512 say), does not have a noticeable impact on the reconstruction time which is dominated by the cost of producing KZG opening proofs.

Incentive Mechanism. Finding a good committee where a majority of parties behave honestly can be quite challenging. The Shutter network, which has deployed threshold encryption, aims to solve this through decentralized governance via a ShutterDao, where the committee members are chosen by stakeholders in the system. To ensure liveness, one can reward nodes and penalize malicious behavior via stake slashing. We note that this is similar to the case where if too many block validators go offline in Ethereum, the blockchain comes to a standstill.

Denial of Service. Unfortunately, transaction privacy comes with the additional challenge of potential denial of service attacks We present a few approaches to handle this below. However, these approaches are application specific and should be addressed more carefully in future work that aims to deploy encrypted mempools.

A general strategy to prevent "spam" is to demand proofs from users that transactions are valid. These proofs can be quite efficient when designed for specific applications. Consider for instance, a DEX such as Uniswap on an L2, and the client wishes to exchange tokens. Here, it may be sufficient to encrypt just the amounts and token names along with a range proof showing that they indeed own enough tokens. Another option is to implement some kind of penalty for submitting invalid transactions, which raises the cost of launching a denial of service attack. Note that sender addresses need not be encrypted, which allows for "blacklisting" of repeat offenders.

Remark 3. *Not all transactions in the mempool need privacy. For instance, a peer-to-peer transfer can be submitted in the clear to the mempool as the recipient would receive the same amount, irrespective of the ordering of the block. Since transactions with mempool privacy are more expensive they can be priced differently, and those with a need for privacy can opt in.*

9.1 End-to-End Performance.

In this section, we compare the end to end performance of our batched-threshold encryption scheme with Ferveo [5], MEVade [49], McFly [23] and Shutter Network [57]. We consider the constant sized ciphertexts scheme from McFly which relies on the committee holding shamir shares of the secret key. The other scheme has ciphertext size linear in the committee size, which is impractical in the blockchain setting.

We provide concrete performance figures in Table 3 for our work and contrast them with Ferveo [5], MEVade [49], McFly [23] and Shutter Network [57]. We also provide benchmarks on how expensive committee changes are in Table 4.

Setup. All schemes apart from ours, only need a random value to be secret shared at the beginning of the protocol. In addition, our protocol needs to carry out multiplication and inversions in MPC. To estimate the cost of this we point to prior work [15] which provides benchmarks for a protocol run over a WAN with 50 parties. The depth of our setup circuit is 3 as the powers of tau can be computed in depth 1 using [3], and the lagrange coefficients need an additional 2 rounds (one for inverse and one for multiplication). The number of gates is $O(B)$ which is < 10000 for a batch size of 512. From Table 5 of [3], we can see that a circuit of depth 20, and 1 million gates can be computed in under 3 minutes for the WAN configuration. This provides a very conservative upper bound on the setup protocol and the cost grows linearly with the number of parties. We note here that these protocols will abort when a party behaves maliciously. There are other guaranteed output delivery MPC protocols, albeit with more rounds but similar communication [33], that can identify misbehaving parties, allowing them to be penalized. One could run the protocols with abort, optimistically, and if parties abort, then they can run the guaranteed output delivery protocol.

Epoch Setup. We assume that the setup for many epochs is carried out at once. Each epoch requires one multiplication and 3 secret shared random values for α, δ and r. However, in this case we additionally demand that the sharing of g^r is a publicly verifiable secret sharing [30, 34, 37, 55] as malicious parties should not be able to cause the protocol to abort after execution. For this particular task, a 50 node network, [37] takes less than 18 seconds on a LAN. We note here that the communication is actually quite small (< 1 MB) and most of the time is spent on computation. Thus, we expect that one can also scale this to a WAN setting without incurring much more overhead. Moreover the epoch setups can be paralellized and carried out well before the actual epoch. It is expected that this number will come down with faster implementations of hidden order groups. This cost can be further amortized when generating many sharings using techniques from [6].

