BEAT-MEV: Epochless Approach to Batched Threshold Encryption for MEV Prevention

Jan Bormet
TU Darmstadt

Sebastian Faust
TU Darmstadt

Hussien Othman
TU Darmstadt

Ziyan Qu
TU Darmstadt

Abstract

In decentralized finance (DeFi), the public availability of pending transactions presents significant privacy concerns, enabling market manipulation through miner extractable value (MEV). MEV occurs when block proposers exploit the ability to reorder, omit, or include transactions, causing financial loss to users from frontrunning. Recent research has focused on encrypting pending transactions, hiding transaction data until block finalization. To this end, Choudhuri et al. (USENIX '24) introduce an elegant new primitive called *Batched Threshold Encryption* (BTE) where a batch of encrypted transactions is selected by a committee and only decrypted *after* block finalization. Crucially, BTE achieves low communication complexity during decryption and guarantees that all encrypted transactions outside the batch remain private. An important shortcoming of their construction is, however, that it progresses in epochs and requires a costly setup in MPC for *each* batch decryption. In this work, we introduce a novel BTE scheme addressing the limitations by eliminating the need for an expensive epoch setup while achieving practical encryption and decryption times. Additionally, we explore a previously ignored question of how users can coordinate their transactions, which is crucial for the functionality of the system. Along the way, we present several optimizations and trade-offs between communication and computational complexity that allows us to achieve practical performance on standard hardware (< 2 ms for encryption and < 440 ms for decrypting 512 transactions). Finally, we prove our constructions secure in a model that captures practical attacks on MEV-prevention mechanisms.

1 Introduction

A fundamental challenge of public blockchains is that all transaction data is publicly available. This does not only have serious implications for user privacy, but hinders many real-world applications such as voting or auctions. A particular domain where transaction privacy is becoming crucial is decentralized finance (DeFi). DeFi offers a plethora of financial applications (e.g., exchanges, lending platforms and more), which are realized via a smart contract running on a decentralized blockchain. They promise a fair and reliable trading platform that is available to everyone, which has attracted huge investments from private and institutional investors. At the time of writing, more than \$80 billion has been locked into various DeFi applications[1].

Similar to traditional financial markets, the DeFi ecosystem is prone to market manipulation. Daian et al. [19] were the first to identify that knowledge of transaction data is a security concern for public blockchains and introduced the concept of *miner/maximal extractable value* (MEV). Maximal extractable value refers to the amount of money that a block proposer can extract by re-ordering, omitting, or including transactions. While most blockchains view some form of transaction selection as benign, e.g., transactions that pay higher fees shall be processed at faster speed, a plethora of works on MEV attacks show that this power can be exploited [24, 30, 56]. For example, a malicious block proposer may front-run a profitable transaction, where it places a buy order of a certain asset just before a huge buy order of the same asset is executed. Flashbots, an organization doing research and development on MEV, estimates that until 2022 almost \$700 million of value has been extracted, benefiting mainly miners or professional trading bots.[2]

On a high-level MEV exploits that transactions are publicly available in the network prior to being integrated into a block. More precisely, users that wish to execute a transaction broadcast it to the blockchain P2P network. The nodes that receive these transactions store them in an internal storage called the *mempool*, where they are queuing for processing. Block proposers select transactions from their local mempool, build a block and broadcast the block to the network. Blocks are then validated by other nodes in the network, and appended to their local view on the blockchain. Once a transaction is part of a block, it is eliminated from the nodes' local mempool. Crucially, the block proposers can freely decide on the order of

[1] https://defillama.com
[2] https://explore.flashbots.net

transaction execution, which is precisely what enables MEV.

While many different solutions to prevent malicious MEV have been proposed (cf. Section 1.3), one of the most promising approaches is to keep transactions in the mempool private [4, 17, 36]. This prevents MEV since block proposers now have to select their transactions without knowledge of the transaction content. An appealing solution to build private mempools is to use threshold encryption [4, 17, 58]. A threshold encryption scheme distributes the decryption key among a set of n servers, where any subset of $t \leq n$ servers can jointly decrypt ciphertexts, while $< t$ servers learn nothing about the encrypted plaintext. In a nutshell, threshold encryption can be used as follows to realize private mempools. A decryption key is distributed among a set of servers – often called the *committee* – and the corresponding public key is published, e.g., on the blockchain. To send a shielded transaction to the private mempool, a user encrypts its transaction with the public key of the committee and broadcasts the ciphertext to the blockchain network. Block proposers then build blocks from the encrypted transactions, which only get decrypted by the committee once the block (and hence the order of transactions) is finalized.

The naive way of realizing private mempools via threshold cryptography suffers from an critical shortcoming. The communication to decrypt a batch of B encrypted transactions among a committee of n servers is $O(nB)$. As outlined in [17], this is particularly problematic in large scale P2P networks with many decryption nodes or in a blockchain setting, where the communication must be stored on-chain for verifiability. Moreover, in time-critical applications such as MEV protection additional latency due to a multi-round decryption protocol is inherently prohibitive. Indeed, an existing system for MEV protection called *Shutter* [1] encountered this communication bottleneck and addresses it via releasing so-called *epoch keys*. Here, the committee members locally derive epoch key shares. When t such shares get published all shielded transactions that were sent by users during this epoch can be decrypted. Therefore, the communication complexity for decryption is reduced to $O(n)$ per batch.

Recently, Choudhuri et al. [17] observed a significant shortcoming of the Shutter system which they term *pending transaction privacy*. In Shutter, all transactions of an epoch lose privacy, even if they are stuck in the mempool and were not confirmed as part of a block. For trading applications, this is a major downside as the content of a transaction can reveal a profitable trading strategy that can be front-run during the next epoch. Choudhuri et al. [17] address this issue via a new notion that they call *batched threshold decryption*. The idea is that the committee members *only* decrypt the subset of transactions that made it from the mempool to the blockchain. Technically, this is done by letting the committee members jointly select a batch of encrypted transactions B for which they reveal the corresponding decryption shares. This guarantees that all transactions from B get decrypted with total communication of $O(n)$, while transactions pending in the mempool remain private. The construction of Choudhuri et al. [17] offers an elegant solution for pending transaction privacy. It suffers, however, from the following shortcomings, which we will address in this work:

1. *Expensive epoch setup:* During each epoch the committee executes an expensive epoch setup, which already for modest committee size takes significantly more time than block creation of popular blockchain systems.
2. *Transaction coordination:* Batched threshold encryption requires users to coordinate, which is not addressed in [17]. In addition, shielded transactions that do not end in a batch must be resent, causing bad user experience.

Along the way of addressing these challenges, we present further optimizations and trade-offs that move batched threshold decryption closer to realizing private mempools. We provide background and more details on our contributions below.

1.1 Batched Threshold Encryption

In *Batched Threshold Encryption* (BTE) a committee of n servers share a secret key sk and is given a batch of B ciphertexts. The ciphertexts are encrypted independently under the public key pk. The task of the committee is to decrypt the batch B. As in standard threshold encryption, any set of t (where t is the fixed threshold) out of n servers should be able to decrypt all the ciphertexts in B, and any set of size $< t$ should learn nothing about the encrypted plaintexts. In addition, a BTE scheme should satisfy the following two requirements: First, the communication complexity for each server should be sublinear in the size of B (optimally, a constant). Second, every ciphertext that is not in B should remain private, which is necessary to achieve pending transaction privacy.

Choudhuri et al. [17] present the first BTE scheme that fulfills the aforementioned additional requirements. Their construction relies on a *polynomial commitment scheme*, concretely on the popular KZG scheme [35]. In a polynomial commitment scheme, the sender can commit to a polynomial $p(X)$ via a short commitment com. Later, he can reveal a tuple (x,y) and a short proof π showing that $p(x) = y$. Fundamentally, in [17], the public key is a commitment com to a yet unspecified polynomial. To encrypt a message m_i, a user chooses (x_i, y_i) and encrypts m_i such that decryption can be done with the proof π_i showing that (x_i, y_i) lies on the polynomial with respect to com. In order to decrypt a batch of B transactions, the committee uses a shared trapdoor (which is part of the secret key shares) to specify a degree B polynomial p (with respect to the commitment com) that is consistent with all points (x_i, y_i) that are part of the batch B. Since a B-degree polynomial can be reconstructed with $B+1$ points, the committee members broadcast $(0, p(0))$. Given (x_i, y_i) and $(0, p(0))$, one can compute the proofs π_i of opening (x_i, y_i), which allows to decrypt the ciphertexts.

The scheme enjoys constant communication during batch decryption as the servers need to compute (and reveal) only the single point $(0, p(0))$. Furthermore, the decryption process is efficient as it requires a single pairing operation per ciphertext ($O(B)$ in total). On the other hand, the construction requires an expensive interactive epoch setup for every batch. In a nutshell using techniques from MPC, the servers need to generate a commitment com to a fresh (unspecified) polynomial and share the corresponding trapdoor used for batch decryption. While the authors of [17] point out that the epoch setup can be done during idle time and is independent of the ciphertexts in the batch, it puts a high computational burden on the servers; e.g., for a modest number of 50 servers its execution requires 18 seconds on a LAN (hence, ignoring network latency in global networks). In addition, the servers need to maintain large secret key shares of size $O(B)$. Finally, the construction ignores the problem of how users that wish to encrypt messages coordinate on choosing distinct indices x_i from the setup. If two ciphertexts pick the same x_i from the setup, which has non-negligible probability for polynomial B, the batch can only contain one of the two ciphertexts.

1.2 Our Contributions

In this work we explore a new practical construction of batched threshold encryption as a cryptographic building block for MEV-prevention. In particular, we make the following contributions:

- We construct the first CCA-secure BTE protocol *without per-epoch setup*. While our construction also requires a one-time setup similar to [17], we achieve several additional improvements. Our one-time setup is not tied to the committee's secret key shares and therefore is universal, i.e., it does not need to be executed by the committee and can be re-used. Further, we achieve constant-sized secret key shares, which is a significant improvement over [17] where the size of the secret key shares is linear in the batch size.
- We present formal proofs of security of our construction that cover relevant practical attacks in the context of MEV-prevention.
- We explore several optimizations and trade-offs between communication and computation complexity. In addition, we introduce an additional feature called *verifiability* that protects against practical attacks in the context of MEV.
- We propose a sub-batching technique for removing coordination between encryptors. To our knowledge, this is the first solution to the coordination problem that does not increase the ciphertext size. This is appealing since ciphertexts are stored on-chain in our use case of MEV-prevention. Our techniques also improve on the coordination problem in other settings, e.g., for batch solving of time-lock puzzles [23].
- We provide a comprehensive practical evaluation of our construction via a publicly available implementation of our construction.[3] In particular, we explore several optimizations and compare our results to the benchmarks given by Choudhuri et al. [17]. We establish that encryption can be done in under 2 ms on commodity hardware and show that encryption and partial decryption outperforms the construction of [17] by avoiding pairings in both operations.We show our ciphertext size is constant and the on-chain storage cost is less than 0.4 USD per ciphertext.

We remark that in contrast to [17], our scheme offers a less efficient reconstruction process to decrypt a batch *after* releasing the decryption shares. The construction of [17] requires a single pairing operation per ciphertext ($O(B)$ pairings to decrypt a batch of B ciphertexts). In our construction, we need to perform B pairing operations per ciphertext (and $O(B^2)$ pairings for the entire batch). To mitigate this shortcoming, we evaluate possible trade-offs between the communication and computation complexity. E.g., by increasing the communication (decryption share size) from $O(1)$ to $O(\sqrt{B})$ we reduce the computation to $O(\sqrt{B})$ pairings per ciphertext and $O(B\sqrt{B})$ pairings for decrypting the whole batch. We evaluate an implementation of this optimization and show reconstruction times on standard hardware that are below 1 second for large batches of 512 ciphertexts. For reference, the Ethereum block time is approximately 12 seconds. We note that in practice our property of *verifiability* can help to further mitigate this shortcoming. Instead of letting all members of the committee do the decryption process, we may outsource it to a few powerful servers, and verify the correctness of decryption, which can be done at low cost.

1.3 Related Work

MEV has been a recognized and well-studied problem in DeFi systems [24,30,56]. The term was first introduced by Daian et al. [19]. Not only does MEV cause economic loss to users, but it also poses a threat to the security of the blockchain [19]. Different approaches have been proposed to mitigate the problem of MEV. Flashbots [19], for example, proposed to mitigate the negative effects of MEV by providing an auction mechanism for transactions to be included. The malicious front-running would then be reduced since users actively interact with the miners via private channels. Later, the Flashbots team introduced MEV-Boost [25] for Proof-of-Stake Ethereum, which is a middleware that achieves proposer-builder-seperation. It further decentralized the MEV extraction process and reduced the threat that MEV poses to the blockchain. Unlike encrypted mempools, these approaches do not prevent MEV, but aim to

[3] https://zenodo.org/records/14672008

reduce its negative effects and ultilize it in a more ethical and transparent way.

