

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

Jan Bormet
TU Darmstadt
jan.bormet@tu-darmstadt.de

Sebastian Faust
TU Darmstadt
sebastian.faust@tu-darmstadt.de

Hussien Othman
TU Darmstadt
hussien.othman@gmail.com

Ziyan Qu
TU Darmstadt
ziyan.qu@tu-darmstadt.de

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 the problem 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 allow 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 applications¹.

Similar to traditional financial markets, the DeFi ecosystem is prone to market manipulation. Daian et al. [15] 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 [20, 27, 51]. 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 [3, 13, 33]. 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 [3, 13, 53]. 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 important shortcoming. The communication to decrypt a batch of encrypted transactions B among a committee of n servers is $O(n|B|)$ instead of $O(|B|)$ without privacy. As outlined in [13], 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 application 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 address it via releasing so-called *epoch keys*. Here, the committee members locally derive epoch key shares. When $O(n)$ such shares get published all shielded transactions that were sent by users during this epoch can be decrypted.

Recently, Choudhuri et al. [13] observe an important 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 an important 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. [13] 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. [13] offers an elegant solution for pending transaction privacy. It suffers, however, from the following important 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 [13]. 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 in our contribution 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) 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. [13] 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 [32]. 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 $p(x) = y$. At a high level, in [13] 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) to go through all (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 running for each batch to be decrypted an expensive interactive epoch setup. 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 [13] 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 [13], 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 [13] 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 [19].

- 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. [13]. 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 [13] by avoiding pairings in both operations.

We remark that in contrast to [13], our scheme offers a less efficient reconstruction process to decrypt a batch *after* releasing the decryption shares. The construction of [13] 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

We focus here on the most important related work that was not already discussed before. A more comprehensive overview is given in Appendix A. One of the mainstream solutions to mitigate MEV is to make the mempool private by shielding transactions. Three different approaches have been considered to achieve this goal: threshold encryption, time-lock encryption, and TEE-based schemes.

Several works [3,33,39,43,53] 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 [41] use threshold identity-based encryption to encrypt transactions to an epoch. As already discussed, these works greatly improve communication complexity but do not achieve pending transaction privacy. Similarly, a recent work of Döttling et al. [18] introduces a scheme for encryption to the future with constant communication complexity. It relies on a signature-based witness encryption scheme, where batches of ciphertexts can be decrypted when a signature for

³<https://anonymous.4open.science/r/btd-impl>

a particular epoch becomes available. Since the witness allows to decrypt all ciphertext of an epoch, pending transaction privacy is not guaranteed.

Time-lock encryption [5] guarantees that a message stays hidden until a pre-defined time, which has been discussed as a solution to provide mempool privacy [49]. 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. In addition, prior work [14] requires investing computation for each encrypted message resulting into huge computationally overheads. Recently, Dujmovic et al. [19] 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 function 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) [40]. TEEs provide secure and private hardware isolation, where data and code is executed in a protected environment. TEE has been used as a solution to MEV [4, 50], but relies on a strong trust assumption (see, e.g., attacks in [44, 52]).

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 [9, 11, 19, 38, 48], which allow amortized solving of Time-Lock Puzzles (TLP) [45]. In particular, our work will follow the framework of Dujmovic, Garg, and Malavolta [19].

Following [19, 48], we base our construction on a primitive called Puncturable Pseudorandom Function (PRF) [8, 36]. A punctured PRF is a regular PRF [28] that is punctured at some point i^* . That is, for a PRF key k , we publish a punctured key k^* such that the value $\text{PRF}(k, i)$ for every $i \neq i^*$ can be computed using the punctured key k^* , while the evaluation at the punctured point, i.e., $\text{PRF}(k, i^*)$, can be computed only using the key k (and should look random even given k^*). In our work, we consider a Key-Homomorphic PRF [10]. In Key-Homomorphic PRFs, for any two keys k_1, k_2 , it holds that $\text{PRF}(k_1 + k_2, i) = \text{PRF}(k_1, i) + \text{PRF}(k_2, i)$. In our construction, we use the Pairing-based Key-Homomorphic PRF from [19], 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 ExpEval such that for any input

i : $\text{PRF}(k, i) = \text{ExpEval}(g^k, i)$. Second, we work in asymmetric pairing groups (see Section 4 for more details) to combine it with a suitable threshold encryption scheme, namely some form of Threshold ElGamal Encryption in the Exponent [24] (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 $\gamma \leftarrow m + \text{PRF}(k, i)$. Observe that by the security of the PRF, the value $\text{PRF}(k, i)$ 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:

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

To see why this is correct, observe that by the puncturing property of the PRF scheme, the value $\text{PRF}(k_j^*, i)$ can be computed for any $i \neq j$ and it is equal to $\text{PRF}(k_j, i)$ (i.e., the evaluation with the secret key k_j).⁴ Also, from the key-homomorphism and evaluation in the exponent of the underlying PRF scheme, it holds that:

$$\text{ExpEval}(K, i) = \text{PRF}(k_1 + \dots + k_B, i) = \sum_j^{[B]} \text{PRF}(k_j, i)$$

and therefore,

$$\sum_{j \neq i}^{[B]} \text{PRF}(k_j^*, i) - \sum_j^{[B]} \text{PRF}(k_j, i) = -\text{PRF}(k_i, i)$$

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.

⁴We abuse notation here and use the same algorithm for the evaluation of the PRF and the punctured evaluation.

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 [19, 25], they use a technique related to finding a perfect matching in bipartite graphs [30]. 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 [30], they show that with overwhelming probability there will be a matching in which each ciphertext has a unique index assuming appropriate parameters d and domain size. We note that this approach can also be used in our construction. This solution however increase the number of ciphertexts 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 $O(B\sqrt{B})$. Thus, we benefit from small ciphertext size and more efficient computation at the cost of 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 [13]:

- *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 [19], we define *Rouge Ciphertext Security* (see Definition 3.3) 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 an tailored NIZK proof system that is based on the well known Schnorr proof of knowledge construction [46]. 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 runtime blows up exponentially (see [47] for more details). In order to overcome this, similar to [13], we can use techniques from [23] 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 $y \stackrel{\$}{\leftarrow} A(x)$ to indicate execution of A with uniform randomness. By $[n]$ for a positive integer n , we denote the set of integers $\{1, \dots, n\}$. 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 $K = g^k$).

From a high level, such a PRF is a tuple of algorithms $\text{PRF} = (\text{Setup}, \text{KeyGen}, \text{Puncture}, \text{Eval}, \text{PEval}, \text{ExpEval})$. $\text{Setup}(1^\lambda, n)$ generates public parameters. Here n is the domain size, which is the number of indices that the PRF can be evaluated on. KeyGen generates a key k , $\text{Puncture}(k, i)$ generates a punctured key k^* for index i and $\text{Eval}(k, i)$ evaluates the PRF under key k on index i . For punctured evaluation, we have $\text{PEval}(k^*, i^*, i)$, which evaluates the PRF under punctured key k^* on index i as long as i is not the punctured index $i \neq i^*$. For exponent evaluation we have $\text{ExpEval}(g^k, i)$ evaluating the PRF on i using the key in the exponent. A more formal definition is contained within the appendix (Definition B.1).