Practical findings. We used average transaction and block sizes, as well as selected a standard curve, to provide real estimates of the cost of our system. In particular, we analyze average transaction size increase, as well as the gas cost of

Parameter	Ferveo/MEVade	McFly	Shutter	This Work
Ciphertext size	$\lvert\mathbb{G}_1\rvert+\lvert\mathbb{G}_2\rvert$	$\lvert\mathbb{G}_T\rvert+\lvert\mathbb{G}_2\rvert$	$\lvert\mathbb{G}_2\rvert+S$	$2\lvert\mathbb{G}_2\rvert+\lvert\mathbb{G}_1\rvert+3\lvert\mathbb{F}\rvert+S+2$
Block decryption broadcast size	$nB\lvert\mathbb{G}_1\rvert$	$n\lvert\mathbb{G}_1\rvert$	$n\lvert\mathbb{G}_1\rvert$	$n(\lvert\mathbb{G}_1\rvert+\lvert\mathbb{F}\rvert)$
Change in transaction size	12%	23%	10%	30%
Ciphertext storage gas cost (USD)	15,120 ($0.7)	30,240 ($1.4)	13,440 ($0.6)	42,000 ($1.9)
|Partial Decryptions|	3MB	6KB	6KB	10KB
|Partial Decryptions|/|Block Size|	500%	1%	1%	2%

Table 3: Ciphertext size and total decryption broadcast size for n committee members, B transactions per block, in terms of S the size in bytes of the encapsulation key, and $\lvert\mathbb{G}_1\rvert,\lvert\mathbb{G}_2\rvert,\lvert\mathbb{G}_T\rvert,\lvert\mathbb{F}\rvert$ the size in bytes of the representation of elements from each group and field. Concrete numbers obtained with the BLS12-381 curve, using the average transaction and block size on Ethereum in 2023. We use the ETH/USD rate on January 1, 2024 for conversions. The last two rows use a committee size of 128 and a block of 512 transactions to give real-world communication cost estimates.

encryption, and the communication cost of decryption. These values are summarized in Table 3. We find:

- Our setup phase is more expensive than prior works, who only need to establish one publicly verifiable secret sharing. But we conservatively expect this overhead to be under 100×. Moreover, our full setup is only run once for a given committee, and our per-epoch setup can be done in advance, and only requires communication between the committee members, where good network connections can be established. Whereas during decryption, the partial decryptions need to be broadcast to the entire network, via gossiping which is typically much slower. Thus, we optimize the latter at the expense of the former. The size of the decryption shares needed to reveal all transactions in a block is only 2% of the size of a block using batched-threshold encryption, versus 500% using Ferveo[9], making propagation to all nodes in the blockchain prohibitively slow.
- During batched decryption, each committee member broadcasts one group element from $\mathbb{G}_1$ and one field element. This is roughly the same as McFly and Shutter's (our message size is roughly double), but we additionally provide transaction privacy. Ferveo provides pending transaction privacy but the amount of information that needs to be broadcast is $B\times$ larger. In a setting with $n=128$ and $B=512$ transactions, we only require a total broadcast of 10KB, versus 3MB for Ferveo. [21] showed that each kilobyte adds ≈ 80 ms to the block propagation time. This implies that Ferveo adds an overhead of ≈ 250 s for block propogation, whereas we add just 0.8 s.
- Our ciphertext is slightly larger than prior works, but still constant size. In practice, we require 400 bytes for our ciphertext, which represents $2 additional spent gas, while Ferveo costs $0.7, McFly $1.4, and Shutter $0.6.