Several works [4, 36, 42, 46, 58] exploit threshold encryption to address the MEV problem. All of them require heavy communication, because the committee members need to release a decryption share for each transaction in the block. Shutter [1] and Fairblock [44] use threshold identity-based encryption to encrypt transactions to an epoch. As discussed, these works greatly improve communication complexity but do not achieve pending transaction privacy. Similarly, Döttling et al. [22] introduce a scheme for encryption to the future with constant communication complexity, which relies on a signature-based witness encryption scheme. Since their witness allows to decrypt all ciphertexts of an epoch, pending transaction privacy is not guaranteed.

Time-lock encryption [7] guarantees that messages remain hidden for a pre-defined time and is considered for mempool privacy [53]. In contrast to threshold cryptography, time-lock encryption has the advantage of not requiring a quorum of honest users. On the downside, however, it requires parties to carry out wasteful computation, which further delays execution. Prior work [18] requires investing computation for each encrypted message, resulting in huge computational overheads. Recently, Dujmovic et al. [23] proposed time-lock puzzles with efficient batch solving. Their scheme uses a linearly homomorphic time-lock puzzle and a puncturable pseudorandom function. We use their idea of puncturable pseudorandom functions as a building block in our scheme and adjust it to work in a threshold setting.

An alternative to MEV prevention mainly followed by industry are trusted execution environments (TEEs) [43]. TEEs provide secure and private hardware isolation, where data and code is executed in a protected environment. TEEs have been used as a solution to MEV [6, 54], but rely on a strong trust assumption (see, e.g., attacks in [47, 57]).

Another MEV countermeasure is fair ordering. By achieving immediate and consistent ordering of transactions globally, MEV could be mitigated since a miner can no longer easily manipulate the order of transactions. Order fairness is proven impossible [40], and several previous works [37, 38] discussed variants of weaker order fairness, i.e., block order fairness. Wendy [40] further discussed this problem and proposed several protocols for different levels of fairness. While their work provides a certain degree of fairness, these solutions do not solve MEV entirely, and are hard to be deployed since they require changes to existing consensus protocols.

1.4 Challenges of Encrypted Mempool

Despite being a popular and practical solution to the MEV problem, encrypted mempool comes with its own challenges [49, 55]. Care must be taken while designing these schemes to avoid potential pitfalls. We discuss two main challenges of encrypted mempool using threshold encryption, validator collision and metadata leakage, and possible solutions.

In threshold encryption schemes, decryption success relies on an honest majority of validators releasing their shares. Since validators tend to be rational, incentives must be designed to discourage malicious behavior that could lead to decryption failures, including validator collusion. Still, oblivious collusion remains a concern, as it is hard to detect if validators collude to decrypt transactions in advance for arbitrage or backrunning opportunities. To mitigate this risk, one could introduce slashing protocols to dispute malicious validators [49, 58] and penalize them or exclude them from the committee. In particular, tracing algorithms can be used to identify colluders in threshold encryption [10], and one possible solution is integrating tracing functionality to BTE schemes such as ours. We leave this as an interesting direction for future work.

Transaction metadata, such as the sender/receiver addresses and gas fees, are required for including and executing transactions. Since completely encrypted transactions invite spamming and DoS attacks, encrypted mempool schemes often make some metadata public. This brings up the metadata leakage problem, as the public metadata might cause information asymmetries and MEV opportunities. In order to minimize the possible leakage, different approaches can be taken, such as pseudonym sender addresses and anonymous payment services [5]. It is also possible to use SNARKs for signature and gas fee verifications instead of revealing them in plaintext.

Despite the challenges, we view encrypted mempool as a promising direction to mitigate MEV, as most issues can be addressed regardless of the specific (batched) threshold encryption scheme employed. Our work advances more efficient and practical encrypted mempool schemes, and our construction is not particularly compromised by these challenges.

2 Technical Overview

We next present a high level overview of our BTE construction. Initially, we restrict ourselves to the setting of security against chosen plaintext attacks (that is, the adversary learns only the ciphertexts but does not get any decryption oracle). Then, we explain possible attacks against the construction and describe how we mitigate them. Our construction is inspired by recent work on batched time-lock puzzles [13, 15, 23, 41, 52], which allow amortized solving of Time-Lock Puzzles (TLP) [48]. In particular, our work will follow the framework of Dujmovic, Garg, and Malavolta [23].

Following [23, 52], we base our construction on a primitive called Puncturable Pseudorandom Function (PRF) [11, 39]. A puncturable PRF is a regular PRF [31] that is punctured at some point i^*. That is, for a PRF key k, we publish a punctured key k^* such that the value $\mathsf{Eval}(k,i)$ for every $i \neq i^*$ can be computed using the punctured key k^*, while the evaluation at the punctured point, i.e., $\mathsf{Eval}(k,i^*)$, can be computed only using the key k (and should look random even given k^*). In our work, we consider a Key-Homomorphic PRF [14]. In Key-Homomorphic PRFs, for any two keys k_1, k_2, it holds that $\mathsf{Eval}(k_1+k_2, i) = \mathsf{Eval}(k_1, i) + \mathsf{Eval}(k_2, i)$. In our construction, we use the Pairing-based Key-Homomorphic PRF from [23], with two modifications to adjust it to our setting. First, we modify it such that it can be evaluated using g^k instead of k (where g^k remains private). More precisely, for any key $k \in \mathbb{Z}_p^*$, we give an algorithm $\mathsf{ExpEval}$ such that for any input i: $\mathsf{Eval}(k, i) = \mathsf{ExpEval}(g^k, i)$. Second, we work in asymmetric pairing groups to combine it with a suitable threshold encryption scheme, namely some form of Threshold ElGamal Encryption in the Exponent [27] (see Section 4 for details).

To encrypt a message m, the encryptor needs to choose a random k, a punctured index i, and puncture the PRF on i under k, producing the punctured key k^*. Then, we compute $\gamma \leftarrow m + \mathsf{Eval}(k, i)$. Observe that by the security of the PRF, the value $\mathsf{PRF}(k, i)$ looks random to anyone who does not know k (or g^k due to our modification for exponent evaluation), thus no information can be learnt about m from γ. In order to enable the decryption committee to learn m, the encryptor attaches to γ a threshold ElGamal encryption ct of g^k under the public key of the decryption committee. Thus, the ciphertext is the tuple (k^*, γ, ct).

The most interesting part of this construction is the decryption process. To decrypt a batch of ciphertexts $\{c_i\}_{[B]}$, there is no need to decrypt the underlying key g^{k_i} used in each ciphertext, rather the servers can multiply all the ElGamal ciphertexts together and release a decryption share for the resulting ElGamal ciphertext, which is an encryption of $K = g^{\sum k_i}$ due to the multiplicative homomorphism. Given at least t such decryption shares, one can aggregate them to learn $g^{\sum k_i}$ and use it to decrypt every ciphertext in the batch as follows. Let the i-th ciphertext be $c_i = (i, k_i^*, \gamma_i, ct_i)$. Given the other ciphertexts in the batch (in particular their punctured keys k_j^*) as well as $K = g^{\sum^{[B]} k_i}$, we can reconstruct m_i as follows:

$$m_i = \gamma_i + \sum_{j \neq i} \mathsf{PEval}(k_j^*, i) - \mathsf{ExpEval}(K, i)$$

To see why this is correct, observe that by the puncturing property of the PRF scheme, the value $\mathsf{PEval}(k_j^*, i)$ can be computed for any $i \neq j$ and it is equal to $\mathsf{Eval}(k_j, i)$ (i.e., the evaluation with the secret key k_j). Also, from the key-homomorphism and evaluation in the exponent of the underlying PRF scheme, it holds that:

$$\mathsf{ExpEval}(K, i) = \mathsf{Eval}(k_1 + \cdots + k_B, i) = \textstyle\sum_j^{[B]} \mathsf{Eval}(k_j, i)$$

and therefore,

$$\textstyle\sum_{j \neq i}^{[B]} \mathsf{PEval}(k_j^*, i) - \sum_j^{[B]} \mathsf{Eval}(k_j, i) = -\mathsf{Eval}(k_i, i)$$

Adding this to γ_i cancels out the PRF padding and reveals m_i. Observe that there was no interaction between the servers. They simply release their threshold ElGamal decryption share of $ct = \prod_j^{[B]} ct_j$, which is a single group element ($O(1)$ communication complexity). Hence, our construction fulfills the efficiency requirement in BTE. Furthermore, observe that the decryption key does not depend on the ciphertexts that are not included in the batch, which preserves their privacy.

Removing Index-Coordination. In the above description, we assumed that each ciphertext in the batch has a distinct punctured index i, i.e., we assume a coordination mechanism to avoid collisions. This assumption was critical for the decryption. If two ciphertexts have the same punctured index, then the decryption process fails. Unfortunately, this assumption is not easily satisfied in our construction, since, in the underlying PRF construction, the index domain is polynomially bounded and all information about indices are included in the setup. Since we allow an unbounded set of encryptors (any user of a blockchain can send transactions), we need to refrain from assuming coordination for practical concerns. Otherwise, we would need to either (i) increase the domain of indices significantly, (ii) let the parties communicate to reach an agreement on the indices, or (iii) introduce a central authority that assigns the indices. All of these solutions are impractical as they either impair the efficiency of the construction or introduce a single point of failure.

In order to overcome this challenge, in [23, 28], they use a technique related to finding a perfect matching in bipartite graphs [33]. On a high level, for a given statistical correctness parameter λ_s, each user samples $d = O(\log \lambda_s)$ indices randomly from the domain of indices, and sends d ciphertexts of the message, each ciphertext corresponding to one index. For decryption, using the Hall's theorem [33], they show that with overwhelming probability there will be a matching in which each ciphertext has a unique index for appropriate parameters d and domain size. We note that this approach can also be used in our construction. This solution however increases the ciphertext size to $d \cdot |c|$, which is costly in a blockchain setting where ciphertexts are stored on-chain. We propose a new technique to avoid index collisions in Section 5.3. In our approach, we consider a trade-off between the communication required for decryption and the ciphertext size. That is, we can increase the communication complexity, e.g., to $O(\sqrt{B})$ instead of $O(1)$, while keeping the ciphertext size constant. The intuition is that if we split a batch into sub-batches, we need to prevent only large collisions, i.e., we only need to guarantee that for every index no more than $\sqrt{B}$ ciphertexts use it. We formally prove that we can achieve this with overwhelming probability under reasonable parameters, and provide practical analysis of statistical correctness. Interestingly, we observe that our approach has the additional benefit to significantly improve efficiency of decryption from $O(B^2)$ to $O(B\sqrt{B})$ pairings. Thus, we benefit from small ciphertext size and more efficient computation at the cost of a reasonable increase in communication.

Non-malleability and Rouge Ciphertext Attacks. The above description was restricted to the setting of Chosen Plaintext Attacks (CPA). For our application of mempool privacy, we need to consider a stronger setting that is illustrated by the following two attacks violating the privacy of ciphertexts outside the batch. Consider a ciphertext $c = (i, k^*, \gamma, ct)$ that is not included in the batch. The adversary can decrypt it by attempting to send a malformed ciphertext and hope that it gets included in the batch. There are two ways for the adversary to achieve this, which were also discussed in [17]:

- *Mauled ciphertexts:* The adversary can maul γ by, e.g., flipping bits. Then, after the malformed ciphertext gets decrypted, he can restore the original γ.
- *Copy attack:* The adversary can copy (i, k^*, ct) from the targeted ciphertext c and choose an arbitrary γ. For decryption it is enough to compute the sum of the PRF keys in the batch. Hence, since the malformed ciphertext uses the same PRF key as in the targeted ciphertext c, the adversary will be able to decrypt c as well.

To address these attacks, we require that our construction fulfills the stronger notion of Chosen Ciphertext Security (CCA). In addition to the above, an adversary may launch an attack to inject faulty ciphertexts into a batch (e.g., use a k^* inconsistent with k encrypted in ct). This may result into an incorrect decryption process, possibly leaking information about transactions pending in the private mempool. Following [23], we define *Rouge Ciphertext Security* (see Definition 3.5) and prove that our construction satisfies it.

Adding Non-Interactive Zero-Knowledge Proof. To secure our encryption scheme against above attacks, we require that the encryptor attaches a non-interactive zero-knowledge proof (NIZK). We construct a tailored NIZK proof system that is based on the well known Schnorr proof of knowledge construction [50]. In particular, the encryptor needs to prove that it knows a random key k as well as ElGamal randomness u such that k^* is a valid punctured key for k at index i and ct is a valid ElGamal encryption of g^k. The proof is tagged to the ciphertext through the random oracle, which allows us to prevent both attacks, as the adversary cannot change any element in the ciphertext without corrupting the proof. We remark that in our CCA security proof, we need to use an extractor that extracts the witnesses (k, u) from the proof. As we need to run the extractor for a polynomial number of queries of the batch decryption oracle, we need to avoid using the rewinding technique, as otherwise the reduction's runtime blows up exponentially (see [51] for more details). In order to overcome this, similar to [17], we can use techniques from [26] to prove non-malleability in Schnorr Signed ElGamal Encryption using a straight-line extractor in the algebraic group model (AGM).