We extend the definition of *functionality preserving* from [19] to "perfect correctness", requiring that exponent evaluation on $K = g^k$ yields the same result as normal evaluation on k in addition to correct evaluation with punctured keys.

The pseudorandomness and key-homomorphism properties are defined as in [19]. In short, we require that $\text{Eval}(\text{pp}, k, i)$ looks pseudorandom as long as k is not revealed, even given a punctured key k^* on index i . We refer to a formal definition of pseudorandomness to the appendix (Definition B.2).

For key-homomorphism we require that, for any $k_1, k_2 \in \mathcal{K}$, it holds that $\text{Eval}(\text{pp}, k_1 + k_2, i) = \text{Eval}(\text{pp}, k_1, i) + \text{Eval}(\text{pp}, k_2, i)$.

Threshold Homomorphic Encryption. A threshold homomorphic encryption protocol THE is a tuple of PPT algorithms $\text{THE} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec}, \text{Combine})$. KeyGen generates a public key pk and n secret key shares $\{\text{sk}_\ell\}_{\ell \in [n]}$ distributed to decryption servers. Given a ciphertext $c \stackrel{\$}{\leftarrow} \text{Enc}(\text{pk}, m)$, any server can derive a decryption share $d_\ell \stackrel{\$}{\leftarrow} \text{Dec}(\text{sk}_\ell, c)$. If a set S of at least t servers provide their decryption shares, the message m can be recovered using $m \leftarrow \text{Combine}(\text{pk}, \{d_\ell\}_S, S, c)$. We require our THE scheme to be correct, multiplicatively message-homomorphic and IND-CPA secure. For formal definitions we refer to the Appendix (Definitions B.3, B.4, B.5 and B.6).

3.1 Batched Threshold Encryption

We introduce the definition of a Batched Threshold Encryption (BTE) scheme following the work of Choudhuro et al. [13] and Dujmovic et al. [19] with some minor modification.

Definition 3.1 (Batched Threshold Encryption). A *Batched Threshold Encryption* scheme (BTE) consists of a tuple of PPT algorithms $\text{BTE} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Verify}, \text{BatchDec}, \text{Combine})$ with the following syntax.

- $\text{pp} \stackrel{\$}{\leftarrow} \text{Setup}(1^\lambda, B_{\max})$: 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.
- $(\text{pk}, \{\text{sk}_\ell\}_{\ell \in [n]}) \stackrel{\$}{\leftarrow} \text{KeyGen}(1^\lambda, n, t)$: 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 $\{\text{sk}_i\}_{i \in [n]}$.
- $(c, \pi) \stackrel{\$}{\leftarrow} \text{Enc}(\text{pk}, m, i)$: 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 .
- $\{1, 0\} \leftarrow \text{Verify}(\text{pk}, c, \pi)$. 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.
- $d_\ell / \perp \leftarrow \text{BatchDec}(\text{sk}_\ell, \{c_i\}_{i \in B})$: 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 \perp .
- $\{m_i / \perp\}_{i \in [B]} \leftarrow \text{Combine}(\text{pk}, \{d_\ell\}_{\ell \in S}, S, \{c_i\}_{i \in [B]})$: 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 \perp .

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.

Correctness. We require that a BTE scheme satisfies perfect correctness. This means that the proof generated during honest encryption always verifies and that every honest decryption and combination of an honestly generated batch of ciphertexts yields the correct messages. A formal definition is given in Appendix (Definition B.7).

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 $\{sk_\ell\}_{\ell \in C}$. After proposing two messages m_0 and m_1 , he receives a challenge ciphertext c^* which is an encryption of one of the messages. The adversary wins the game by guessing correctly, whether the challenge encrypts 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 his choice. The only restriction is that the adversary cannot query for too many decryption shares of any batch containing the challenge ciphertext.⁵ 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 3.2 (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 $\text{negl}(\lambda)$ such that $\Pr[\text{Game-B-IND-CCA}_{\mathcal{A}}^{\text{BTE}}(1^\lambda) = 1] \leq 1/2 + \text{negl}(\lambda)$ where $\text{Game-B-IND-CCA}_{\mathcal{A}}$ is defined in Figure 1.

Rogue Ciphertext Security. Dujmovic et al. [19] 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.

⁵If the adversary learns $t - |C|$ decryption shares, it can win the game trivially by combining with the $|C|$ shares of the corrupted parties

Definition 3.3 (Rogue Ciphertext Security 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 $\text{negl}(\lambda)$ such that $\Pr[\text{Game-Rogue}_{\mathcal{A}}^{\text{BTE}}(1^\lambda) = 1] \leq \text{negl}(\lambda)$ where $\text{Game-Rogue}_{\mathcal{A}}$ is defined in Figure 1.

4 Building Blocks

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

Key-homomorphic Puncturable PRFs. For our building block of key-homomorphic puncturable PRF with exponent evaluation, we adapt the construction in [19] to our needs. The full construction is given in Appendix B.1, 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 [19] by publishing more elements in the setup. Furthermore, the PRF construction of [19] 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.⁶ 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 [19] 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 [7], which is less standard.⁷ We expect that our constructions can be easily adapted to the second construction as well.

It is important to highlight that both constructions from [19] 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 [13]. However, we present several solutions to remove this coordination assumption in Section 5.3.

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 [24]. Our version of threshold ElGamal encryption works by Shamir-sharing the ElGamal secret key sk into

⁶Since DDH is not a hard problem with symmetric pairings $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, we will use asymmetric pairings instead and require that DDH is hard in one of the source groups (here written as \mathbb{G}_2).

⁷The assumption has been proven to hold in the bilinear generic group model by [6].

Game-B-IND-CCA $_{\mathcal{A}}(1^\lambda)$	Oracle $O^{\text{b-dec}}(\ell, \{(c_i, \pi_i)\}_{i \in [B]})$	Game-Rogue $_{\mathcal{A}}(\lambda)$
$c^* \leftarrow \perp; ctr \leftarrow 0; b \xleftarrow{\$} \{0, 1\}$ $pp \xleftarrow{\$} \text{Setup}(1^\lambda, B_{\max})$ $(pk, \{sk_\ell\}_{[n]}) \xleftarrow{\$} \text{KeyGen}(1^\lambda, n, t)$ $(C, st_1) \xleftarrow{\$} \mathcal{A}_1(1^\lambda, pp, pk, n, t)$ if $C \not\subseteq [n] \vee C \geq t$ then return 0 $(m_0, m_1, i, st_2) \xleftarrow{\$} \mathcal{A}_2^{O^{\text{b-dec}}}(st_1, \{sk_\ell\}_C)$ $(c^*, \pi) \leftarrow \text{Enc}(pk, m_b, i)$ $b' \xleftarrow{\$} \mathcal{A}_3^{O^{\text{b-dec}}}(st_2, c^*, \pi)$ return $b \stackrel{?}{=} b'$	if $c^* \in \{c_i\}_{[B]}$ then $ctr \leftarrow ctr + 1$ if $ctr \geq t - C $ return \perp for $i \in [B]$ do if $\text{Verify}((pp, pk, c_i), \pi_i) = 0$ then return \perp return $\text{BatchDec}(sk_\ell, \{c_i\}_{[B]})$	$pp \xleftarrow{\$} \text{Setup}(1^\lambda, B_{\max}); (pk, \{sk_\ell\}_{[n]}) \xleftarrow{\$} \text{KeyGen}(1^\lambda, n, t)$ $(m, i, st) \xleftarrow{\$} \mathcal{A}_1(1^\lambda, pp, B_{\max}, pk, \{sk_\ell\}_{[n]})$ $(c_i, \pi_i) \xleftarrow{\$} \text{Enc}(pk, m, i)$ $(B, S, \{(c_j, \pi_j)\}_{j \in [B] \setminus \{i\}}) \xleftarrow{\$} \mathcal{A}_2(st, c_i)$ 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. $\text{Verify}(pk, c_j, \pi_j) = 0$ then return 0 $\{d_\ell\}_{\ell \in S} \leftarrow \{\text{BatchDec}(sk_\ell, \{c_j\}_{j \in [B]})\}_{\ell \in S}$ $\{m_j\}_{j \in [B]} \xleftarrow{\$} \text{Combine}(pk, \{d_\ell\}_{\ell \in S}, S, \{c_j\}_{j \in [B]})$ if $m_i \neq m$ return 1 else return 0