[9] For a block size of 512, using the average size of an Ethereum transaction

Committee churn. When a party joins or leaves the committee, the old committee needs to *securely* "transfer" the secrets it holds to the new committee. This can be done using a dynamic pro-active secret sharing protocol such as [32]. We adapt their code to our setting and provide benchmarks in Table 4. Due to the structure of the protocol in [32], the setup phase, in one-shot, generates correlations required for the refresh of $n/2$ correlations. Even if we only need to refresh one secret, as is the case in prior work, such as Ferveo, we still have to pay the price of generating $n/2$ correlations during the setup phase. However, the refresh phase is the time taken to refresh one secret, but again this can be trivially parallelized across the total number of secrets that need to be refreshed. Extrapolating from Table 4, for a blocksize of 512, with a committee size of 64, tolerating 31 corruptions, and assuming each node can refresh 8 secrets in parallel (needs 8 threads), it would take less than 2 minutes on a WAN and under 4 minutes on a LAN.

Network	Setup (s)	Refresh (s)
LAN	6.24	0.22
WAN	8.21	1.5

Table 4: Time taken to change a committee with 64 parties using the protocol from [32] on a LAN and WAN network. Here Setup refers to the time taken to establish $n/2$ correlations used to "transfer" ownership of shamir secret shared values to a new committee. Refresh refers to the time taken to transfer one secret to the new committee.

In conclusion, we believe that our work represents an important step towards achieving mempool privacy in blockchains. Although the (epoch) setup is more expensive than prior work, we believe it offers a desirable tradeoff when the committee changes are infrequent viz. orders of magnitude less communication in the time-sensitive decryption step in exchange for a more expensive setup.

10 Acknowledgments

We thank the USENIX Security Symposium reviewers for their valuable feedback. Research supported in part by DARPA under Agreement No. HR00112020026, AFOSR Award FA9550-19-1-0200, NSF CNS Award 1936826, and research gifts/awards by Visa Inc, BAIR Commons Meta Fund, Stellar Development Foundation, a J.P. Morgan Faculty Research Award, a Berkely Center for Responsible, Decentralized Intelligence (RDI) Fellowship and a Bakar Fellows Spark Award.

References

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

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

[3] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In Piotr Rudnicki, editor, *8th ACM PODC*, pages 201–209. ACM, August 1989.

[4] Carsten Baum, Bernardo David, and Tore Kasper Frederiksen. P2DEX: privacy-preserving decentralized cryptocurrency exchange. In Kazue Sako and Nils Ole Tippenhauer, editors, *Applied Cryptography and Network Security - 19th International Conference*, 2021.

[5] Joseph Bebel and Dev Ojha. Ferveo: Threshold decryption for mempool privacy in BFT networks. Cryptology ePrint Archive, Report 2022/898, 2022. https://eprint.iacr.org/2022/898.

[6] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 213–230. Springer, Heidelberg, March 2008.

[7] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 26–45. Springer, Heidelberg, August 1998.

[8] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. https://eprint.iacr.org/2018/046.

[9] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, Heidelberg, April 2006.

[10] 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, Heidelberg, August 2018.

[11] 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, Heidelberg, August 2001.

[12] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 410–428. Springer, Heidelberg, August 2013.

[13] 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, Heidelberg, May 1999.

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

[15] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority MPC for malicious adversaries. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 34–64. Springer, Heidelberg, August 2018.

[16] 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.

[17] 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.

[18] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 572–590. Springer, Heidelberg, August 2007.

[19] Isaac David, Liyi Zhou, Kaihua Qin, Dawn Song, Lorenzo Cavallaro, and Arthur Gervais. Do you still need a manual smart contract audit?, 2023.

[20] 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.

[21] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10, 2013.

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

[23] Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wohnig. McFly: Verifiable encryption to the future made practical. Cryptology ePrint Archive, Report 2022/433, 2022. https://eprint.iacr.org/2022/433.

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

[25] 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.

[26] Dankrad Feist and Dmitry Khovratovich. Fast amortized KZG proofs. Cryptology ePrint Archive, Report 2023/033, 2023. https://eprint.iacr.org/2023/033.

[27] 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, Heidelberg, April 1990.

[28] Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind schnorr signatures and signed ElGamal encryption in the algebraic group model. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 63–95. Springer, Heidelberg, May 2020.

[29] 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.

[30] 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, Heidelberg, May / June 2022.

[31] 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.

[32] 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, Heidelberg, March 2022.