3 Preliminaries

Notation. We denote the security parameter as $\lambda \in \mathbb{N}$ and 1^λ as its unary representation. To assign expression y to variable x we write $x \leftarrow y$ and $x \xleftarrow{\$} S$ for the uniform random sampling of a value x from a set S. For an algorithm A, we denote by $y \leftarrow A(x; r)$ the execution of A on input x with randomness r that outputs y. We usually omit the randomness and write $y \xleftarrow{\$} A(x)$ to indicate execution of A with uniform randomness. By $[n]$ for a positive integer n, we denote the set of integers $\{1, \ldots, n\}$. Furthermore, we write $\{x_i\}_{i \in S}$ as a shorthand for the set $\{x_i \mid i \in S\}$. We use L_i to denote the Lagrange coefficient for some set S evaluated at 0 such that $L_i = \prod_{j \in S, j \neq i} \frac{-x_j}{x_i - x_j}$ where the set S is clear from context. For simplicity, we assume that shareholders in a (t, n)-threshold cryptosystem have participant indices in $[n]$.

Our construction relies on *bilinear pairing groups*. Given groups $\mathbb{G}_1$, $\mathbb{G}_2$ with generators g_1 and g_2 as well as $\mathbb{G}_T$ of prime order p a bilinear pairing is a function $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ that satisfies bilinearity, non-degeneracy and is efficiently computable. We always write $\mathbb{G}_1$ and $\mathbb{G}_2$ as multiplicative groups. In some cases, we use additive notation for $\mathbb{G}_T$ to enhance readability.

Key-Homomorphic Puncturable PRFs. A *puncturable* pseudorandom function PRF is a PRF with additional algorithms that allow to puncture the PRF and evaluate it using the punctured key. Puncturing with respect to a key k and an index i yields a punctured key k^*, which can in turn be used to evaluate the PRF on any index but the punctured one. We require two additional properties from our PRF: (1) The PRF should be *additively key-homomorphic* and (2) given that the PRF key k is a field element from some prime field $\mathbb{Z}_p$, it should be possible to *evaluate the PRF "in the exponent"* (i.e., using $K = g^k$).

Definition 3.1 (Puncturable Pseudorandom Functions). A puncturable pseudorandom function family on key space $\mathcal{K} = \{\mathcal{K}_\lambda\}_{\lambda \in \mathbb{N}}$, exponent key space $\mathcal{G} = \{\mathcal{G}_\lambda\}_{\lambda \in \mathbb{N}}$, domain $\mathcal{X} = \{\mathcal{X}_{\lambda,n}\}_{\lambda,n \in \mathbb{N}}$ and range $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of a tuple of PPT algorithms $\mathsf{PRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Puncture}, \mathsf{Eval}, \mathsf{ExpEval}, \mathsf{PEval})$ such that:

- $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, n)$. Setup takes the security parameter λ as well as the domain parameter n as input an outputs public parameters pp.
- $k \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{pp})$. KeyGen takes the public parameters as input and returns a key $k \in \mathcal{K}_\lambda$.
- $k^* \leftarrow \mathsf{Puncture}(\mathsf{pp}, k, i^*)$. Puncture is a deterministic algorithm that, given public parameters pp, a key $k \in \mathcal{K}_\lambda$ and an index $i^* \in \mathcal{X}_{\lambda,n}$, returns a punctured key k^*.
- $y \leftarrow \mathsf{Eval}(\mathsf{pp}, k, i)$. Eval is a deterministic algorithm that takes as input the public parameters pp, a key $k \in \mathcal{K}_\lambda$ and an index $i \in \mathcal{X}_{\lambda,n}$ from the domain and outputs $y \in \mathcal{Y}_\lambda$.

- $y \leftarrow \mathsf{ExpEval}(\mathsf{pp}, K, i)$. ExpEval is a deterministic algorithm that takes as input the public parameters pp, a key $K \in \mathcal{G}_\lambda$ as well as an index $i \in \mathcal{X}_{\lambda,n}$ from the domain. It outputs the result $y \in \mathcal{Y}_\lambda$.
- $y \leftarrow \mathsf{PEval}(\mathsf{pp}, k^*, i^*, i)$. PEval is a deterministic algorithm that takes as input the public parameters pp, a punctured key k^*, an index $i^* \in \mathcal{X}_{\lambda,n}$ and an index $i \in \mathcal{X}_{\lambda,n}$ with $i \neq i^*$. It outputs $y \in \mathcal{Y}_\lambda$.

The pseudorandomness and key-homomorphism properties are as in [23]. In short, we require that $\mathsf{Eval}(\mathsf{pp}, k, i)$ looks pseudorandom as long as k is not revealed, even given a punctured key k^* on index i.

For key-homomorphism we require that, for any $k_1, k_2 \in \mathcal{K}$, it holds that $\mathsf{Eval}(\mathsf{pp}, k_1 + k_2, i) = \mathsf{Eval}(\mathsf{pp}, k_1, i) + \mathsf{Eval}(\mathsf{pp}, k_2, i)$. The definition of pseudorandomness and the construction of our PRF are given in Appendix A.

Threshold Homomorphic Encryption. A threshold homomorphic encryption protocol THE is a tuple of PPT algorithms $\mathsf{THE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Combine})$. KeyGen generates a public key pk and n secret key shares $\{\mathsf{sk}_\ell\}_{[n]}$ distributed to decryption servers. Given a ciphertext $c \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, m)$, any server can derive a decryption share $d_\ell \xleftarrow{\$} \mathsf{Dec}(\mathsf{sk}_\ell, c)$. If a set S of at least t servers provide their decryption shares, the message m can be recovered using $m \leftarrow \mathsf{Combine}(\mathsf{pk}, \{d_\ell\}_S, S, c)$. We require our THE scheme to be correct, multiplicatively message-homomorphic and IND-CPA secure. For formal definitions we refer to the full version [12].

3.1 Batched Threshold Encryption

We introduce the definition of a Batched Threshold Encryption (BTE) scheme following the work of Choudhiro et al. [17] and Dujmovic et al. [23] with some minor modification.

Definition 3.2 (Batched Threshold Encryption). A *Batched Threshold Encryption* scheme (BTE) consists of a tuple of PPT algorithms $\mathsf{BTE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Verify}, \mathsf{BatchDec}, \mathsf{Combine})$ with the following syntax.

- $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, B_{\max})$: This algorithm initializes the scheme, receiving the security parameter $\lambda \in \mathbb{N}$, and the maximum batch size $B_{\max}$. It produces the public parameters pp which are implicit inputs to all subsequent algorithms.
- $(\mathsf{pk}, \{\mathsf{sk}_\ell\}_{\ell \in [n]}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, n, t)$: The key generation algorithm takes the security parameter λ, the total number of parties n, and the threshold t. It returns a public key pk along with a set of secret key shares $\{\mathsf{sk}_i\}_{i \in [n]}$.
- $(c, \pi) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, m, i)$: Given a public key pk a message m and an index i, the encryption algorithm outputs the ciphertext c for batch position i, along with a proof π.
- $\{1, 0\} \leftarrow \mathsf{Verify}(\mathsf{pk}, c, \pi)$. Verify is a deterministic algorithm that takes as input a public key pk, a ciphertext c and a proof π. It outputs 1 if the proof is valid and 0 otherwise.
- $d_\ell / \bot \leftarrow \mathsf{BatchDec}(\mathsf{sk}_\ell, \{c_i\}_{i \in B})$: Utilizing a secret key share sk_ℓ and a batch of ciphertexts $\{c_i\}_{i \in B}$ where $|B| \leq B_{\max}$, the decryption algorithm generates a decryption share d_ℓ or returns an error symbol $\bot$.
- $\{m_i / \bot\}_{i \in [B]} \leftarrow \mathsf{Combine}(\mathsf{pk}, \{d_\ell\}_{\ell \in S}, S, \{c_i\}_{i \in [B]})$: The combining algorithm takes the public key pk, a set of decryption shares $\{d_\ell\}_{\ell \in S}$ with $S \subseteq [n]$ and $|S| \geq t$, and a batch of ciphertexts $\{c_i\}_{i \in [B]}$. It outputs the decrypted messages $\{m_i\}_{i \in [B]}$ or an error symbol $\bot$.

We require that $B_{\max}$, n and t are polynomial in λ.

For sake of simplicity, we work in the coordinated setting, where we assume that all ciphertexts in a batch have unique indices i. We elaborate on techniques to remove this assumption in Section 5.3.

Definition 3.3 (Correctness of BTE). A Batched Threshold Encryption scheme BTE is correct if for all $\lambda, n, t, B_{\max} \in \mathbb{N}$ where $n \geq t$, all $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(\lambda, B_{\max})$, all $(\mathsf{pk}, \{\mathsf{sk}_\ell\}_{\ell \in [n]}) \xleftarrow{\$} \mathsf{KeyGen}(\lambda, n, t)$, all $B \in [B_{\max}]$, all $(m_1, \ldots, m_B) \in \mathcal{M}_\lambda^B$, all $S \in [n]$ where $|S| \geq t$, it holds that

$$\forall i \in [B]: \ \mathsf{Verify}(\mathsf{pk}, c_i, \pi_i) = 1 \quad \text{and}$$
$$\mathsf{Combine}(\mathsf{pk}, \{d_\ell\}_{\ell \in S}, S, \{c_i\}_{i \in [B]}) = \{m_i\}_{i \in [B]},$$

where $(c_i, \pi_i) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, m_i, i)$ for all $i \in [B]$ and $d_\ell \leftarrow \mathsf{BatchDec}(\mathsf{sk}_\ell, \{c_i\}_{i \in [B]})$ for all $\ell \in S$.

Efficiency. We require that the per-party communication complexity of a BTE scheme is $o(B)$ (i.e. sublinear in the batch size B). This excludes trivial constructions, where each server just sends a partial decryption of every ciphertext in the batch as in standard threshold encryption.

CCA-Security. We model security of a BTE scheme against Chosen-Ciphertext Attacks (CCA) using the security game Game-B-IND-CCA defined in Figure 1. This game is a standard game-based definition of threshold IND-CCA security, adapted to the batched setting (B-IND-CCA). First, the adversary statically corrupts up to $t-1$ parties C and receives their secret key shares $\{\mathsf{sk}_\ell\}_{\ell \in C}$. After proposing two messages m_0 and m_1, he receives a challenge ciphertext $c^\star$ which is an encryption of one of the messages. The adversary wins the game by guessing correctly, whether the challenge encryptes m_0 or m_1. The adversary gets access to a batch decryption oracle $O^{\text{b-dec}}$, which allows him to query for batch-decryption shares on behalf of honest parties for ciphertext-batches of its choice. The only restriction is that the adversary cannot query for too many decryption shares of any batch containing the

```
Game-B-IND-CCA_$\mathcal{A}$$(1^\lambda)$
$c^\star \leftarrow \perp; ctr \leftarrow 0; b \xleftarrow{\$} \{0,1\}$
$\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, B_{\max})$
$(\mathsf{pk}, \{\mathsf{sk}_\ell\}_{[n]}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, n, t)$
$(C, \mathsf{st}_1) \xleftarrow{\$} \mathcal{A}_1(1^\lambda, \mathsf{pp}, \mathsf{pk}, n, t)$
if $C \not\subseteq [n] \vee |C| \geq t$ then return $0$
$(m_0, m_1, i, \mathsf{st}_2) \xleftarrow{\$} \mathcal{A}_2^{O^{\text{b-dec}}}(\mathsf{st}_1, \{\mathsf{sk}_\ell\}_C)$
$(c^\star, \pi) \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b, i)$
$b' \xleftarrow{\$} \mathcal{A}_3^{O^{\text{b-dec}}}(\mathsf{st}_2, c^\star, \pi)$
return $b \stackrel{?}{=} b'$
```

```
Oracle $O^{\text{b-dec}}(\ell, \{(c_i, \pi_i)\}_{i \in [B]})$
if $c^\star \in \{c_i\}_{[B]}$ then
  $ctr \leftarrow ctr + 1$
  if $ctr \geq t - |C|$ return $\perp$
for $i \in [B]$ do
  if $\mathsf{Verify}((\mathsf{pp}, \mathsf{pk}, c_i), \pi_i) = 0$
   then return $\perp$
return $\mathsf{BatchDec}(\mathsf{sk}_\ell, \{c_i\}_{[B]})$
```

```
Game-Rogue_$\mathcal{A}$$(\lambda)$
$\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, B_{\max}); (\mathsf{pk}, \{\mathsf{sk}_\ell\}_{[n]}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, n, t)$
$(m, i, \mathsf{st}) \xleftarrow{\$} \mathcal{A}_1(1^\lambda, \mathsf{pp}, B_{\max}, \mathsf{pk}, \{\mathsf{sk}_\ell\}_{[n]})$
$(c_i, \pi_i) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, m, i)$
$(B, S, \{(c_j, \pi_j)\}_{j \in [B] \setminus \{i\}}) \xleftarrow{\$} \mathcal{A}_2(\mathsf{st}, c_i)$
if $B > B_{\max} \vee i \notin [B] \vee S \not\subseteq [n] \vee |S| < t$ then return $0$
if $\exists j \in [B]$ s.t. $\mathsf{Verify}(\mathsf{pk}, c_j, \pi_j) = 0$ then return $0$
$\{d_\ell\}_{\ell \in S} \leftarrow \{\mathsf{BatchDec}(\mathsf{sk}_\ell, \{c_j\}_{j \in [B]})\}_{\ell \in S}$
$\{m_j\}_{j \in [B]} \xleftarrow{\$} \mathsf{Combine}(\mathsf{pk}, \{d_\ell\}_{\ell \in S}, S, \{c_j\}_{j \in [B]})$
if $m_i \neq m$ return $1$ else return $0$
```

Figure 1: Security games of BTE.

challenge ciphertext.[4] This definition covers the requirement of *pending transaction privacy*, as it ensures that the adversary cannot learn anything about the challenge ciphertext, even if it is allowed to decrypt batches of ciphertexts through $O^{\text{b-dec}}$.