Figure 1: Security games of BTE.

$\{sk_\ell\}_{[n]}$. We write $pk_\ell = g^{sk_\ell}$ for the parties' individual public keys and set the overall public key to be $pk = (g^{sk}, \{pk_\ell\}_{[n]})$. Given a ciphertext $c = (A, B) = (g^u, pk^u \cdot m)$ we perform a partial decryption by computing $d_\ell \leftarrow (A^{sk_\ell})$. One can then combine a set of at least t partial decryptions to recover the message m using Lagrange interpolation.

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

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 Appendix D.

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 .⁸

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 $\{sk_\ell\}_{\ell \in [n]}$ 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.

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 $\gamma \leftarrow m + \text{PRF.Eval}(k, i)$.⁹ 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

⁸One can extend the message space to arbitrary bitstrings using standard encapsulation techniques.

⁹Note that k^* is not sufficient to evaluate the PRF at index i .

Setup($1^\lambda, B_{\max}$)	Enc(pk, m, i)	Combine(pk, $\{d_\ell\}_{\ell \in S}, S, \{c_i\}_{i \in [B]}$)
<i>I</i> Setup the pairing ensemble	$k \leftarrow \text{PRF.KeyGen}(\text{pp})$	Parse c_i as $(i, k_i^*, \gamma_i, ct_i)$
return pp $\xleftarrow{\$}$ PRF.Setup($1^\lambda, B_{\max}$)	$k^* \leftarrow \text{PRF.Puncture}(k, i)$	$C \leftarrow \prod_i^{[B]} ct_i$
KeyGen($1^\lambda, n, t$)	$\gamma \leftarrow m + \text{PRF.Eval}(k, i)$	$K \leftarrow \text{THE.Combine}(\text{pk}, \{d_\ell\}_{\ell \in S}, S, C)$
$(\text{pk}, \{\text{sk}_i\}_{i \in [n]}) \xleftarrow{\$}$ THE.KeyGen($1^\lambda, n, t$)	$u \xleftarrow{\$} \mathbb{Z}_p$	for $i \in [B]$ do
return (pk, $\{\text{sk}_i\}_{i \in [n]}$)	$ct \leftarrow \text{THE.Enc}(\text{pk}, g_2^k; u)$	$m_i \leftarrow \gamma_i + \sum_{j \neq i}^{[B]} \text{PRF.PEval}(k_j^*, i) - \text{PRF.ExpEval}(d, i)$
BatchDec($\text{sk}_i, \{c_i\}_{i \in [B]}$)	$c \leftarrow (i, k^*, \gamma, ct)$	return $\{m_i\}_{i \in [B]}$
$C \leftarrow \prod_i^{[B]} ct_i$	$\chi \leftarrow (\text{pp}, \text{pk}, c)$	Verify(pk, c, π)
$d_i \leftarrow \text{THE.Dec}(\text{sk}_i, C)$	$\omega \leftarrow (k, u)$	return $\Pi.\text{Verify}(\text{pp}, \text{pk}, c, \pi)$
return d_i	return (c, π)	

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

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

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

First, note that $\gamma_i = m_i + \text{PRF.Eval}(k_i, i)$. Because of the punctureability of the PRF, we get that $\text{PRF.PEval}(k_j^*, i) = \text{PRF.Eval}(k_j, i)$ 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 .¹⁰ In summary, because of the exponent evaluation, we have for the last part of Equation 1 that $\text{PRF.ExpEval}(K, i) = \text{PRF.Eval}(\sum_j^{[B]} k_j, i)$, which is equal to $\sum_j^{[B]} \text{PRF.Eval}(k_j, i)$ due to the additive key-homomorphic property of the PRF. Hence, the PRF evaluations cancel out

¹⁰We 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 $g_2^{\sum_i k_i}$ from the partial decryptions and not $\sum_i k_i$ directly. We avoid using additively homomorphic protocols such as Paillier [42] since they are less efficient to thresholdize.

and we are left with m_i :

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

Non-Interactive Zero-Knowledge Proof. As we discussed our BTE construction uses a non-interactive proof system $\Pi = (\text{Prove}, \text{Verify})$, which we require to achieve IND-CCA security and Rogue Ciphertext Security. The idea is similar to Schnorr non-interactive zero-knowledge proof [31]. 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 $\chi = (\text{pp}, \text{pk}, c)$, where $c = (i, k^*, \gamma, ct)$ and $ct = (A, B) = (g^u, \text{pk}^u g_2^k)$. 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 C 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 Appendix D.1 and D.2, respectively.

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 pseudo-randomness of the PRF and the simulability and simulation-extractability of the proof system Π .*

We use multiple game hops to prove B-IND-CCA security of our BTE construction. The main challenge here is that secrecy of a ciphertext c_i relies on the encryption of the PRF keys k_i . For batching, we *need* this encryption of k_i to be homomorphic (which is why we instantiate with ElGamal), but CCA security and homomorphic encryption seem contradictory at first glance as we need to prove B-IND-CCA security while reducing to IND-CPA security of threshold ElGamal where we do not have access to a decryption oracle.

The key idea is to use simulation-extractability of the proof system Π . Given a batch of ciphertexts $\{(c_i, \pi_i)\}_{i \in [B]}$ in $O^{\text{b-dec}}$ we extract the ElGamal randomness u_i from *each* proof π_i . Using this information, we can simulate the decryption share d_ℓ for the batch as $d_\ell \leftarrow \text{pk}_\ell^{\sum u_i} [= (g_2^{\sum u_i})^{sk_i}]$.

As explored by previous work [13], this requires a straight-line extractor to prevent exponential blow-up of the reduction due to rewinding. It has been shown that Schnorr proofs such as ours have straight-line extractors in the AGM and random oracle model [21, 22]. We refer the reader to the Appendix D.1 for a detailed proof of B-IND-CCA security.