[33] Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority MPC. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 618–646. Springer, Heidelberg, August 2020.

[34] Jens Groth. Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Report 2021/339, 2021. https://eprint.iacr.org/2021/339.

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

[36] 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.

[37] 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, Paper 2023/451, 2023. https://eprint.iacr.org/2023/451.

[38] 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, Heidelberg, December 2010.

[39] 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.

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

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

[42] 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.

[43] Tom CW Lin. The new market manipulation. *Emory LJ*, 66:1253, 2016.

[44] Varun Madathil, Sri Aravinda Krishnan Thyagarajan, Dimitrios Vasilopoulos, Lloyd Fournier, Giulio Malavolta, and Pedro Moreno-Sanchez. Cryptographic oracle-based conditional payments. In *NDSS*. The Internet Society, 2023.

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

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

[47] Julien Piet, Jaiden Fairoze, and Nicholas Weaver. Extracting godl [sic] from the salt mines: Ethereum miners extracting value, 2022.

[48] 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. IEEE, 2023.

[49] Julien Piet, Vivek Nair, and Sanjay Subramanian. Mevade: An mev-resistant blockchain design. In *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2023.

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

[51] Dan Robinson. Ethereum is a dark forest. https://www.paradigm.xyz/2020/08/ethereum-is-a-dark-forest, 2020. Accessed: 2022-02-16.

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

[53] samczsun. Escaping the dark forest. https://samczsun.com/escaping-the-dark-forest/, 2020. Accessed: 2022-02-16.

[54] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991.

[55] 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, Heidelberg, August 1999.

[56] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *Journal of Cryptology*, 15(2):75–96, March 2002.

[57] Shutter Network contributors. The shutter network. https://shutter.network, 2021.

[58] Shutter Network contributors. Rolling shutter: Mev protection built into layer 2. https://blog.shutter.network/announcing-rolling-shutter/, 2022.

[59] 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.

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

[61] Lloyd R Welch and Elwyn R Berlekamp. Error correction for algebraic block codes, December 30 1986. US Patent 4,633,470.

[62] 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.

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

[64] 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.

A Security Proof

We prove security with guaranteed output delivery against a fully malicious adversary corrupting up to $t < n/3$ members of the committee and any number of users creating ciphertexts. We note that it is also possible to upgrade the protocol to $t = n/2$ using standard techniques of publishing commitments to secret keys of parties and proving that partial decryptions were computed correctly. For instance, one could use the KZG commitment scheme itself and prove that the computation was carried out using a zkSNARK such as the

Halo2/STARK [8] proof systems. Although this would make the decryption procedure more expensive, one can always *optimistically* decrypt without proofs and if it fails, then demand proofs. By introducing appropriate penalities such as slashing of stake, we can ensure that incorrect partial decryptions are rarely sent – similar to validators on Ethereum attesting to incorrect state transitions.

In order to highlight the core technical ideas without distractions, we focus on the $t = n/3$ case. In the process, we introduce a new assumption that is closely related to the assumption introduced in [38]. We call this the q-strong Bilinear Diffie Hellman Triple Assumption to indicate that there are additional triples given to the adversary beyond that in q-SDH and "Bilinear" to indicate that the Adversary needs to output a value in the target group.

Definition 4 (q-SBDHT Assumption)**.** *Let a, b,c and d be sampled uniformly at random from* $\mathbb{F}$*. Given as input a tuple* $(g, g^a, g^{a^2}, \ldots, g^{a^n}, g^b, g^d, h^d, h^{ac}, h^{cd})$*, for every probabilistic polynomial time algorithm* $\mathcal{A}$*,*

$$\Pr[\mathcal{A}(g, g^a, g^{a^2}, \ldots, g^{a^n}, g^b, g^d, h^a, h^d, h^{ac}, h^{cd}) \to e(g,h)^{bc}] \leq \mathsf{negl}(\lambda).$$