Definition 3.4 (B-IND-CCA-security of BTE). A BTE scheme is B-IND-CCA secure if for all PPT adversaries $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that $\Pr[\mathsf{Game\text{-}B\text{-}IND\text{-}CCA}_{\mathcal{A}}^{\mathsf{BTE}}(1^\lambda) = 1] \leq 1/2 + \mathsf{negl}(\lambda)$ where $\mathsf{Game\text{-}B\text{-}IND\text{-}CCA}_{\mathcal{A}}$ is defined in Figure 1.

Rogue Ciphertext Security. Dujmovic et al. [23] introduce the notion of Rogue Puzzle Attacks, which is a class of attacks on batched TLP protocols. In rogue puzzle attacks, the adversary injects maliciously crafted puzzles into the batch to disrupt the batch-solving of honest puzzles. We extend this notion to "Rogue Ciphertext Attacks" in the context of Batched Threshold Encryption. In this attack, the adversary tries to inject some ciphertexts into a batch such that batch-decryption of the honest ciphertexts fails or yields incorrect messages. We model security against rogue ciphertext attacks with the security game Game-Rogue defined in Figure 1.

Definition 3.5 (Rogue Ciphertext Secuirty of BTE). A BTE scheme is secure against rogue ciphertext attacks if for all PPT adversaries $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2)$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that $\Pr[\mathsf{Game\text{-}Rogue}_{\mathcal{A}}^{\mathsf{BTE}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda)$ where $\mathsf{Game\text{-}Rogue}_{\mathcal{A}}$ is defined in Figure 1.

4 Building Blocks

In this section, we present constructions for our two building blocks PRF and THE.

[4]If the adversary learns $t - |C|$ decryption shares, it can win the game trivially by combining with the $|C|$ shares of the corrupted parties

Key-homomorphic Puncturable PRFs. For our building block of key-homomorphic puncturable PRF with exponent evaluation, we adapt the construction in [23] to our needs. The full construction is given in Appendix A, and we provide here only a high-level overview. In particular, we modify the construction such that the PRF can be evaluated not only with the key k but also with the key in the exponent, i.e., g^k. To achieve this, we change the construction from [23] by publishing more elements in the setup. Furthermore, the PRF construction of [23] is based on a symmetric pairing group setup, which we need to avoid, since we want to use the punctured PRF along with ElGamal which is not secure in a symmetric pairing group.[5] For our construction, we rely on the pairing-based construction with quadratic setup, which is secure under a variant of the decisional bilinear Diffie-Hellman (DBDH) assumption. We note that [23] present a second pairing based construction with linear setup. In our analysis, we focus on the first construction, as their second construction is based on the decisional n-power Diffie-Hellman assumption [9], which is less standard.[6] We expect that our constructions can be easily adapted to the second construction as well.

It is important to highlight that both constructions from [23] have a polynomially bounded domain size, which is restricted by the size of the setup. Looking ahead, we will require that during encryption, clients choose an index from this domain to encrypt a message. The choice of index is important, as all ciphertexts in a batch must not have colliding indices. For now we will assume that there is some form of coordination between encryption clients, similar to the batched encryption scheme from [17]. However, we present several solutions to remove this coordination assumption in Section 5.3.

[5]Since DDH is not a hard problem with symmetric pairings $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, we will use asymmetric pairings instead and require that DDH is hard in one of the source groups (here written as $\mathbb{G}_2$).

[6]The assumption has been proven to hold in the bilinear generic group model by [8].

Threshold ElGamal Construction. As a building block for our batched threshold encryption protocol, we require a threshold homomorphic encryption scheme, which we instantiate with an IND-CPA-secure thresholdized version of ElGamal encryption [27]. Our version of threshold ElGamal encryption works by Shamir-sharing the ElGamal secret key sk into $\{\mathsf{sk}_\ell\}_{[n]}$. We write $\mathsf{pk}_\ell = g^{\mathsf{sk}_\ell}$ for the parties' individual public keys and set the overall public key to be $\mathsf{pk} = (g^{\mathsf{sk}}, \{\mathsf{pk}_\ell\}_{[n]})$. Given a ciphertext $c = (A, B) = (g^u, \mathsf{pk}^u \cdot m)$ we perform a partial decryption by computing $d_\ell \leftarrow (A^{\mathsf{sk}_\ell})$. One can then combine a set of at least t partial decryptions to recover the message m using Lagrange interpolation.

$$m \leftarrow B / \prod_{\ell \in S} d_\ell^{L_\ell} \left[= m \cdot \mathsf{pk}^u / g^{u \sum_{\ell \in S} \mathsf{sk}_\ell L_\ell} = m \cdot g^{\mathsf{sk} \cdot u} / g^{\mathsf{sk} \cdot u} = m \right]$$

Theorem 4.1 *The above threshold ElGamal encryption scheme is correct, homomorphic and IND-CPA secure.*

A proof sketch of Theorem 4.1 is given in the full version [12].

5 Our Batched Threshold Encryption Scheme

We first describe our construction in the coordinated setting in Section 5.1. Then, we show how one can use a trade-off between communication and computation complexity to optimize our construction to any specific setting in Section 5.2 before presenting in detail how we are able to remove coordination between encryptors (Section 5.3).

5.1 Construction

The construction is depicted in Figure 2 and described in detail below.

Setup and Key Generation. First, we setup the PRF and generate the pairing ensemble $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p)$. Here we also set the domain size of the PRF to $B_{\max}$, as every ciphertext in a batch will need to sample a unique index from the PRF domain. Note that PRF keys will be sampled from $\mathbb{Z}_p$ and exponent evaluation is possible using the secret key in the exponent of $\mathbb{G}_2$ (i.e., g_2^k). Further, we will use ElGamal encryption in $\mathbb{G}_2$ to encrypt PRF keys in the exponent. In the end, the key-space of the BTE construction is $\mathbb{Z}_p$ and the message space is $\mathbb{G}_T$.[7]

For key generation, we perform (t, n)-threshold ElGamal key generation, which is essentially a Shamir secret sharing of a master secret key belonging to a public key pk. The secret key shares $\{\mathsf{sk}_\ell\}_{\ell \in [n]}$ are distributed among the n decryption servers. To remove a trusted dealer, one can substitute this step with a DKG protocol, to jointly generate a sharing of a random master secret key.

[7]One can extend the message space to arbitrary bitstrings using standard encapsulation techniques.

Encryption. To encrypt a message m, a client first picks an index $i \in [B_{\max}]$ and samples a PRF key k. Then the client punctures k at index i to get k^*. With the punctured key k^*, the PRF can be evaluated under k^* at any index $j \neq i$. The message m is encrypted by masking it with the evaluation of the PRF under index i as $\gamma \leftarrow m + \mathsf{PRF.Eval}(k, i)$.[8] Next, the client encrypts g_2^k under pk using ElGamal encryption, yielding ElGamal ciphertext ct. Finally, the client constructs a NIZK proof π tagged to the ciphertext and the setup that proves knowledge of the key k as well as the randomness u used in the ElGamal encryption such that the punctured key is valid for k at index i and the ElGamal ciphertext is a valid ElGamal encryption of g_2^k using randomness u. The final ciphertext is the tuple $c = (i, k^*, \gamma, ct)$ with proof π.

Partial Decryption. Given a batch of B ciphertexts, a server can now compute a partial decryption share using BatchDec. To do so, the server first verifies that the proof π_i is valid for each ciphertext c_i in the batch. Then, the server aggregates all the ElGamal ciphertexts into $C = \prod_i^{[B]} c_i.ct$. As ElGamal encryption is additively homomorphic in the exponent, C is now an ElGamal encryption of $K = g_2^k = g_2^{\sum_i k_i}$, which is the sum of all the PRF keys in the exponent. Each server can now perform a partial decryption by releasing the threshold ElGamal decryption share d_ℓ for C under its secret key share sk_ℓ.

Decryption. When at least t servers have released their partial decryption shares $\{d_\ell\}_{\ell \in S}$, anyone can combine these shares using Lagrange interpolation and decrypt C into K. Given this information, one can decrypt any message m_i in the batch by computing

$$m_i \leftarrow \gamma_i + \sum_{j \neq i}^{[B]} \mathsf{PRF.PEval}(k_j^*, i) - \mathsf{PRF.ExpEval}(K, i) \quad (1)$$

First, note that $\gamma_i = m_i + \mathsf{PRF.Eval}(k_i, i)$. Because of the punctureability of the PRF, we get that $\mathsf{PRF.PEval}(k_j^*, i) = \mathsf{PRF.Eval}(k_j, i)$ for $j \neq i$. As we want all the PRF evaluations to cancel out, we need to subtract the evaluation of the sum of all the keys K at index i. Here is where we need the key-homomorphic property of the PRF as well as the ability to evaluate it *in the exponent*. Given that ElGamal is multiplicatively homomorphic in $\mathbb{G}_2$, it is also additively homomorphic in the exponent, so we know that $K = g_2^{\sum_i k_i}$. We subtract an exponent evaluation of the PRF under K to cancel out the sum of all the PRF evaluations on the left side of Equation 1 as well as the evaluation under k_i.[9] In summary, because of the exponent evaluation, we have for the last part of Equation 1

[8]Note that k^* is *not* sufficient to evaluate the PRF at index i.

[9]We rely on the observation that we can evaluate the PRF in the exponent, because ElGamal is multiplicatively homomorphic (i.e. additively homomorphic in the exponent), meaning we only learn $g_2^{\sum k_i}$ from the partial decryptions and not $\sum k_i$ directly. We avoid using additively homomorphic protocols such as Paillier [45] since they are less efficient to thresholdize.

```
$\mathsf{Setup}(1^\lambda, B_{\max})$
    // Setup the pairing ensamble
    return $\mathsf{pp} \xleftarrow{\$} \mathsf{PRF.Setup}(1^\lambda, B_{\max})$

$\mathsf{KeyGen}(1^\lambda, n, t)$
    $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \xleftarrow{\$} \mathsf{THE.KeyGen}(1^\lambda, n, t)$
    return $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]})$

$\mathsf{BatchDec}(\mathsf{sk}_i, \{c_i\}_{i\in[B]})$
    $C \leftarrow \prod_i^{[B]} ct_i$
    $d_i \leftarrow \mathsf{THE.Dec}(\mathsf{sk}_i, C)$
    return $d_i$
```

```
$\mathsf{Enc}(\mathsf{pk}, m, i)$
    $k \leftarrow \mathsf{PRF.KeyGen}(\mathsf{pp})$
    $k^* \leftarrow \mathsf{PRF.Puncture}(k, i)$
    $\gamma \leftarrow m + \mathsf{PRF.Eval}(k, i)$
    $u \xleftarrow{\$} \mathbb{Z}_p$
    $ct \leftarrow \mathsf{THE.Enc}(\mathsf{pk}, g_2^k; u)$
    $c \leftarrow (i, k^*, \gamma, ct)$
    $\chi \leftarrow (\mathsf{pp}, \mathsf{pk}, c)$
    $\omega \leftarrow (k, u)$
    $\pi \xleftarrow{\$} \Pi.\mathsf{Prove}(\chi, \omega)$
    return $(c, \pi)$
```

```
$\mathsf{Combine}(\mathsf{pk}, \{d_\ell\}_{\ell\in S}, S, \{c_i\}_{i\in[B]})$
    Parse $c_i$ as $(i, k_i^*, \gamma_i, ct_i)$
    $C \leftarrow \prod_i^{[B]} ct_i$
    $K \leftarrow \mathsf{THE.Combine}(\mathsf{pk}, \{d_\ell\}_{\ell\in S}, S, C)$
    for $i \in [B]$ do
        $m_i \leftarrow \gamma_i + \sum_{j\neq i}^{[B]} \mathsf{PRF.PEval}(k_j^*, i) - \mathsf{PRF.ExpEval}(d, i)$
    return $\{m_i\}_{i\in[B]}$

$\mathsf{Verify}(\mathsf{pk}, c, \pi)$
    return $\Pi.\mathsf{Verify}((\mathsf{pp}, \mathsf{pk}, c), \pi)$
```