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 ciphertext 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. The formal proof is presented in the Appendix D.2.

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 $\text{PRF.ExpEval}(K, i)$ as well as $B - 1$ pairings for the punctured evaluations of the PRF with the punctured keys k_j^* for $j \neq i$ ($\text{PRF.PEval}(k_j^*, i)$).

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 $O(B\sqrt{B})$. 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. [19] 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. Our main observations are, when every party samples one index, the probability of having index collision in a batch can be modeled as a generalized birthday problem [2]. Besides, when sampling from the same range $[N]$, the probability of index collision decreases when we put less ciphertexts into one batch.

Based on the above observations, 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 D.3.

For practical settings, we guarantee 40 bits of statistical correctness by convention [19], meaning that the probability of index collision is smaller than 2^{-40} . Our scheme already provides 41 bits of statistical correctness when $B = 256, \alpha = 16, k = 1$. Here we present the probability of having index collision for different settings in Table 1. We calculate all the probability 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 D.3, 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	α	N	$-\log_2 P$
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 [19] for small B , depending on specific requirement of the applications. As a reference, the time complexity of the matching technique in [19] 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 two 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 $BTE.Enc$ 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.¹¹ 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 Combine is the most expensive operation, as it involves computing pairings. Recall that 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 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 degenerated mode,

¹¹In our construction, the adversary could release a random group element in \mathbb{G}_2

where *everyone* has to perform the expensive computation in 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 [13] who similarly perform expensive pairings for aggregation.¹²

We propose to establish an additional property of *verifiable aggregation*, which adds a proof π_{agg} to the output of 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 Combine, excluding obvious solutions such as 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 $(\text{com}, \text{op}) \xleftarrow{\$} \text{Commit}(m)$ where com is the commitment and op is the opening value. The client encrypts the message and the opening $m||\text{op}$ instead of just the message and also adds com to the ciphertext. Finally, the client adds a NIZK-proof to prove 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 Combine to decrypt the batch and publish $m||\text{op}$ for all transactions in the batch, essentially involving no computational overhead. Anyone (including a smart contract) can then very efficiently verify the commitment $\text{Verify}(\text{com}, m, \text{op})$ 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.

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` [17] library the cryptographic primitives and pairing operations. The source code is public.¹³

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.

¹²The construction in [13] still requires $O(B)$ pairings for aggregation, which is too expensive for on-chain computation.

¹³<https://anonymous.4open.science/r/btd-impl>

One-time Setup. Similar to [13], 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 [13] 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 [12, 16, 26] to remove the trusted dealer. This also opens the door for efficient protocols to support dynamically changing committees. Choudhuri et al. [13] explore the possibility of using dynamic proactive secret sharing protocols such as [29] 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 $\alpha = 2 \cdot \sqrt{B}$ subbatches (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 [13]. We elect to compare our results to the construction from Choudhuri et al. [13], 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 [13] 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 [13] 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.¹⁴

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 [13], we do not require any pairing operations for the verification of the proofs nor for the generation of the decryption share.

Batch Size	[13]	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 3: Partial decryption Time in ms. The comparison to [13] is based on different implementation and hardware.

We compare our measurements for partial decryption with different degrees of optimization to the results from [13] in Figure 3. 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 [13] 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 4.

¹⁴The proof size is not as relevant, as it does not need to be persisted on-chain.

Batch Size	[13]	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 4: Aggregation time given a batch of ciphertexts of size B and the according decryption shares. The comparison to [13] 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.

Practical advantages of requiring no Epoch Setup. In contrast to [13] 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 [13], 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} in practice than the construction from [13]. 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.

References

- [1] 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] Joseph Bebel and Dev Ojha. Ferveo: Threshold decryption for mempool privacy in bft networks. *Cryptology ePrint Archive*, 2022.
- [4] 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.
- [5] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.
- [6] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 440–456. Springer, 2005.
- [7] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Annual international cryptology conference*, pages 258–275. Springer, 2005.
- [8] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, Lecture Notes in Computer Science, pages 280–300, 2013.
- [9] 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, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II*, Lecture Notes in Computer Science, pages 407–437, 2019.
- [10] 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 Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 1–30, 2015.
- [11] Jeffrey Burdges and Luca De Feo. Delay encryption. In *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, Lecture Notes in Computer Science, pages 302–326, 2021.
- [12] Ran Canetti, Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In *Advances in Cryptology—CRYPTO’99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pages 98–116. Springer, 1999.
- [13] 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, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024.
- [14] Dan Cline, Thaddeus Dryja, and Neha Narula. Clockwork: an exchange protocol for proofs of non front-running. The Stanford Blockchain Conference, 2020.
- [15] 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. IEEE, 2020.
- [16] 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.
- [17] DEDIS. kyber: Advanced Crypto Library for Go. <https://github.com/dedis/kyber>, 2023.
- [18] Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wöhrig. Mcfly: Verifiable encryption to the future made practical. In *Financial Cryptography and Data Security - 27th International Conference, FC 2023, Bol, Brač, Croatia, May 1-5, 2023, Revised Selected Papers, Part I*, volume 13950 of *Lecture Notes in Computer Science*, pages 252–269, 2023.
- [19] Jesko Dujmovic, Rachit Garg, and Giulio Malavolta. Time-lock puzzles with efficient batch solving. In *Annual International Conference on the Theory and Appli-*

- cations of Cryptographic Techniques*, pages 311–341, 2024.
- [20] 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, St. Kitts, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23*, pages 170–189. Springer, 2020.
- [21] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. Cryptology ePrint Archive, Paper 2017/620, 2017.
- [22] Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind schnorr signatures and signed ElGamal encryption in the algebraic group model. Cryptology ePrint Archive, Paper 2019/877, 2019.
- [23] 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 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, Lecture Notes in Computer Science, pages 63–95, 2020.
- [24] 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 of *Lecture Notes in Computer Science*, pages 396–402, 1985.
- [25] 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, Copenhagen, Denmark, November 26-30, 2023*, pages 1093–1107, 2023.
- [26] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*, pages 295–310. Springer, 1999.
- [27] 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.
- [28] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, Lecture Notes in Computer Science, pages 276–288, 1984.
- [29] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. In *IACR International Conference on Public-Key Cryptography*, pages 252–282. Springer, 2022.
- [30] Philip Hall. On representatives of subsets. *Classic Papers in Combinatorics*, pages 58–62, 1987.
- [31] Feng Hao. Schnorr non-interactive zero-knowledge proof. Technical report, 2017.
- [32] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010.
- [33] Alireza Kavousi, Duc V Le, Philipp Jovanovic, and George Danezis. Blindperm: Efficient mev mitigation with an encrypted mempool and permutation. *Cryptology ePrint Archive*, 2023.
- [34] 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.
- [35] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *Advances in Cryptology—CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III 40*, pages 451–480. Springer, 2020.
- [36] 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, Berlin, Germany, November 4-8, 2013*, pages 669–684, 2013.
- [37] 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.