We briefly discuss the need for the new assumption, and discuss why the 'KZG assumption', i.e. the q-strong Diffie Hellman (q-SDH) assumption does not suffice for our setting. Specifically, in the KZG polynomial commitment scheme, the q-SDH is used to argue *evaluation binding*, which states that even for a (potentially adversarially) chosen polynomial commitment, the adversary cannot produce two accepting evaluation proofs for two distinct evaluations at the same evaluation point, i.e. adversary cannot find a commitment c, a tuple x, y, y', π, π' with $y \neq y'$ such that: (i) π attests that y is a valid evaluation of x with respect to c; and (ii) π' attests that y' is a valid evaluation of x with respect to c. Whereas we require that for an honestly generated commitment, if the adversary does not know the polynomial used in the commitment, it *cannot* generate a valid proof for *any* pair (x, y), even if y was such that $p(x) = y$, where p was the polynomial that was committed. We thus view the above assumption as a strengthening of q-SDH), necessary for our construction

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* $\mathcal{A}$ *corrupting* $< n/3$ *parties, given any weakly simulation-extractable NIZK, and provided the q-SBDHT assumption holds in the programmable random oracle model.*

Remark 6. *We note that the* $n/3$ *threshold arises from the fact that adversarial committee members may send incorrect shares during the batch decryption procedure. One can also allow the adversary to corrupt* $< n/2$ *parties and prove that the protocol emulates the ideal functionality with abort. To achieve guaranteed output delivery in this setting we can use standard techniques (at the cost of concrete efficiency) of attaching proofs that all parties carried out their local computation correctly, thereby allowing us to successfully recover secrets from threshold secret sharing, even if* $n/2 - 1$ *corrupt parties behave maliciously and provide incorrect shares.*[10]

We begin by describing the simulator Sim who simulates all interaction between the adversary and the honest parties using only the ideal functionality. Sim first specifies Γ to be the distribution of an encryption of a random message, under a randomly chosen public key. Next, the Sim as a part of Setup: (i) uses the NIZK simulator to sample $\mathsf{crs}_{\mathsf{NIZK}}$; (ii) Elements of crs are sampled honestly; and (iii) sends random field elements as the corrupt party shares. During EpochSetup, Sim: (i) samples com as a random group elements; (ii) sets crs_2 using knowledge of δ; (iii) a random field elements for corrupt party shares of $[\alpha L_B(\gamma)]$; (iv) samples a random value r_i for each corrupt party share; and (v) generates commitments for corrupt party shares as $c_i = \mathsf{Com}(r_i)$, whereas for the shares of honest parties, Sim commits to $c_i = \mathsf{Com}(0)$. Sim remembers all queries made by the adversary to the Random Oracle. Unless specified, if the query has not been previously made, it lazily samples the response and remembers it. If the same query is made again, it uses the same response as before. We will prove security in the programmable Random Oracle model. For convenience we define a set of previously seen honest party ciphertexts $\mathcal{L}$. Every time the simulator sees a ciphertext $\mathsf{ct} = (\mathsf{ct}^{(1)}, \mathsf{ct}^{(2)}, \mathsf{ct}^{(3)}, S, \hat{x}, \phi)$, it updates the set as $\mathcal{L} = \mathcal{L} \cup \{(\mathsf{ct}^{(1)}, \mathsf{ct}^{(2)}, \mathsf{ct}^{(3)}, S, \hat{x})\}$ (dropping the proof). For convenience we write $\mathcal{L} = \mathcal{L} \cup \{\mathsf{ct}\}$ while implicitly dropping the proof. When simulating Encryption, there are two cases:

- For all ciphertexts created by honest parties, Sim receives a receipt ct (say) from the ideal functionality. It aborts if $\mathsf{ct} \in \mathcal{L}$, else Sim forwards the receipt to the adversary, by dropping the attached proof ϕ, and replacing it with a simulated proof ϕ'.
- For any ciphertext $\mathsf{ct} = (\mathsf{ct}^{(1)}, \mathsf{ct}^{(2)}, \mathsf{ct}^{(3)}, S, \hat{x}, \phi)$ created by corrupt parties, Sim first verifies the NIZK proof ϕ and the pairing checks and discards the ciphertext if either fails. Next, it checks if there exists some $\mathsf{ct}' = (\mathsf{ct}'^{(1)}, \mathsf{ct}'^{(2)}, \mathsf{ct}'^{(3)}, S', \hat{x}') \in \mathcal{L}$ such that
  - $\mathsf{ct}'^{(2)} = \mathsf{ct}^{(2)}$, $\mathsf{ct}'^{(3)} = \mathsf{ct}^{(3)}$, and $H(S') = H(S)$
  - $\mathsf{ct}'^{(1)} \neq \mathsf{ct}^{(1)}$

  and aborts if so. Once both the above checks pass, it runs the extractor and receives the randomness ρ used

[10] In order to verify proofs of correct local computation, the statements need to be 'public'. Proving correct computation over private shares is often done by broadcasting private shares to the committee by encrypting to the committee member in question, but as this is not relevant to our contribution, we ignore such details from this work and refer the reader to relevant multiparty computation works for more details.

to prepare the ciphertext. Sim then extracts the message $M = \mathsf{ct}^{(1)} \oplus H(e(\mathsf{com}/g^{\mathsf{tg}}, h)^{\rho})$ and sends this to the ideal functionality.

Finally, to simulate the BatchDecryption protocol, Sim receives the entire batch of messages $(M_1, \ldots, M_B)$ from the ideal functionality. Sim samples a random element as $p(\gamma)$, and then given the tags for the corresponding ciphertexts can computed *expected* corrupt party shares of $p(\gamma)$, and then reverse sample honest party shares such that they reconstruct to $p(\gamma)$. To compute the randomness that explains com with respect to the polynomial p - determined by the tags and $p(\gamma)$ - Sim computes $g^{r'} = (\mathsf{com}/g^{p(\tau)})^{1/\delta}$, where $g^{p(\tau)}$ can be computed from crs_1. Now, given $g^{r'}$ and the sampled shares sent to the corrupt party, Sim again reverse samples honest shares of $g^{r'}$. Note that it cannot compute honest shares of r' in the clear, but as stated before, the proof generation only requires $g^{r'}$. It finally simulates the NIZK ϕ'. This allows the simulator to generate the output of the BatchDecryption protocol.

When the number of corrupt parties is $t < n/3$, the simulator can always recover the polynomial via Reed-Solomon error correction [61] and g^r can be recovered by dropping shares which do not come with valid proofs. Thus for every ciphertext ct_i in the batch of ciphertexts being decrypted, Sim can compute the corresponding KZG proof π_i. When recovering the messages there are two cases:

- For any ciphertext ct_i created by a corrupt party, Sim follows the protocol from Fig. 2 honestly.
- For any ciphertext ct_i that came as the receipt for an honest party's ciphertext from the ideal functionality, Sim computes $e(\pi_i^1, \mathsf{ct}_i^{(2)}) \cdot e(\pi_i^2, \mathsf{ct}_i^{(3)})$ but this time, it aborts if this point has been previously queried to the random oracle. If the point has not been queried, then it then 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 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 and executes the protocol described in Fig. 2.

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

$\mathbf{H}_2$: In this hybrid, whenever a corrupt party creates a ciphertext ct, Sim first verifies the NIZK proof and discards the ciphertext if it fails. Next, it checks if there exists some $\mathsf{ct}' = (\mathsf{ct}'^{(1)}, \mathsf{ct}'^{(2)}, \mathsf{ct}'^{(3)}, S', \hat{x}') \in \mathcal{L}$ such that

- $\mathsf{ct}'^{(2)} = \mathsf{ct}^{(2)}$, $\mathsf{ct}'^{(3)} = \mathsf{ct}^{(3)}$, and $H(S') = H(S)$
- $\mathsf{ct}'^{(1)} \neq \mathsf{ct}^{(1)}$

and aborts if so. Sim then decrypts the ciphertext using the trapdoor τ as $M = \mathsf{ct}^{(1)} \oplus H(e(\mathsf{com}/g^{\mathsf{tg}}, (\mathsf{ct}^{(2)})^{(\tau-\hat{x})^{-1}}))$. We will also be simulating the NIZK proofs using SimProve.