Figure 2: Construction of the Batched Threshold Encryption scheme BTE.

that $\mathsf{PRF.ExpEval}(K, i) = \mathsf{PRF.Eval}(\sum_j^{[B]} k_j, i)$, which is equal to $\sum_j^{[B]} \mathsf{PRF.Eval}(k_j, i)$ due to the additive key-homomorphic property of the PRF. Hence, the PRF evaluations cancel out and we are left with m_i:

$$\begin{aligned} &m_i + \mathsf{Eval}(k_i, i) + \sum_{j\neq i}^{[B]} \mathsf{Eval}(k_j, i) - \sum_j^{[B]} \mathsf{Eval}(k_j, i) \\ =&m_i + \sum_j^{[B]} \mathsf{Eval}(k_j, i) - \sum_j^{[B]} \mathsf{Eval}(k_j, i) = m_i \end{aligned}$$

Non-Interactive Zero-Knowledge Proof. As we discussed our BTE construction uses a non-interactive proof system $\Pi = (\mathsf{Prove}, \mathsf{Verify})$, which we require to achieve IND-CCA security and Rogue Ciphertext Security. The idea is similar to the Schnorr non-interactive zero-knowledge proof [34]. That is, the prover proves knowledge of the PRF key k and the randomness u used in ElGamal encryption, such that:

- The punctured key in the ciphertext is consistent with k and the index i
- The ElGamal ciphertext ct is a valid ElGamal encryption in the exponent of g_2^k using randomness u.

This yields the witness $\omega = (k, u)$, with statement $\chi = (\mathsf{pp}, \mathsf{pk}, c)$, where $c = (i, k^*, \gamma, ct)$ and $ct = (A, B) = (g^u, \mathsf{pk}^u g_2^k)$. The proof is tagged to the setup, the public key, and the ciphertext. Intuitively, this means that modifying anything in the ciphertext invalidates the proof of knowledge of (k, u), which is crucial for CCA-security. We instantiate Π with a custom Schnorr-proof construction. We refer the reader to Appendix B for a detailed construction and analysis of the proof system Π.

Security. We prove the B-IND-CCA and Rogue Ciphertext Security of our BTE construction. The full proofs of the following two theorems can be found in the full version [12].

Theorem 5.1 (CCA-security of BTE). *The batched threshold encryption scheme BTE is B-IND-CCA-secure given the CPA-security of our thresholdized ElGamal in $\mathbb{G}_2$, the pseudorandomness of the PRF and the simulability and simulation-extractability of the proof system Π.*

Theorem 5.2 (Rogue Ciphertext Security of BTE). *The batched threshold encryption scheme BTE is secure against rogue ciphertext attacks given the soundness of the proof system Π, and the correctness of the PRF and threshold ElGamal.*

Intuitively, Rogue Ciphertext Security follows from the soundness of Π, because soundness guarantees that for any batch of ciphertexts with valid proofs, the ElGamal part ct_i of each ciphertext is a valid ElGamal encryption of $g_2^{k_i}$, while the punctured key k_i^* is a valid punctured key under the same key k_i for index i. Given a batch of ciphertexts for which the above statement holds, one can verify that the honest ciphertext in the batch decrypts to the correct message, given correctness of the PRF and correctness of Threshold ElGamal.

5.2 Optimizations

The BTE construction as presented in Section 5 is *very* efficient with respect to communication. In fact, a shareholder only needs to publish a *single* group element as partial decryption, independent of the batch size B. On the other hand, the BTE.Combine operation is computationally expensive, as it involves computing $O(B^2)$ pairings. This is the case because during Combine we need to compute B pairings per ciphertext. For every ciphertext, we need one pairing for the evaluation of the PRF with the combined key in the exponent $\mathsf{PRF.ExpEval}(K, i)$ as well as $B-1$ pairings for the punctured evaluations of the PRF with the punctured keys k'_j for $j \neq i$ $(\mathsf{PRF.PEval}(k'_j, i))$.

We introduce an optimization that trades increased size of the decryption shares released by the committee members for a significant reduction in the number of pairings required during Combine.

Splitting a Large Batch into Smaller Sub-Batches. Consider decryption of a batch of B ciphertexts. Instead of releasing a single partial decryption share for the entire batch, we split the batch into α sub-batches of size B/α. Setting e.g. $\alpha = \sqrt{B}$, each shareholder now releases $\sqrt{B}$ group elements (which is still sublinear). During aggregation, we now only have to perform $\sqrt{B}$ pairings for each ciphertext, decreasing the overall number of pairings to $O\big(B\sqrt{B}\big)$. This optimization is particularly useful, because it also proves beneficial to solving the coordination problem as we discuss in Section 5.3. We note that there is nothing magical about setting $\alpha = \sqrt{B}$. The parameter α can be chosen specifically for the concrete application, weighing the trade-off between communication and computation. When using α-subbatching, each partial decryption contains α group elements and the aggregation requires B/α pairings per ciphertext.

In our evaluation (Section 7), we show that the *computational overhead* on behalf of the shareholders for this optimization is very small, while showing a significant improvement in the efficiency of Combine.

5.3 Removing Coordination

In the above construction, we assume coordination among parties to ensure that every party owns a unique index. However, as discussed in Section 2, having coordination among parties would be cumbersome in practice.

Dujmovic et al. [23] introduced a method to convert any batched TLP scheme that requires unique indices to a non-coordinated scheme, which can also be applied to our batched threshold encryption scheme. However, their method is not desired in our scenario as it increases the size of ciphertext, motivating us to propose a different approach. We let every party samples one index, and model the probability of having index collision in a batch as a generalized birthday problem [2]. Additionally, we make use of the sub-batching optimization proposed in Section 5.2.

We propose a new technique to remove coordination in our BTE scheme. We divide the whole batch $[B]$ into α sub-batches of the size B/α, and guarantee that there will be no index collision within every sub-batch. Let each party sample a random index from $[N]$ during encryption, we sort the B ciphertexts by the occurrences of the indices, i.e. the ciphertexts with the most-repeated index will rank the first, and the ciphertexts with unique indices will rank the last. The ciphertexts are then distributed into the sub-batches according to their sorted order, so that ciphertexts with the same index are distributed into different sub-batches. The sub-batches each sized B/α are then batch decrypted and combined in the same way as the original construction.

With the above technique, it is guaranteed that there will be no index collision within any sub-batch if the most repeated index has no more than α occurrences, and consequently, the whole batch has no index collision. Now the probability of having index collision is reduced to the probability of having no less than $\alpha + 1$ parties sample the same index.

It is vital that the correctness follows straightforwardly from the original construction, and the security is also preserved. We provide the proof in Appendix C.

For practical settings, we guarantee 40 bits of statistical correctness by convention [23], meaning that the probability of index collision is smaller than 2^{-40}. Our scheme already provides 41 bits of statistical correctness when $B = 256, \alpha = 16, N = 256$. Here we present the probability of having index collision for different settings in Table 1. We calculate all the probabilities using the exact formula $P_B^{(\alpha+1)} = 1 - (1 - 1/N^{\alpha})^{\binom{B}{\alpha+1}}$. The detailed definition of $P_B^{(\alpha+1)}$ is provided in Appendix C, and we transform it to $-\log_2 P$ for better readability. To give an intuition, we count the probability that no set of $\alpha+1$ parties all sample the same index, and model $P_B^{(\alpha+1)}$ from this perspective.

B	$\boldsymbol{\alpha}$	**N**	$-\log_2 \mathbf{P}$
256	16	256	41
128	11	384	40
64	16	64	46
64	8	320	32
16	8	64	35

Table 1: Probability of index collision for different settings.

This table indicates that we can easily gain practical statistical correctness when B is relatively large, without further increasing N or α. Since the index collision probability is dependent on B, we achieve better correctness as B increases. For smaller B, it is always possible to achieve the desired correctness level by increasing α or N. We could also consider using the technique from [23] for small B, depending on specific requirement of the applications. As a reference, the time complexity of the matching technique in [23] is $O(B \cdot d\sqrt{B})$, where d is the number of indices sampled by each party. In our construction, an efficient sorting algorithm has a time complexity of $O(B \log B)$. Our construction has constant ciphertext size, while theirs is linear in d.

6 Attacks and Mitigations

Considering that we aim to present a practical protocol that can be instantiated on blockchains to prevent MEV attacks, we need to deal with other practical properties aside from just security. In particular, we want to ensure that malicious actors cannot prevent the protocol from functioning correctly. To this end, we address three practical attacks that are relevant in the context of MEV prevention.

Selective Decryption Attacks. Consider an adversary who is part of the decryption committee and wants to submit a transaction tx performing a trade on a decentralized exchange (DEX). In the MEV-prevention setting, the adversary encrypts tx using $\mathsf{BTE.Enc}$ and submits the ciphertext c_{tx} to the blockchain, thereby committing to the transaction. A fixed amount of time later, the committee will release decryption shares for the block that contains c_{tx} and subsequently tx gets executed. An adversary who is part of the decryption committee can perform the following attack:

1. Submit c_{tx} to the blockchain and wait until just before the decryption shares are released. In this time, the adversary monitors prices on the DEX and observes whether the value of the trade has increased or decreased.
2. If the adversary now determines that the trade is profitable, it releases their decryption share. In this case the transaction is executed and the adversary profits.
3. If the adversary determines that the trade is not profitable given the new price, it releases a *malformed* decryption share.[10] This causes decryption to fail or produce garbage, which means the trade is *not* executed.

We propose to solve this problem by adding public *share-verifiability* to the decryption shares released by committee members, adding a proof π_{share} to the decryption shares that can be verified by anyone. This prevents the above attack, as any adversary who releases a malformed decryption share will be identified and can be penalized. On top of that, as long as at least t servers release valid shares, one can simply discard the invalid shares and proceed with decryption *without* any need for honest servers to rerun the partial decryption. We can construct an efficient proof system Π_{share} by essentially proving knowledge of the secret key share sk_ℓ such that d_ℓ is a valid ElGamal decryption for the batch under sk_ℓ using Schnorr proofs. As an additional benefit, the public share-verifiability also protects against generic denial of service attacks that involve releasing malformed decryption shares.

Amplified Decryption Attacks. In our construction $\mathsf{Combine}$ is the most expensive operation, as it involves computing pairings. Remember that $\mathsf{Combine}$ can be performed by anyone in order to decrypt a batch of ciphertexts, *after* the committee members have released their decryption shares. Ideally we would like to outsource this computationally expensive operation to a dedicated service with adequate computational resources and parallelism, who is tasked with performing $\mathsf{Combine}$ and publishing the decrypted transactions on the Blockchain afterwards.

The issue here is that we do not want to trust this service, as it could potentially produce an arbitrary batch of transactions and claim they are the result of decryption. The consequence of this is that the system reverts to a degraded mode, where *everyone* has to perform the expensive computation in $\mathsf{Combine}$, or even worse, it would have to be carried out inside a smart contract. This is not a problem specific to our construction but also applies, to a lesser extent, to existing solutions like [17] who similarly perform expensive pairings for aggregation.[11]

We propose to establish an additional property of *verifiable aggregation*, which adds a proof π_{agg} to the output of $\mathsf{Combine}$ that can be efficiently verified to confirm correct overall decryption of ciphertexts. We would like to achieve this property while burdening minimal additional computational overhead on the server performing $\mathsf{Combine}$, therefore excluding obvious solutions such as proving correct aggregation via SNARKs.

We present an idea to solving this problem already during encryption, without additional overhead to the decryption process. Suppose during encryption a user additionally commits to the message m using a cryptographic commitment scheme $(\mathsf{com}, \mathsf{op}) \xleftarrow{\$} \mathsf{Commit}(m)$ where com is the commitment and op is the opening value. The client encrypts the message and the opening $m||\mathsf{op}$ instead of just the message and also adds com to the ciphertext. Finally, the client adds a NIZK proof of valid construction (i.e. that the ciphertext encrypts the opening and the message hidden in com). This solves our problem by allowing the aggregator who performs $\mathsf{Combine}$ to decrypt the batch and publish $m||\mathsf{op}$ for all transactions in the batch, essentially involving no computational overhead. Anyone (including a smart contract) can then efficiently verify the commitment $\mathsf{Verify}(\mathsf{com}, m, \mathsf{op})$ for all transactions in the batch. This approach outsources the burden of proof to the encrypting clients, which is reasonable because encryption is already very efficient. We note that one could use SNARKs for the encryption proofs, but finding even more efficient solutions is an interesting open problem.