- [38] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. Homomorphic time-lock puzzles and applications. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 620–649, 2019.
- [39] Dahlia Malkhi and Pawel Szalachowski. Maximal extractable value (mev) protection on a dag. *arXiv preprint arXiv:2208.00940*, 2022.
- [40] 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.
- [41] 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. Springer, 2022.
- [42] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceedings*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, 1999.
- [43] Julien Piet, Vivek Nair, and Sanjay Subramanian. Mevade: An mev-resistant blockchain design. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9. IEEE, 2023.
- [44] 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. IEEE, 2021.
- [45] Ronald L. Rivest, Adi Shamir, and David Wagner. Time-lock puzzles and timed-release crypto. 1996.
- [46] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, *Lecture Notes in Computer Science*, pages 239–252, 1989.
- [47] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *J. Cryptol.*, 2002.
- [48] 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*, *Lecture Notes in Computer Science*, pages 554–584, 2023.
- [49] 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 *IACR International Conference on Public-Key Cryptography*, pages 554–584. Springer, 2023.
- [50] Chrysoula Stathakopoulou, Signe Rüsçh, 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. IEEE, 2021.
- [51] 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.
- [52] 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.
- [53] Haoqian Zhang, Louis-Henri Merino, Ziyang 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)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

A Additional Related Works

Here we provide additional related works that were not already discussed in Section 1.3, to give a more comprehensive overview of the background.

Miner/Maximal Extractable Value (MEV) MEV has long been a problem in DeFi systems, and the term was first introduced by Daian et al. [15]. Researches have been conducted to discuss and understand MEV, many of which has shown that

MEV is a significant problem in blockchain [20, 27, 51]. Not only does MEV cause economic loss to users, but it also poses a threat to the security of the blockchain, as it could incentive unnecessary forks [15]. Different approaches have been proposed to mitigate the problem of MEV. Flashbots [15], 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. Other popular approaches to mitigate MEV include threshold encryption, time-lock encryption, and TEE-based schemes, which we discussed in Section 1.3.

Order Fairness 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 [37], and several previous works [34, 35] discussed variants of weaker order fairness, i.e., block order fairness. Wendy [37] further discussed this problem and proposed several protocols for different level of fairness. While their work provides a certain degree of fairness, the solutions do not solve MEV completely, and is hard to be deployed since it requires new consensus protocols.

B Definitions, Games and Construction Details

B.1 Key-Homomorphic Puncturable PRF

Definition B.1 (Puncturable Pseudorandom Functions). A puncturable pseudorandom function family on key space $\mathcal{K} = \{\mathcal{K}_\lambda\}_{\lambda \in \mathbb{N}}$, keyspace $\mathcal{G} = \{\mathcal{G}_\lambda\}_{\lambda \in \mathbb{N}}$, 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 $\text{PRF} = (\text{Setup}, \text{KeyGen}, \text{Puncture}, \text{Eval}, \text{ExpEval}, \text{PEval})$ such that:

- $\text{pp} \xleftarrow{\$} \text{Setup}(1^\lambda, n)$. Setup takes the security parameter λ as well as the domain parameter n as input and outputs some public parameters pp .
- $k \xleftarrow{\$} \text{KeyGen}(\text{pp})$. KeyGen takes the public parameters as input and returns a key $k \in \mathcal{K}_\lambda$.
- $k^* \leftarrow \text{Puncture}(\text{pp}, k, i^*)$. 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^* .
- $y \leftarrow \text{Eval}(\text{pp}, k, i)$. 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 $y \in \mathcal{Y}_\lambda$.
- $y \leftarrow \text{ExpEval}(\text{pp}, K, i)$. 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 $y \in \mathcal{Y}_\lambda$.

- $y \leftarrow \text{PEval}(\text{pp}, k^*, i^*, i)$. 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 $y \in \mathcal{Y}_\lambda$.

Definition B.2 (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 $\text{negl}(\lambda)$ such that

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

where $\text{Game-PRF}_{\mathcal{A}}$ is defined in Figure 5.

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

Figure 5: Pseudorandomness game of PRF.

Key-Homomorphic Punctured PRF Construction.

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

Note that this setup is adapted from the key-homomorphic punctured PRF of [19], where we additionally publish $\{g_1^{z_i}\}_{i \in [n]}$.

- $\text{KeyGen}(\text{pp})$: Sample $k \xleftarrow{\$} \mathbb{Z}_p^*$ and return k .
- $\text{Puncture}(\text{pp}, k, i^*)$: Return $k^* \leftarrow (g_2^{x_{i^*}})^k = g_2^{x_{i^*} k}$.
- $\text{Eval}(\text{pp}, k, i)$:
 - Choose an index $j \in [n]$ with $j \neq i$.
 - Return $y \leftarrow e(g_1^{z_i/x_j}, g_2^j)^k$.

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 $g_T^{z_i^i} = e(g_1^{z_i^i}, g_2)$ using the public setup.

- $\text{ExpEval}(\text{pp}, K, i)$: Return $y \leftarrow e(g_1^{z_i^i}, K)$. For $K = g_2^k$, this yields $y = g_T^{z_i^k}$.
- $\text{PEval}(\text{pp}, k^*, i^*, i)$:
 - If $i = i^*$, return \perp .
 - Otherwise, compute $y \leftarrow e(g_1^{z_i^i/x_{i^*}^*}, k^*) = e(g_1^{z_i^i/x_{i^*}^*}, g_2^{x_{i^*}^* k^*}) = g_T^{z_i^k}$.

Output y .

Correctness and Pseudorandomness. The correctness is straightforward from the description above. The proof of pseudorandomness from [19] carries over with our modification. The only modification we make is including $g_1^{z_i^i}$ for every i in the setup. This modification does not change the security proof in [19] since one can easily modify their reduction to also publish $g_1^{z_i^i}$.

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

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

B.2 Missing Security Games and Definitions.

Definition B.3 (Threshold Homomorphic Encryption protocol). A threshold homomorphic encryption protocol THE is a tuple of PPT algorithms $\text{THE} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec}, \text{Combine})$ with the following syntax.

- $\text{pp} \xleftarrow{\$} \text{Setup}(1^\lambda)$. Setup is a probabilistic algorithm that takes the security parameter $\lambda \in \mathbb{N}$ as input and outputs some public parameters pp.
- $(\text{pk}, \{\text{sk}_i\}_{i \in [n]}) \xleftarrow{\$} \text{KeyGen}(\text{pp}, n, t)$. KeyGen is a probabilistic algorithm that takes the public parameters pp, the number of servers $n \in \mathbb{N}$, and the threshold $t \in \mathbb{N}$ where $0 < t \leq n$. It returns a public key pk as well as n secret key shares $\{\text{sk}_i\}_{i \in [n]}$.
- $c \xleftarrow{\$} \text{Enc}(\text{pk}, m)$. Enc is a probabilistic algorithm that receives a public key pk and a message m as input, returning a ciphertext c .
- $d_i/\perp \leftarrow \text{Dec}(\text{sk}_i, c)$. Dec is a deterministic algorithm that takes a secret key share sk_i and a ciphertext c as input. It outputs the corresponding decryption share d_i .