We will now argue that Sim aborts with negligible probability. Note that because the NIZK is weakly simulation-extractable, despite being able to query an oracle for proofs on an arbitrary set of statements, if the adversary produces a valid proof ϕ^* for a statement $\bar{x}^*$, then with overwhelming probability, one of the following must be true:

- $\bar{x}^* \in Q$, where Q is the set of queries made by the adversary
- the PPT extractor Ext outputs the witness w^* corresponding to $\bar{x}^*$

Thus, if we view the ciphertexts from honest parties as the set of queries the adversary has made, then any ciphertext output by the adversary is either identical to a ciphertext from an honest party (ignoring the proof) or there exists an extractor that outputs $\mathsf{dlog}_g(S)$ and $\mathsf{dlog}_{h^{\tau-\hat{x}}}(\mathsf{ct}^{(2)})$. In the reduction that follows, we will need to extract witnesses from (potentially) many ciphertexts, which could lead to an exponential runtime in the number of ciphertexts. By demanding that the extractor is straight-line we avoid this problem.

Now observe that all honest party ciphertexts are created using a randomly sampled instance of the discrete log problem and if Sim aborts, it means all parts of the ciphertext except for $\mathsf{ct}^{(1)}$ and the NIZK proof match some honest party's ciphertext, which implies that we are in the latter case and hence, there exists an extractor which outputs $\mathsf{dlog}_g(S)$ and $\mathsf{dlog}_{h^{\tau-\hat{x}}}(\mathsf{ct}^{(2)})$. We will now use this extractor to solve the discrete log problem.

In the reduction, we insert the discrete-log challenge in one of the honest party ciphertexts $\tilde{\mathsf{ct}}$ and simulate the NIZK proof using SimProve. With non-negligible probability the adversary produces a ciphertext ct such that all parts of the ciphertext are the same as $\tilde{\mathsf{ct}}$ except for $\tilde{\mathsf{ct}}^{(1)}$ as described above. Then, by running the extractor on the adversary we have a solution of the discrete-log challenge.

Hence, Sim aborts with negligible probability and this hybrid is computationally indistinguishable from the previous hybrid.

A similar argument can be made for ϕ' where Sim can check if the revealed share of g^{r_i} does not match the share received by the adversary during epoch setup. The details follow in an identical manner, where one can use the extracted opening to break the binding property of the commitment scheme.

$\mathbf{H_3}$: This hybrid is identical to the previous, except that Sim outputs the commitments to the honest parties to be commitments to 0. During batch decryption, Sim uses the correct honest shares to compute $g^{[g]_i}$, but uses the simulated NIZK proof ϕ' to "explain the incorrect opening".

$\mathbf{H_4}$: This hybrid is identical to the previous hybrid, except now Sim uses a different strategy to decrypt ciphertexts from the adversary. Here, the simulator runs the extractor to recover the randomness ρ used by the adversary to create the ciphertext and recovers the message as $M = \mathsf{ct}^{(1)} \oplus H(e(\mathsf{com}/g^{\mathsf{tg}}, h)^{\rho})$. Note that this strategy fails in the case where the adversary is able to *maul* the ciphertext to produce a related ciphertext i.e. the scenario where the simulator in the previous hybrid aborts because then there does not exist an extractor. But as we argued in the previous hybrid, this happens with negligible probability. Furthermore, the messages recovered using the two strategies are identical with overwhelming probability due to the soundness of the NIZK which guarantees that ciphertext is well-formed and hence $e(\mathsf{com}/g^{\mathsf{tg}}, (\mathsf{ct}^{(2)})^{(\tau-\hat{x})^{-1}}) = e(\mathsf{com}/g^{\mathsf{tg}}, h)^{\rho}$.