Denial of Service (DoS) and Censorship Attacks. An adversary could launch an expensive DoS attack by submitting a large number of transactions with high transaction fees, thereby causing the network to be congested. This could be easier in encrypted mempool systems since the transactions can be invalid, and in [17] some solutions are proposed. Another possible attack faced by some BTE constructions is the censorship attack. Consider a scenario, where every ciphertext is encrypted under one (or more in some other constructions) public index, and the capacity of ciphertexts with the same index in a block is limited, say $\sqrt{B}$ by default in our construction and 1 in [17]. An adversary could censor a target transaction by submitting many transactions with the same index as the target transaction. To the best of our knowledge, this attack was not considered by previous work.

Our construction handles this attack dynamically, since our remove-coordination mechanism does not compromise correctness even if collisions appear. As introduced in Section 5.3, after a batch of B ciphertexts is determined, a sorting

[10] In our construction, the adversary could release a random group element in $\mathbb{G}_2$

[11] The construction in [17] still requires $O(B)$ pairings for aggregation, which is too expensive for on-chain computation.

algorithm is used to distribute them into sub-batches. This algorithm is run by all committee members and is deterministic. It guarantees that even when collisions happen, which is very unlikely when there is no adversary, all ciphertexts will still be decrypted by increasing the number of sub-batches to the largest amount of collisions that appear on a single index. We consider this a 'downgraded' mode, as it increases communication overhead to maintain perfect correctness. In the worst case, our BTE construction will decrypt each ciphertext individually, and the communication is the same as a naive threshold decryption.

7 Experimental Evaluation

To establish concrete efficiency of our BTE scheme as an MEV prevention measure, we implemented our construction in Go and analyze its performance. Our implementation makes use of the dedis/kyber [21] library for the cryptographic primitives and pairing operations. The source code is public.[12]

Testbed. Our experiments were conducted on a desktop machine equipped with an AMD Ryzen 7 5800x 8-core CPU and 32GB of DDR4 RAM. All experiments run without parallelism enabled if not stated otherwise. We use the BLS12-381 curve for pairing operations because its widespread adoption. We switch the groups in the construction, as group operations in $\mathbb{G}_1$ are generally more efficient on BLS12-381. We provide micro-benchmarks for the group operations in Figure 3.

Operation	$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{G}_T$
Multiplication	0.001 ms	0.002 ms	0.006 ms
Exponentiation	0.097 ms	0.207 ms	0.471 ms
Pairing	-	-	0.699 ms

Figure 3: Micro-benchmarks of the group operations in time per operation on our testbed.

One-time Setup. Similar to [17], our construction requires a one-time setup. While their setup is a KZG-setup, our setup is special to the PRF used in the construction and directly related to the domain size. Our setup though does not need to be performed by the committee itself and can be done by anyone or any set of parties using MPC while the one-time setup in [17] is tied to the secrets held by the committee members and thus needs to be executed by the committee. In our evaluation, we consider a trusted setup.

Key Generation. One benefit of our construction is that the threshold ElGamal we use only requires a standard Shamir-shared dlog-keypair. We expect that one can use existing DKG protocols [16, 20, 29] to remove the trusted dealer. This also opens the door for efficient protocols to support dynamically changing committees. Choudhuri et al. [17] explore the possibility of using dynamic proactive secret sharing protocols such as [32] to combat committee churn. We expect this to be applicable to our construction as well.

Criteria. We choose the three most significant criteria for MEV-prevention, namely *encryption time* (Enc), *partial decryption time* (BatchDec), and *aggregation time* (Combine). We also evaluate the impact of the optimization we present in Section 5.2. In particular, we evaluate the scheme without any further optimization (henceforth called normal), with subbatching for $\alpha = \sqrt{B}$ (Opt-1) and with $\alpha = 2 \cdot \sqrt{B}$ sub-batches (Opt-2). On top of that, we highlight the ciphertext size and partial decryption size per party for the different optimizations. We compare results for varying batch sizes up to $B = 512$, which exceeds typical transactions per block rates.

Comparison to [17]. We elect to compare our results to the construction from Choudhuri et al. [17], as it is the most closely related work targeted at MEV-prevention, and they provide measurements for the same curve BLS12-381. It is important to note that their measurements are based on an implementation in Rust and performed on a slightly less powerful machine. We choose to compare the results anyway, as they suffice to highlight practical advantages and disadvantages of both constructions.

Encryption. Encryption performance is independent of the batch size and does not require any pairing operations. The most expensive operation during encryption is the generation of the NIZK proof for CCA security, for which we provide an efficient instantiation. In total, we measure an average encryption time of 1.58ms, while [17] achieves around 6ms. The ciphertext consists of the index i, which can be represented using 2 bytes, 3 group elements in $\mathbb{G}_1$ (the punctured key and the ElGamal ciphertext) and one group element from $\mathbb{G}_T$ (which is γ). This amounts to a total of 722 bytes per ciphertext while [17] achieves 370 bytes. The proof for CCA-security consists of 3 elements from $\mathbb{G}_1$ and 2 field elements, which totals to 208 bytes.[13]

Partial Decryption. For partial decryption, a committee member needs to verify all CCA-proofs in the batch, aggregate the ElGamal ciphertexts and compute a partial decryption share. The most expensive operation is the verification of the CCA-proofs. Unlike [17], we do not require any pairing operations for the verification of the proofs nor for the generation of the decryption share.

[12] https://zenodo.org/records/14672008

[13] The proof size is not as relevant, as it does not need to be persisted on-chain.

Batch Size	[17]	normal	Opt-1	Opt-2
8	41.5	8.2	8.7	9.0
32	173.4	31.7	32.2	32.7
128	678.11	78.7	79.6	80.1
512	2818.6	293.5	295.4	297.0

Figure 4: Partial decryption Time in ms. The comparison to [17] is based on different implementation and hardware.

We compare our measurements for partial decryption with different degrees of optimization to the results from [17] in Figure 4. The overall takeaway here is that in our MEV-prevention scheme the partial decryption is very efficient, as it does not need any pairings. An interesting observation is the very small increase in partial decryption time for the optimizations Opt-1 and Opt-2. This is because the most expensive operation is the verification of the CCA-proofs, which is not affected by the optimizations. The aggregation of ElGamal ciphertexts and computation of the partial decryption shares is comparatively cheap.

For normal partial decryption, every committee member releases a single element from $\mathbb{G}_1$, which amounts to 48 bytes per party. The construction from [17] requires 80 bytes per party, as they publish an additional field element. As our optimizations are essentially trade-offs between computational efficiency of Combine and the size of partial decryption shares, we get larger sizes for Opt-1 and Opt-2. For Opt-1 we need to release $\sqrt{B}$ group elements. For $B = 512$ this rounds to 22 group elements or 1056 bytes per party. In Opt-2 we need to release $2 \cdot \sqrt{B}$ group elements, which rounds to 45 group elements or 2160 bytes per party. We believe that this trade-off is reasonable, given the significant improvement of Combine efficiency for the Opt-2 optimization, especially for larger B.

Aggregation. We expect the aggregation of partial decryption shares and subsequent decryption of the batch to be the most expensive operation in our scheme, which is why we focused on optimizing this operation. The results are presented in Figure 5.

Batch Size	[17]	normal	Opt-1	Opt-2
8	41.9 ms	55.0 ms	28.8 ms	15.4 ms
32	165.0 ms	769.0 ms	160.3 ms	74.2 ms
128	781.4 ms	10.9 s	1.0 s	500.8 ms
512	3.5 s	169.4 s	7.7 s	3.8 s

Figure 5: Aggregation time given a batch of ciphertexts of size B and the according decryption shares. The comparison to [17] is based on different implementation and hardware.

To interpret the results we recall that the Ethereum produces one block approximately every 12 seconds. Supposing that every result below 12 seconds can be considered acceptable, we can see that our unoptimized construction can handle batches up to $B = 128$ well enough. For larger batches up to $B = 512$ we still get good aggregation times of around 3.8 seconds for Opt-2. We argue though that these results are still acceptable, as the measurements are without any parallelism. We measure a parallelized implementation of Opt-2 to take around 439 ms per Combine for $B = 512$ on the same CPU, while parallelized Opt-1 achieves 894 ms. On top of that, we can expect that the aggregation only needs to be performed by a very small amount of powerful servers, when employing the verifiability measures described in Section 6.

On-chain Storage. We analyze the on-chain storage and cost estimates for our construction. Every ciphertext needs to be stored on-chain, which has a constant size of 722 bytes per ciphertext. This would introduce approximately 0.33 USD cost per ciphertext on Ethereum[14], and much less on layer 2 solutions. Without additional verification mechanisms, the decryption shares also need to be stored on-chain, which works the same for naive threshold decryption schemes. Since this might blow up the on-chain storage when the committee size is large, we could use the encryption with commitments idea discussed in Section 6. This slightly increases the size of a transaction by adding a commitment, but in turn only 48 bytes for a single aggregated decryption key need to be stored on-chain for every sub-batch instead of all decryption shares. Alternatively, one could use a SNARK to prove that the aggregated key comes from valid decryption shares, and persist the proof instead of the key shares on-chain.

Practical advantages of requiring no Epoch Setup. In contrast to [17] our construction does not require any per-epoch setup. Apart from less communication and computation for the decryption committee, this fact comes with a number of advantages for the MEV-application.

First, clients that want to submit a protected transaction can encrypt independent of the current epoch setup. This means that (1) they do not need to wait for the committee to release the new epoch setup and (2) their ciphertexts *stay valid*, even if they do not make it inside a block in the current epoch, and can be included in following epochs. Both of these properties are not fulfilled by the construction in [17], as encryption is tied to one epoch setup. Second, because of the lack of epoch setup, we can practically support dynamic batch sizes. Consider a scenario where there is an unusually large amount of transactions inside one epoch. If, at the end of the epoch, the amount of ciphertexts exceeds $B_{\max}$, we can simply split the ciphertexts into two or more sub-batches similar to the optimizations discussed above. The honest parties in the decryption committee can observe the ciphertexts on the blockchain and release decryption shares for all resulting batches. This way we can still decrypt all ciphertexts atomically, which allows our scheme to be instantiated with significantly lower $B_{\max}$

[14]According to the gas and eth price on 25.11.2024 using calldata.

in practice than the construction from [17]. Coincidentally, this helps both the efficiency of Combine and the collision problem (Section 5.3), as there are more possibilities to sort transactions into collision-free sub-batches.

8 Acknowledgements

This work is funded by the European Research Council (ERC) under the European Union's Horizon 2020 and Horizon Europe research and innovation programs (grant CRYPTOLAYER-101044770). Furthermore, this research is supported by the German Federal Ministry of Education and Research, as well as the Hessen State Ministry for Higher Education, Research and the Arts, through their joint sponsorship of the *National Research Center for Applied Cybersecurity ATHENE*.

9 Ethics Considerations

While our research does not involve human subjects or access public data, we recognize the potential impacts of our work on blockchain users, developers, and the broader public. We have engaged in a thorough ethical analysis consistent with the guidelines provided by USENIX Security and the principles delineated in The Menlo Report [3].

Our research contributes positively to blockchain industry by mitigating the MEV problem, enhancing user privacy, and reducing financial risks. This helps to support security and fairness in blockchain, benefiting all the users. We are aware that some individuals or entities have been exploiting MEV for financial gain, and our work may disrupt their activities. However, since MEV is commonly regarded as malicious and profiting from it is unethical, and that some MEV such as front-running caused direct harm to users, we believe that our work is justified and ethical.

With detailed security proof and discussion on potential attacks and mitigations, we ensure that our work is robust and secure, and that our work will not introduce new vulnerabilities to the blockchain systems. While transactions in the mempool are encrypted, we strengthen the privacy of users. Given the fact that transactions written to the blockchain will eventually be public, we preserve the transparency of blockchain, and do not introduce new potential financial crimes. Our research methodology does not use any personal data or proprietary information, and our implementation is completely open-source. All the drawbacks and potential risks of related works introduced in our work have been discussed or disclosed in literature, thus we do not introduce new risks to the community.

10 Open Science.

Since all our constructions are well described, and our code as well as our raw evaluation results are open-source and persistent[15], we believe that our work is reproducible and verifiable, following the principles of open science. We will continue to evaluate the ethical implications of our research and commit to addressing any unforeseen negative outcomes and to taking responsibility for the long-term impact of our work.

In conclusion, we believe that our ethical considerations, coupled with our commitment to open science, exemplify sound and responsible research practice.

References

[1] Shutter network. `https://shutter.network`, 2021.

[2] Richard Arratia, Larry Goldstein, and Louis Gordon. Two moments suffice for poisson approximations: the chen-stein method. *The Annals of Probability*, pages 9–25, 1989.