- $m/\perp \leftarrow \text{Combine}(\text{pk}, \{d_i\}_{i \in S}, S, c)$. Combine is a deterministic algorithm that takes as input the public key pk, a set of decryption shares $\{d_i\}_{i \in S}$ for $|S| \geq t$ as well as the ciphertext c . It combines the decryption shares to decrypt c and outputs the message m or \perp upon failure.

Definition B.4 (Correctness of THE). A homomorphic threshold encryption protocol THE is correct if for all λ and $\text{pp} \xleftarrow{\$} \text{Setup}(1^\lambda)$, all $0 < t \leq n$, all $(\text{pk}, \{\text{sk}_i\}_{i \in [n]}) \xleftarrow{\$} \text{KeyGen}(\text{pp}, n, t)$, all $m \in \mathcal{M}_\lambda$, all $c \xleftarrow{\$} \text{Enc}(\text{pk}, m)$ and all $S \subseteq [n]$ with $|S| \geq t$ it holds that $m = \text{Combine}(\text{pk}, \{\text{Dec}(\text{sk}_i, c)\}_{i \in S}, S, c)$.

Definition B.5 (Multiplicative Message-Homomorphism of THE). A THE protocol with message-space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ and ciphertext-space $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ is called multiplicatively message-homomorphic if for all λ and $\text{pp} \xleftarrow{\$} \text{Setup}(1^\lambda)$ ($\mathcal{M}_\lambda, \cdot$) is a group and $(\mathcal{C}_\lambda, *)$ is a group and for all $0 < t \leq n$, all $m_1, m_2 \in \mathcal{M}_\lambda$, all $(\text{pk}, \{\text{sk}_i\}_{i \in [n]}) \xleftarrow{\$} \text{KeyGen}(\text{pp}, n, t)$, all $c_1 \xleftarrow{\$} \text{Enc}(\text{pk}, m_1)$, $c_2 \xleftarrow{\$} \text{Enc}(\text{pk}, m_2)$ and $c \xleftarrow{\$} \text{Enc}(\text{pk}, m_1 \cdot m_2)$ as well as all sets $S, S' \subseteq [n]$ with $|S| \geq t$ and $|S'| \geq t$ it holds that $\text{Combine}(\text{pk}, \{\text{Dec}(\text{sk}_i, c)\}_{i \in S}, S, c) = \text{Combine}(\text{pk}, \{\text{Dec}(\text{sk}_i, c_1 * c_2)\}_{i \in S'}, S', c_1 * c_2)$.

<p style="text-align: center;">Game-IND-CPA$_{\mathcal{A}}(1^\lambda)$</p> <hr/> $\text{pp} \xleftarrow{\$} \text{Setup}(1^\lambda)$ $(\text{pk}, \{\text{sk}_i\}_{i \in [n]}) \xleftarrow{\$} \text{KeyGen}(\text{pp}, n, t)$ $(C, \text{st}_1) \xleftarrow{\$} \mathcal{A}_1(\text{pp}, \text{pk})$ if $C \not\subseteq [n] \vee C \geq t$ then return 0 $(m_0, m_1, \text{st}_2) \xleftarrow{\$} \mathcal{A}_2(\text{st}_1, \{\text{sk}_i\}_C)$ $b \xleftarrow{\$} \{0, 1\}$ $c \leftarrow \text{Enc}_1(\text{pk}, m_b)$ $b' \xleftarrow{\$} \mathcal{A}_3(\text{st}_2, c)$ return $b \stackrel{?}{=} b'$

Figure 6: IND-CPA game of THE.

Definition B.6 (IND-CPA of THE). A threshold homomorphic encryption protocol THE is IND-CPA secure if for all PPT adversaries $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ there exists a negligible function $\text{negl}(\lambda)$ such that

$$\Pr[\text{Game-IND-CPA}_{\mathcal{A}}^{\text{THE}}(\lambda) = 1] \leq \frac{1}{2} + \text{negl}(\lambda)$$

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

Definition B.7 (Correctness of BTE). A Batched Threshold Encryption scheme BTE is correct if for all $\lambda, n, t, B_{\max} \in \mathbb{N}$ with $n \geq t$, all $\text{pp} \xleftarrow{\$} \text{Setup}(\lambda, B_{\max})$, all $(\text{pk}, \{\text{sk}_\ell\}_{\ell \in [n]}) \xleftarrow{\$} \text{KeyGen}(\lambda, n, t)$, all $B \in [B_{\max}]$, all $(m_1, \dots, m_B) \in \mathcal{M}_\lambda^B$, all $S \subseteq [n]$ with $|S| \geq t$ we get that

$$\forall i \in [B] : \text{Verify}(\text{pk}, c_i, \pi_i) = 1$$

and

$$\begin{aligned} & \text{Combine}(\text{pk}, \{d_\ell\}_{\ell \in S}, S, \{c_i\}_{i \in [B]}) \\ &= \{m_i\}_{i \in [B]} \end{aligned}$$

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

C NIZK Proof System for CCA-security

The construction of our proof system Π for the language $\mathcal{L} = \{\chi : (\chi, \omega) \in \mathcal{R}\}$ is shown in Figure 7. Prove takes as input the statement $\chi = (\text{pp}, \text{pk}, c)$ and the witness $\omega = (k, u)$, and outputs a proof π . Verify takes as input the statement χ and the proof π , and outputs 1 if the proof is valid, and 0 otherwise. Recall that $c = (i, k^*, \gamma, ct)$, and $ct = (A, B) = (g^u, \text{pk}^u g_2^k)$. For simplicity, we write $y = k^* = g_2^{x_i^* k}$ in our proof system.

Prove(χ, ω)	Verify(χ, π)
$u' \xleftarrow{\$} \mathbb{Z}_p$	$f \leftarrow H(\chi, A', B', y')$
$k' \xleftarrow{\$} \mathbb{Z}_p$	Check:
$A' \leftarrow g^{u'}$	$g^{\hat{u}} \stackrel{?}{=} A' \cdot A^f$
$B' \leftarrow \text{pk}^{u'} g_2^{k'}$	$\text{pk}^{\hat{u}} g_2^{\hat{k}} \stackrel{?}{=} B' \cdot B^f$
$y' \leftarrow (g_2^{x_i^*})^{k'}$	$(g_2^{x_i^*})^{\hat{k}} \stackrel{?}{=} y' \cdot y^f$
$f \leftarrow H(\chi, A', B', y')$	output 1 if all checks pass,
$\hat{u} \leftarrow u' + fu$	else output 0.
$\hat{k} \leftarrow k' + fk$	
output $\pi = (A', B', y', \hat{u}, \hat{k})$	

Figure 7: Proof system for Batched Threshold Encryption scheme BTE with knowledge of witness k, u

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

Definition C.1 (Completeness). The proof system Π is complete if $\forall \chi \in \mathcal{L}, \omega \in \mathcal{R}(\chi)$, i.e. there exists a valid witness ω for χ , $\Pr[\pi \leftarrow \text{Prove}(\chi, \omega) : \text{Verify}(\chi, \pi) = 1] = 1$.