$\mathbf{H_5}$: In this hybrid, we begin simulating BatchDecryption as follows. First note that Sim knows the shares $[\alpha L_B(\gamma)]$ and $[r]$ held by the corrupted parties and knows the trapdoor δ as it simulated EpochSetup. Given a batch of messages $(M_1, \ldots, M_B)$, the simulator samples a uniformly random value for $p(\gamma)$ and recovers $p(X)$. It then computes g^r such that that it remains consistent with the $p(\gamma)$ it just sampled as $g^r = (\mathsf{com} \cdot g^{-p(\tau)})^{\delta^{-1}}$. This information is sufficient to fully determine the shares of g^r and $p(\gamma)$ that honest parties should announce in order to remain consistent with the shares held by corrupt parties. Finally, for all honest party ciphertexts it programs the random oracle, aborting if necessary as described above. We now argue that Sim aborts with negligible probability.

Claim 7. *If the q-SBDHT assumption holds, then for all PPT adversary $\mathcal{A}$ and for all x^*, y^*,*

$$\Pr[\mathcal{A}((g, \tilde{g}, h, \tilde{h}, g^{\tau}, \ldots, g^{\tau^n}, g^{\alpha}\tilde{g}^r, h^{\tau}, h^{(\tau - x^*)\rho}, \tilde{h}^{\rho})) \to e(g^{\alpha - y^*}\tilde{g}^r, h)^{\rho})] \leq \mathsf{negl}(\lambda)$$

where τ, α, r and ρ are sampled uniformly at random from $\mathbb{F}$.

Proof. Suppose the above claim was false. Then, we construct an adversary $\mathcal{A}'$ such that

$$\Pr[\mathcal{A}(g, g^a, g^{a^2}, \ldots, g^{a^n}, g^b, g^d, h^a, h^d, h^{ac}, h^{cd}) \to e(g,h)^{bc}] \geq \mathsf{poly}(\lambda),$$

thereby violating the q-SBDHT assumption.

$\mathcal{A}'$ provides $\mathcal{A}$ with an instance such that $\tau = a + x^*$, $g^{\alpha}\tilde{g}^r = g^{b+y^*}$, $\rho = c$ and $\delta = d$. First note that we have all the powers of a in the exponent, and x_i is public so we can compute all powers of τ in the exponent. Correcntess is easy to see for the remaining terms. When this instance is given to $\mathcal{A}$, it makes $\mathsf{poly}(\lambda)$ Random Oracle queries and as a guess for $e(g^{\alpha - y^*}\tilde{g}^r, h)^{\rho})$, $\mathcal{A}'$ outputs a randomly chosen query. Note that since, $e(g^{\alpha - y^*}\tilde{g}^r, h)^{\rho}) = e(g,h)^{bc}$, $\mathcal{A}'$ chooses the correct query with probability at least $1/\mathsf{poly}(\lambda)$, thus violating the q-SBDHT assumption. □

Thus, from Claim 7, the adversary queries the random oracle at the *bad* point $e(g^{\alpha - y^*}\tilde{g}^r, h)^{\rho}$ with at most negligible probability and hence this hybrid is computationally indistinguishable from the previous hybrid.

$\mathbf{H_6}$: Sim now simulates ciphertexts generated by honest parties. It does so by encrypting a random message, under the public key. From Claim 7, $H(e(g^{\alpha - y^*}\tilde{g}^r, h)^{\rho})) \approx_c U$ as the adversary queries the point $e(g^{\alpha - y^*}\tilde{g}^r, h)^{\rho}$, with negligible probability. Hence this hybrid is computationally indistinguishable from the previous hybrid.

$\mathbf{H_7}$: Sim no longer uses any inputs from honest parties and the receipts of ciphertexts from the ideal functionality have a distribution identical to encryptions under a random message. After extracting messages from adversary ciphertexts, it sends them to the ideal functionality and when batch decryption is invoked the simulator uses the batch of messages it received from the ideal functionality to simulate the communication during batch decryption. Thus this final hybrid has an identical distribution to the previous hybrid.