[3] Michael Bailey, David Dittrich, Erin Kenneally, and Doug Maughan. The menlo report. *IEEE Security & Privacy*, 10(2):71–75, 2012.

[4] Joseph Bebel and Dev Ojha. Ferveo: Threshold decryption for mempool privacy in bft networks. *Cryptology ePrint Archive*, 2022.

[5] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, 2014.

[6] Iddo Bentov, Yan Ji, Fan Zhang, Lorenz Breidenbach, Philip Daian, and Ari Juels. Tesseract: Real-time cryptocurrency exchange using trusted hardware. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1521–1538, 2019.

[7] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788, 2018.

[8] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT 2005*, 2005.

[9] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO 2005*, pages 258–275. Springer, 2005.

[15] `https://zenodo.org/records/14672008`

[10] Dan Boneh, Aditi Partap, and Lior Rotem. Accountability for misbehavior in threshold decryption via threshold traitor tracing. In *CRYPTO 2024*, 2024.

[11] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *Advances in Cryptology - ASIACRYPT 2013*, 2013.

[12] Jan Bormet, Sebastian Faust, Hussien Othman, and Ziyan Qu. BEAT-MEV: Epochless approach to batched threshold encryption for MEV prevention. Cryptology ePrint Archive, Paper 2024/1533, 2024.

[13] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In *Theory of Cryptography - 17th International Conference, TCC 2019*, 2019.

[14] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic prfs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015*, 2015.

[15] Jeffrey Burdges and Luca De Feo. Delay encryption. In *Advances in Cryptology - EUROCRYPT 2021*, 2021.

[16] Ran Canetti, Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In *CRYPTO'99*, pages 98–116, 1999.

[17] Arka Rai Choudhuri, Sanjam Garg, Julien Piet, and Guru-Vamsi Policharla. Mempool privacy via batched threshold encryption: Attacks and defenses. In *33rd USENIX Security Symposium, USENIX Security 2024*, 2024.

[18] Dan Cline, Thaddeus Dryja, and Neha Narula. Clockwork: an exchange protocol for proofs of non front-running. The Stanford Blockchain Conference, 2020.

[19] 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 (SP)*, pages 910–927, 2020.

[20] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5359–5376, 2023.

[21] DEDIS. kyber: Advanced Crypto Library for Go. https://github.com/dedis/kyber, 2023.

[22] Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wohnig. Mcfly: Verifiable encryption to the future made practical. In *Financial Cryptography and Data Security - 27th International Conference, FC 2023*, 2023.

[23] Jesko Dujmovic, Rachit Garg, and Giulio Malavolta. Time-lock puzzles with efficient batch solving. In *EUROCRYPT 2024*, 2024.

[24] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: front-running attacks on blockchain. In *Financial Cryptography and Data Security: FC 2019 International Workshops, VOTING and WTSC*, pages 170–189, 2020.

[25] Flashbots. Mev-boost. https://github.com/flashbots/mev-boost, 2022.

[26] Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind schnorr signatures and signed elgamal encryption in the algebraic group model. In *Advances in Cryptology - EUROCRYPT 2020*, pages 63–95, 2020.

[27] Taher El Gamal. On computing logarithms over finite fields. In *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, volume 218, pages 396–402, 1985.

[28] Rachit Garg, George Lu, Brent Waters, and David J. Wu. Realizing flexible broadcast encryption: How to broadcast to a public-key directory. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023*, 2023.

[29] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT'99*, pages 295–310, 1999.

[30] 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*, pages 3–16, 2016.

[31] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In *Advances in Cryptology – CRYPTO '84*, 1984.

[32] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. In *PKC 2022*, 2022.

[33] Philip Hall. On representatives of subsets. *Classic Papers in Combinatorics*, pages 58–62, 1987.

[34] Feng Hao. Schnorr non-interactive zero-knowledge proof. Technical report, 2017.

[35] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology - ASIACRYPT 2010*, 2010.

[36] Alireza Kavousi, Duc V Le, Philipp Jovanovic, and George Danezis. Blindperm: Efficient mev mitigation with an encrypted mempool and permutation. *Cryptology ePrint Archive*, 2023.

[37] Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. Themis: Fast, strong order-fairness in byzantine consensus. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 475–489, 2023.

[38] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *Advances in Cryptology–CRYPTO 2020*, 2020.

[39] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13*, 2013.

[40] 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*, pages 25–36, 2020.

[41] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. Homomorphic time-lock puzzles and applications. In *Advances in Cryptology - CRYPTO 2019*, 2019.

[42] Dahlia Malkhi and Pawel Szalachowski. Maximal extractable value (mev) protection on a dag. *arXiv preprint arXiv:2208.00940*, 2022.

[43] Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd, and Carlos Rozas. Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave. In *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, pages 1–9. 2016.

[44] Peyman Momeni, Sergey Gorbunov, and Bohan Zhang. Fairblock: Preventing blockchain front-running with minimal overheads. In *International Conference on Security and Privacy in Communication Systems*, pages 250–271, 2022.

[45] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99*, 1999.

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

[47] Hany Ragab, Alyssa Milburn, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Crosstalk: Speculative data leaks across cores are real. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1852–1867, 2021.

[48] Ronald L. Rivest, Adi Shamir, and David Wagner. Time-lock puzzles and timed-release crypto. 1996.

[49] Antoine Rondelet and Quintus Kilbourn. Mempool privacy: An economic perspective, 2023.

[50] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology - CRYPTO '89*, pages 239–252, 1989.

[51] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *J. Cryptol.*, 2002.

[52] Shravan Srinivasan, Julian Loss, Giulio Malavolta, Kartik Nayak, Charalampos Papamanthou, and Sri Aravinda Krishnan Thyagarajan. Transparent batchable time-lock puzzles and applications to byzantine consensus. In *Public-Key Cryptography - PKC 2023 - 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7-10, 2023, Proceedings, Part I*, pages 554–584, 2023.

[53] Shravan Srinivasan, Julian Loss, Giulio Malavolta, Kartik Nayak, Charalampos Papamanthou, and Sri AravindaKrishnan Thyagarajan. Transparent batchable time-lock puzzles and applications to byzantine consensus. In *IACR International Conference on Public-Key Cryptography*, 2023.

[54] Chrysoula Stathakopoulou, Signe Rüsch, Marcus Brandenburger, and Marko Vukolić. Adding fairness to order: Preventing front-running attacks in bft protocols using tees. In *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*, pages 34–45, 2021.

[55] James Stearn. Cryptographic approaches to complete mempool privacy. 2022.

[56] 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 (USENIX Security 21)*, pages 1343–1359, 2021.

[57] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein,

Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient {Out-of-Order} execution. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 991–1008, 2018.

[58] Haoqian Zhang, Louis-Henri Merino, Ziyan Qu, Mahsa Bastankhah, Vero Estrada-Galiñanes, and Bryan Ford. F3b: A low-overhead blockchain architecture with per-transaction front-running protection. In *5th Conference on Advances in Financial Technologies (AFT 2023)*, 2023.

A Key-Homomorphic Puncturable PRF

Definition A.1 (Pseudorandomness of PRF). A puncturable PRF is pseudorandom if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\Pr[\mathsf{Game\text{-}PR}_{\mathcal{A}}^{\mathsf{PRF}}(1^\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $\mathsf{Game\text{-}PR}_{\mathcal{A}}$ is defined in Figure 6.

```
$\mathsf{Game\text{-}PR}_{\mathcal{A}}(1^\lambda)$
$(n, \mathsf{st}_1) \xleftarrow{\$} \mathcal{A}_1(\lambda)$
$\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, n)$
$(i^*, \mathsf{st}_2) \xleftarrow{\$} \mathcal{A}_2(\mathsf{st}_1, \mathsf{pp})$
$k \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{pp}); k^* \leftarrow \mathsf{Puncture}(\mathsf{pp}, k, i^*); b \xleftarrow{\$} \{0,1\}$
**if** $b = 0$ **then** $y \xleftarrow{\$} \mathcal{Y}$ **else** $y \leftarrow \mathsf{Eval}(\mathsf{pp}, k, i^*)$
$b' \xleftarrow{\$} \mathcal{A}_3(\mathsf{st}_2, y)$
**return** $b \overset{?}{=} b'$
```

Figure 6: Pseudorandomness game of PRF.

Key-Homomorphic Punctured PRF Construction.

- $\mathsf{Setup}(1^\lambda, 1^n)$:
  - Generate a pairing group
    $$\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, g_T, e) \xleftarrow{\$} \mathsf{GroupGen}(1^\lambda).$$
  - Sample $x_i \xleftarrow{\$} \mathbb{Z}_p^*$ for $i \in [n]$.
  - Sample $z_i \xleftarrow{\$} \mathbb{Z}_p^*$ for $i \in [n]$.
  - Output $\mathsf{pp} = (\mathcal{G}, \{{g_1}^{z_i/x_j}\}_{i,j \in [n] \text{ s.t. } i \neq j}, \{{g_1}^{z_i}\}_{i \in [n]}, \{{g_2}^{x_i}\}_{i \in [n]})$.

  Note that this setup is adapted from the key-homomorphic punctured PRF of [23], where we additionally publish $\{{g_1}^{z_i}\}_{i \in [n]}$.
- $\mathsf{KeyGen}(\mathsf{pp})$: Sample $k \xleftarrow{\$} \mathbb{Z}_p^*$ and return k.
- $\mathsf{Puncture}(\mathsf{pp}, k, i^*)$: Return $k^* \leftarrow (g_2^{x_{i^*}})^k = g_2^{x_{i^*}k}$.
- $\mathsf{Eval}(\mathsf{pp}, k, i)$:
  - Choose an index $j \in [n]$ with $j \neq i$.
  - Return $y \leftarrow e(g_1^{z_i/x_j}, g_2^{x_j})^k$.

  Note: This yields $y = e(g_1, g_2)^{z_i k} = g_T^{z_i k}$, which means one can evaluate the PRF without computing a pairing, if one precomputes and saves $g_T^{z_i} = e(g_1^{z_i}, g_2)$ using the public setup.
- $\mathsf{ExpEval}(\mathsf{pp}, K, i)$: Return $y \leftarrow e(g_1^{z_i}, K)$. For $K = g_2^k$, this yields $y = g_T^{z_i k}$.
- $\mathsf{PEval}(\mathsf{pp}, k^*, i^*, i)$:
  - If $i = i^*$, return $\perp$.
  - Otherwise, compute $y \leftarrow e(g_1^{z_i/x_{i^*}}, k^*) = e(g_1^{z_i/x_{i^*}}, g_2^{x_{i^*}k}) = g_T^{z_i k}$.

  Output y.

Correctness and Pseudorandomness. The correctness is straightforward from the description above. The proof of pseudorandomness from [23] carries over with our modification. The only modification we make is including $g_1^{z_i}$ for every i in the setup. This modification does not change the security proof in [23] since one can easily modify their reduction to also publish $g_1^{z_i}$.

Key-Homomorphism with Exponent Evaluation. Observe that given two keys in the exponent $K_1 = g^{k_1}, K_2 = g^{k_2}$, it holds that

$$\begin{aligned}
&\mathsf{ExpEval}(\mathsf{pp}, K_1 \cdot K_2, i) = e(g_1^{z_i}, K_1 \cdot K_2) \\
&= e(g_1^{z_i}, K_1) + e(g_1^{z_i}, K_2) \\
&= \mathsf{ExpEval}(\mathsf{pp}, K_1, i) + \mathsf{ExpEval}(\mathsf{pp}, K_2, i).
\end{aligned}$$

B NIZK Proof System for CCA-security

We construct a proof system Π for the relation $\mathcal{R}$ of statement-witness pairs (χ, ω). The relation $\mathcal{R}$ is defined as the set of all tuples (χ, ω) of statements $\chi = (\mathsf{pp}, \mathsf{pk}, c := (i, y := k^*, \gamma, ct := (A, B)))$ and witnesses $\omega = (k, u)$ such that

$$A = g_2^u \,\wedge\, B = \mathsf{pk}^u g_2^k \,\wedge\, y = g_2^{x_{i^*}k}.$$

We also define the corresponding language $\mathcal{L} = \{\chi \mid \exists \omega \colon (\chi, \omega) \in \mathcal{R}\}$.

The construction is laid out in Figure 7.

We move on to establish that our protocol achieves the standard NIZK-proof properties of completeness, soundness, zero-knowledge. For our $\mathsf{B\text{-}IND\text{-}CCA}$ proof of BTE, we also require the property of simulation-extractability.