Completeness means that an honest prover can always convince the honest verifier with true statements. To prove completeness, suppose the prover uses the correct witness and follows the Prove algorithm. The proof system is complete because: $A' \cdot A^f = g^{u'} \cdot g^{fu} = g^{u'+fu} = g^{\hat{u}}$, $B' \cdot B^f = \text{pk}^{u'} g_2^{k'} \cdot \text{pk}^{fu} g_2^{fk} = \text{pk}^{u'+fu} g_2^{k'+fk} = \text{pk}^{\hat{u}} g_2^{\hat{k}}$, 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 C.2 (Soundness). The proof system Π is sound if for any PPT adversary \mathcal{A} , if $\chi \notin \mathcal{L}$, $\Pr[\pi \leftarrow \mathcal{A}(\chi) : \text{Verify}(\chi, \pi) = 1] = \text{negl}(\lambda)$.

Soundness guarantees that no adversary can convince the verifier with false statements with non-negligible probability. We prove statistical soundness of Π . Assuming $\text{Verify}(\chi, \pi)$ accepts the proof, we show that the statement χ must be in the language \mathcal{L} .

Without loss of generality, we assume $\chi = (\text{pp}, \text{pk}, c)$, $c = (i, k_0^*, \gamma, ct)$, $ct = (A, B) = (g^{u_0}, \text{pk}^{u_0} g_2^{k_0})$, and $y = k_0^* = g_2^{x_i^* k_0}$. Similarly, we can assume $A' = g^{u'_0}, B' = \text{pk}^{u'_0} g_2^{k'_0}, y' = g_2^{x_i^* k'_0}$. We have $k_0, k_1, u_0, u_1, k'_0, k'_1, u'_0, u'_1 \in \mathbb{Z}_p$. The malicious prover can generate all the values differently.

Since the Verify algorithm accepts the proof, we have:

- $g^{\hat{u}} = A' \cdot A^f = g^{u'_0} \cdot g^{fu_0} = g^{u'_0+fu_0}$, 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 $\hat{k} = k'_0 + fk_0$.
- $\text{pk}^{\hat{u}} g_2^{\hat{k}} = B' \cdot B^f = \text{pk}^{u'_0} g_2^{k'_0} \cdot \text{pk}^{fu_0} g_2^{fk_0} = \text{pk}^{u'_0+fu_0} g_2^{k'_0+fk_0}$.

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

Therefore, combining the above equations, we proved that there must exist a witness $\omega(k, u)$ where $k = k_0 = k_1$ and $u = u_0 = u_1$, indicating statistical soundness of the proof system Π .

Definition C.3 (Zero-Knowledge). $\forall \chi \in \mathcal{L}$, there exists a PPT simulator Sim , such that the two results are computationally indistinguishable: $\{\text{Sim}(\chi)\} \approx \{\text{Prove}(\chi, \omega)\}$.

Zero-knowledge means that the verifier Verify learns nothing from the proof π except the validity of the statement χ . We prove that the proof system Π is honest-verifier 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 gets the statement $\chi = (\text{pp}, \text{pk}, c)$ as input, with $c = (i, k^*, \gamma, ct)$ and $ct = (A, B) = (g^u, \text{pk}^u g_2^k)$, and generates a simulated proof $\pi = (A', B', y', \hat{u}, \hat{k})$ as follows:

- Choose $\hat{u} \xleftarrow{\$} \mathbb{Z}_p, \hat{k} \xleftarrow{\$} \mathbb{Z}_p$.
- Choose $f \xleftarrow{\$} \mathbb{Z}_p$.

- Compute $A' = g^{\hat{u}}A^{-f}$, $B' = \text{pk}^{\hat{u}}g_2^{\hat{k}}B^{-f}$, 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 as the real proof. Because A', B', y' are computed according to the check equations, the verifier Verify will accept the simulated proof π as a real proof. Thus, the simulator Sim can form a valid proof without knowing the witness. The proof system satisfied honest-verifier zero-knowledge.

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

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

Here, we also require the extractor to be online (or straight-line).

Remark C.5 We notice that the same techniques used in [23] to prove non-malleability in Schnorr-Signed El-Gamal Encryption in the ROM and AGM using straight-line extractor can be used in our construction. In this technique, instead of rewinding, we use the AGM assumption that any group element in the output of the adversary should be a linear combination of the group elements that were given to the adversary. Hence, we can construct an extractor by exploiting the linear dependency between the group elements that are given to the adversary as response to the oracle queries and its responses. Observe that our proof system Π uses the same technique used to construct the proofs in [23].

D Security Proofs

Proof of Theorem 4.1. Correctness and message-homomorphism follow directly from the correctness and message-homomorphism of standard ElGamal encryption. IND-CPA security follows straight from the IND-CPA security of standard ElGamal encryption and the security of Shamir's secret sharing, as one can trivially simulate up to $t - 1$ Shamir shares for corrupted parties in Game-IND-CPA (Figure 6) by sampling random field elements. ■

D.1 Proof of of Theorem 5.1

Proof of Theorem 5.1. We prove B-IND-CCA security of BTE (Definition 3.2) through a series of game-hops where we eliminate all dependencies on the internal bit b of

Game-B-IND-CCA. Let $\text{Game-B-IND-CCA}_{0, \mathcal{A}}^{\text{BTE}}$ be the original game defined in Figure 1.

Game-B-IND-CCA $_{1, \mathcal{A}}^{\text{BTE}}$: In this game, we change how the proof π is computed by the game during encryption of the challenge m_b . Instead of computing π as $\Pi.\text{Prove}(\chi, \omega)$, where $\chi = (\text{pp}, \text{pk}, c)$ and $\omega = (k, u)$, we simulate the proof using the simulator Sim . Hence, we set $\pi \leftarrow \text{Sim}(\text{pp}, \text{pk}, c)$.

Claim D.1 If the proof system Π is simulable (Definition C.3) then $\text{Game-B-IND-CCA}_{1, \mathcal{A}}^{\text{BTE}}$ is computationally indistinguishable from $\text{Game-B-IND-CCA}_{0, \mathcal{A}}^{\text{BTE}}$. $|\Pr[\text{Game-B-IND-CCA}_{1, \mathcal{A}}^{\text{BTE}}(1^\lambda) = 1] - \Pr[\text{Game-B-IND-CCA}_{0, \mathcal{A}}^{\text{BTE}}(1^\lambda) = 1]| \leq \text{negl}(\lambda)$

Proof of Claim D.1. The claim follows straight from the zero-knowledge property of the proof system Π . ■

In a following game-hop, we want to make a reduction to the IND-CPA-security of the underlying threshold ElGamal encryption scheme. In order to do that, we need to simulate the batch-decryption oracle $O^{\text{b-dec}}$ though, which has a dependency on the threshold ElGamal secret key shares sk_ℓ . Because we only require the threshold homomorphic encryption to be IND-CPA-secure, we do not have access to a decryption oracle, which we could use to simulate the partial threshold ElGamal decryption. Hence, we remove the dependency on sk_ℓ of $O^{\text{b-dec}}$ within the following two game-hops using the simulation-extractability of the proof system Π to simulate decryption shares.