Definition B.1 (Completeness). The proof system Π is complete if for all $(\chi, \omega) \in \mathcal{R}$ it holds that $\Pr[\pi \xleftarrow{\$} \mathsf{Prove}(\chi, \omega) : \mathsf{Verify}(\chi, \pi) = 1] = 1$.

```
$\mathsf{Prove}(\chi,\omega)$
    // Parse $\chi := (\mathsf{pp},\mathsf{pk},(i,k^*,\gamma,(A,B)))$
    // Parse $\omega := (k,u)$
$u' \xleftarrow{\$} \mathbb{Z}_p$
$k' \xleftarrow{\$} \mathbb{Z}_p$
$A' \leftarrow {g_2}^{u'}$
$B' \leftarrow \mathsf{pk}^{u'}{g_2}^{k'}$
$y' \leftarrow (g_2^{x_{i^*}})^{k'}$
$f \leftarrow H(\chi,A',B',y')$
$\hat{u} \leftarrow u' + fu$
$\hat{k} \leftarrow k' + fk$
output $\pi = (A',B',y',\hat{u},\hat{k})$
```

```
$\mathsf{Verify}(\chi,\pi)$
    // Parse $\chi := (\mathsf{pp},\mathsf{pk},(i,k^*,\gamma,(A,B)))$
    // Parse $\pi := (A',B',y',\hat{u},\hat{k})$
$f \leftarrow H(\chi,A',B',y')$
Check:
${g_2}^{\hat{u}} \overset{?}{=} A' \cdot A^f$
$\mathsf{pk}^{\hat{u}}{g_2}^{\hat{k}} \overset{?}{=} B' \cdot B^f$
$(g_2^{x_{i^*}})^{\hat{k}} \overset{?}{=} y' \cdot y^f$
output 1 if all checks pass,
else output 0.
```

Figure 7: Proof system Π for our Batched Threshold Encryption scheme.

To prove completeness, we show that the verifier Verify accepts the proof π generated by the prover Prove for any valid statement-witness pair $(\chi,\omega) \in \mathcal{R}$. Let $(\chi,\omega) \in \mathcal{R}$, then it holds that

$$A' \cdot A^f = {g_2}^{u'} \cdot {g_2}^{fu} = {g_2}^{u'+fu} = {g_2}^{\hat{u}},$$

$$B' \cdot B^f = \mathsf{pk}^{u'}{g_2}^{k'} \cdot \mathsf{pk}^{fu}{g_2}^{fk} = \mathsf{pk}^{u'+fu}{g_2}^{k'+fk} = \mathsf{pk}^{\hat{u}}{g_2}^{\hat{k}},$$

and

$$y' \cdot y^f = (g_2^{x_{i^*}})^{k'} \cdot (g_2^{x_{i^*}})^{fk} = g_2^{x_{i^*}(k'+fk)} = (g_2^{x_{i^*}})^{\hat{k}}.$$

Definition B.2 (Soundness). The proof system Π is sound if for all $\chi \notin \mathcal{L}$ and all adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that $\Pr[\pi \xleftarrow{\$} \mathcal{A}(\chi) : \mathsf{Verify}(\chi,\pi) = 1] \leq \mathsf{negl}(\lambda)$.

Soundness guarantees that no adversary can forge a proof of a false statement. In the following, we give a proof sketch of the soundness of the proof system Π.

Without loss of generality, we assume $\chi = (\mathsf{pp},\mathsf{pk},c)$, where $c = (i,k_0^*,\gamma,ct)$ and $ct = (A,B) = ({g_2}^{u_0},\mathsf{pk}^{u_1}{g_2}^{k_1})$, and $y = k_0^* = g_2^{x_{i^*}k_0}$. Further, let $\pi = (A',B',y',\hat{u},\hat{k})$, where $A',B',y' \in \mathbb{G}_2$ and $\hat{u},\hat{k} \in \mathbb{Z}_p$ such that $\mathsf{Verify}(\chi,\pi) = 1$. We can assume $A' = {g_2}^{u'_0}$, $B' = \mathsf{pk}^{u'_1}{g_2}^{k'_1}$, and $y' = g_2^{x_{i^*}k'_0}$ for some $k_0,k_1,u_0,u_1,k'_0,k'_1,u'_0,u'_1 \in \mathbb{Z}_p$.

Since the Verify algorithm accepts the proof, we have:

- ${g_2}^{\hat{u}} = A' \cdot A^f = {g_2}^{u'_0} \cdot {g_2}^{fu_0} = {g_2}^{u'_0+fu_0}$, thus $\hat{u} = u'_0 + fu_0$.
- $g_2^{x_{i^*}\hat{k}} = y' \cdot y^f = g_2^{x_{i^*}k'_0} \cdot g_2^{x_{i^*}fk_0} = g_2^{x_{i^*}(k'_0+fk_0)}$, thus $\hat{k} = k'_0 + fk_0$.
- $\mathsf{pk}^{\hat{u}}{g_2}^{\hat{k}} = B' \cdot B^f = \mathsf{pk}^{u'_1}{g_2}^{k'_1} \cdot \mathsf{pk}^{fu_1}{g_2}^{fk_1} = \mathsf{pk}^{u'_1+fu_1}{g_2}^{k'_1+fk_1}$.

Since we know $\hat{u} = u'_0 + fu_0$ and $\hat{k} = k'_0 + fk_0$, we can rewrite the last equation as $\mathsf{pk}^{u'_0+fu_0}{g_2}^{k'_0+fk_0} = \mathsf{pk}^{u'_1+fu_1}{g_2}^{k'_1+fk_1}$. So we have $\mathsf{pk}^{(u'_0-u'_1)+f(u_0-u_1)}{g_2}^{(k'_0-k'_1)+f(k_0-k_1)} = 1$. This equation holds if $(u'_0-u'_1)+f(u_0-u_1) = 0$ and $(k'_0-k'_1)+f(k_0-k_1) = 0$. Since f is derived from a random oracle, the probability of finding f such that the above equations hold is negligible. Thus, we have $u'_0 = u'_1$, $u_0 = u_1$, $k'_0 = k'_1$, and $k_0 = k_1$.

Combining above equations, there must exist a witness $\omega = (k,u)$ where $k = k_0 = k_1$ and $u = u_0 = u_1$ such that $(\chi,\omega) \in \mathcal{R}$. Hence the proof system Π is sound.

Definition B.3 (Zero-Knowledge). There exists a PPT simulator Sim such that for all $(\chi,\omega) \in \mathcal{R}$ it holds that $(\chi,\mathsf{Sim}(\chi)) \approx_c (\chi,\mathsf{Prove}(\chi,\omega))$.

Zero-knowledge means that the verifier Verify learns nothing from the proof π except the validity of the statement χ. We show that Π is zero-knowledge in the Random Oracle Model (ROM) by constructing a simulator Sim that can simulate a valid proof π without knowing the witness ω. The simulator can program the random oracle, and an adversary who can query the oracle will get responses chosen by the simulator.

The simulator Sim receives the statement $\chi = (\mathsf{pp},\mathsf{pk},c)$ as input, with $c = (i,k^*,\gamma,ct)$ and $ct = (A,B) = ({g_2}^u,\mathsf{pk}^u{g_2}^k)$, and generates a simulated proof $\pi = (A',B',y',\hat{u},\hat{k})$ as follows:

- Sample $\hat{u},\hat{k},f \xleftarrow{\$} \mathbb{Z}_p$.
- Compute $A' = {g_2}^{\hat{u}}A^{-f}$, $B' = \mathsf{pk}^{\hat{u}}{g_2}^{\hat{k}}B^{-f}$, and $y' = (g_2^{x_{i^*}})^{\hat{k}}y^{-f}$.
- Program H so that $H(\chi,A',B',y') = f$.
- Output $\pi = (A',B',y',\hat{u},\hat{k})$.

Since $\hat{u},\hat{k},f$ are chosen uniformly at random from $\mathbb{Z}_p$, the simulated proof π is distributed identically and indistinguishable to the real proof.

Definition B.4 (Simulation-Extractability). The proof system Π is simulation-extractable if there exists a PPT extractor $\mathsf{Extract}$, such that for any PPT adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that $Pr[(\chi,\pi) \leftarrow \mathcal{A}^{\mathsf{SimProve}} \wedge \omega \leftarrow \mathsf{Extract}(\chi,\pi,\mathcal{Q}) : \mathsf{Verify}(\chi,\pi) = 1 \wedge (\chi,\omega) \notin \mathcal{R} \wedge (\chi,\omega) \notin \mathcal{Q}] \leq \mathsf{negl}(\lambda)$.

Here, $\mathsf{SimProve}$ returns the simulated proof $\pi = \mathsf{Sim}(\chi)$ for the given statement χ. $\mathcal{Q}$ is the set of queries made by $\mathcal{A}$ to $\mathsf{SimProve}$.

Here, we also require the extractor to be online (or straight-line), which means that the extractor does not rewind the adversary. To do this, we prove the simulation-extractability of the proof system Π in the Algebraic Group Model (AGM), following the techniques used in [26]. The proof can be found in the full version of this work [12].

C Removing Coordination

Recall from Section 5.3 that removing coordination in the BTE scheme does not compromise correctness or security. This is because, when we have $\alpha = \sqrt{B}, N = kB$, under the condition $\lambda_s < \sqrt{B}(\log k\sqrt{B} - 2)$, where λ_s is the statistical correctness parameter, we have negligible probability of index collision $P_B^{(\sqrt{B}+1)} < 2^{-\lambda_s}$. What's more, even when index collision really happens, we can always downgrade to have more sub-batches to accommodate the situation. This means that in practice, we could always set α dynamically and achieve perfect correctness at the cost of a bit more communication overhead. We define $P_B^{(\alpha+1)}$ as follows, and provide the complete proof here.

Given B parties, there are $\binom{B}{\alpha+1}$ ways to choose a set of $\alpha+1$ parties. The probability that a fixed set of $\alpha+1$ parties sample the same index from a domain of size N is $N \cdot 1/N^{\alpha+1} = 1/N^{\alpha}$. Since the indices are sampled randomly and independently, the probability that they do not sample the same index is $1 - 1/N^{\alpha}$. Thus, the probability that no set of $\alpha+1$ parties all sample the same index is $(1-1/N^{\alpha})^{\binom{B}{\alpha+1}}$. The probability that there exist a set of $\alpha+1$ parties with the same index is then $P_B^{(\alpha+1)} = 1 - (1-1/N^{\alpha})^{\binom{B}{\alpha+1}}$. This probability $P_B^{(\alpha+1)}$ is then the probability of having index collision in a batch of B parties using our technique. Given a statistical correctness parameter λ_s, we want $P_B^{(\alpha+1)} < 2^{-\lambda_s}$.

Given the relation below, we can simplify the above expression:

$$\binom{B}{\alpha+1} \leq B^{\alpha+1}/(\alpha+1)! \tag{2}$$

$$1 - 1/N^{\alpha} \approx e^{-1/N^{\alpha}} \text{ for small } 1/N^{\alpha} \tag{3}$$

With reasonable B and N, i.e. $B \geq 16, N \geq B$, the term on the exponent is dominating, and we could bound the probability by:

$$P_B^{(\alpha+1)} \leq 1 - e^{-B^{\alpha+1}/(\alpha+1)!N^{\alpha}} \leq \frac{B^{\alpha+1}}{(\alpha+1)!N^{\alpha}} \tag{4}$$

According to Stirling's approximation, $(\alpha+1)! \geq \sqrt{2\pi}\sqrt{\alpha+1}(\frac{\alpha+1}{e})^{\alpha+1}$. Taking efficiency optimization from 5.2 into consideration, it is natural that we set $\alpha = \sqrt{B}$ (assuming $\sqrt{B}$ is an integer). If we also model the relationship between n and N as $N = kB$, where k is some constant, we can further simplify the expression:

$$P_B^{(\sqrt{B}+1)} \leq \frac{B}{(\sqrt{B}+1)!k^{\sqrt{B}}} \tag{5}$$

$$P_B^{(\sqrt{B}+1)} \leq \frac{B}{\sqrt{2\pi}\sqrt{\sqrt{B}+1}(\frac{\sqrt{B}+1}{e})^{\sqrt{B}+1}k^{\sqrt{B}}} \tag{6}$$

$$\leq \frac{e^{\sqrt{B}+1}B}{\sqrt{2\pi}(\sqrt{B}+1)^{\sqrt{B}+1}k^{\sqrt{B}}} \tag{7}$$

Given the statistical correctness parameter λ_s, we would want to have $P_B^{(\sqrt{B}+1)} < 2^{-\lambda_s}$. It holds that

$$P_B^{(\sqrt{B}+1)} \leq \frac{e^{\sqrt{B}+1}B}{\sqrt{2\pi}(\sqrt{B}+1)^{\sqrt{B}+1}k^{\sqrt{B}}} \tag{8}$$

$$\leq \frac{e^2}{\sqrt{2\pi}\cdot k}\left(\frac{e}{k\sqrt{B}}\right)^{\sqrt{B}-1}. \tag{9}$$

Therefore, loosely, we get the required probability when $\lambda_s < \sqrt{B}(\log k\sqrt{B} - 2)$. For practical choices of B, we can have larger values of λ_s when we do tighter calculations.