Game-B-IND-CCA $_{2, \mathcal{A}}^{\text{BTE}}$: In this game, we make changes to the batch-decryption oracle $O^{\text{b-dec}}$. After verifying each proof π_i , we extract the witnesses $(\tilde{k}_i, \tilde{u}_i)$. We then recompute the punctured key \tilde{k}'_i and the ElGamal encryption of $g_2^{\tilde{k}'_i}$ and check that they match the respective components in c_i . If any of the checks fail, we abort. The following Figure 8 shows the changes to the $O^{\text{b-dec}}$ oracle described above.

Claim D.2 If the underlying proof system Π is simulation-extractable (Definition C.4) then $\text{Game-B-IND-CCA}_{2, \mathcal{A}}^{\text{BTE}}$ is computationally indistinguishable from $\text{Game-B-IND-CCA}_{1, \mathcal{A}}^{\text{BTE}}$. $|\Pr[\text{Game-B-IND-CCA}_{2, \mathcal{A}}^{\text{BTE}}(1^\lambda) = 1] - \Pr[\text{Game-B-IND-CCA}_{1, \mathcal{A}}^{\text{BTE}}(1^\lambda) = 1]| \leq \text{negl}(\lambda)$

Proof of Claim D.2. Clearly the only way for a PPT adversary \mathcal{A} to distinguish between the games is by triggering the additional abort conditions in $O_2^{\text{b-dec}}$. To do so, \mathcal{A} must submit a pair (c_i, π_i) with a verifying proof π_i such that the extractor Extract fails to extract the correct witness $(\tilde{k}_i, \tilde{u}_i)$. The simulation-extractability of Π implies that \mathcal{A} 's probability of success is negligible. Given an adversary \mathcal{A} that submits a total of q ciphertexts for decryption to $O^{\text{b-dec}}$, we get

$$|\Pr[\text{Game-B-IND-CCA}_{2, \mathcal{A}}^{\text{BTE}}(1^\lambda) = 1] - \Pr[\text{Game-B-IND-CCA}_{1, \mathcal{A}}^{\text{BTE}}(1^\lambda) = 1]| \leq q \cdot \text{negl}(\lambda)$$

Oracle $O_1^{\text{b-dec}}(\ell, \{(c_i, \pi_i)\}_{i \in [B]})$	Oracle $O_2^{\text{b-dec}}(\ell, \{(c_i, \pi_i)\}_{i \in [B]})$
1: if $c^* \in \{c_i\}_{[B]}$ then	1: if $c^* \in \{c_i\}_{[B]}$ then
2: $ctr \leftarrow ctr + 1$	2: $ctr \leftarrow ctr + 1$
3: if $ctr \geq t - C $ then return \perp	3: if $ctr \geq t - C $ then return \perp
4: for $i \in [B]$ do	4: for $i \in [B]$ do
5: if $\Pi.\text{Verify}(\text{pp}, \text{pk}, c_i, \pi_i) = 0$ then return \perp	5: if $\Pi.\text{Verify}(\text{pp}, \text{pk}, c_i, \pi_i) = 0$ then return \perp
6: return $\text{BatchDec}(\text{sk}_\ell, \{c_i\}_{[B]})$	6: $(\tilde{k}_i, \tilde{u}_i) \xleftarrow{\$} \text{Extract}((\text{pp}, \text{pk}, c_i), \pi_i, Q)$
	7: $\tilde{k}'_i \leftarrow g_2^{x_i \tilde{k}_i}$
	8: if $\tilde{k}'_i \neq c_i.k'$ then return \perp
	9: $\tilde{c}_i \leftarrow (g_2^{\tilde{u}_i}, \text{pk}^{\tilde{u}_i} g_2^{\tilde{k}_i})$
	10: if $\tilde{c}_i \neq c_i.ct$ then return \perp
	11: return $\text{BatchDec}(\text{sk}_\ell, \{c_i\}_{[B]})$

Figure 8: Game hop from Game-B-IND-CCA₁ to Game-IND-CCA₂. The changes to the $O^{\text{b-dec}}$ oracle are highlighted in grey.

which is negligible because q is bounded by a polynomial in λ . \blacksquare

Game-B-IND-CCA_{3, \mathcal{A}} ^{BTE}: In this game we make another change to $O^{\text{b-dec}}$, which finally removes its dependency on sk_ℓ . During the BatchDec , we assemble all the ElGamal ciphertexts into $\text{CT} \leftarrow \prod_{i=1}^B ct_i = (X, Y) = (g^u, \text{pk}^u \cdot g^k)$ where $u = \sum_{i=1}^B u_i$ and $k = \sum_{i=1}^B k_i$. In $O_2^{\text{b-dec}}$, we are computing a decryption share of CT under sk_ℓ as $d_\ell \leftarrow X^{\text{sk}_\ell} [= g^{u \cdot \text{sk}_\ell}]$, as mandated by the threshold ElGamal decryption algorithm. In $O_3^{\text{b-dec}}$ we directly compute the decryption share as $\tilde{d}_\ell = \text{pk}^{\tilde{u}} [= g^{\tilde{u} \cdot \text{sk}_\ell}]$ where $\tilde{u} = \sum_{i=1}^B \tilde{u}_i$. Note that $\tilde{d}_\ell = d_\ell$, given $u = \tilde{u}$. We detail the change in Figure 9.

Claim D.3 The games are identical.

Proof of Claim D.3. Given an ElGamal Ciphertext $c_i.ct = ct_i = (g_2^{u_i}, \text{pk}^{u_i} g_2^{k_i})$, observe that the extracted witnesses \tilde{k}_i and \tilde{u}_i must be equal to the values used in ct_i and we have $k_i = \tilde{k}_i$ and $u_i = \tilde{u}_i$. This is because there is only one unique $(\tilde{k}_i, \tilde{u}_i)$ such that $ct_i = (g_2^{\tilde{u}_i}, \text{pk}^{\tilde{u}_i} g_2^{\tilde{k}_i})$ and both games abort if this condition is not met. Hence $\tilde{u} = \sum_{i=1}^B \tilde{u}_i = \sum_{i=1}^B u_i = u$, which implies that the returned decryption shares are equal in both games:

$$\tilde{d}_\ell = \text{pk}^{\tilde{u}} = g_2^{\tilde{u} \cdot \text{sk}_\ell} = g_2^{u \cdot \text{sk}_\ell} = d_\ell$$

Now that we have removed the dependency on the threshold ElGamal secret key shares sk_ℓ from the batch-decryption oracle, we can proceed with a game hop that replaces the ElGamal encryption of g_2^k in the challenge ciphertext with an encryption of an arbitrary constant (say g_2). We can reduce the indistinguishability of this game hop to the IND-CPA security of threshold ElGamal, because we no longer need

the sk_ℓ in the batch-decryption oracle.

Game-B-IND-CCA_{4, \mathcal{A}} ^{BTE}: In this game we change how the challenge is computed in Game-B-IND-CCA and replace the encryption of g_2^k with an encryption of g_2 . Instead of computing $\text{ct} \leftarrow \text{THE}.\text{Enc}(\text{pk}, g_2^k)$, we set $\text{ct} \leftarrow \text{THE}.\text{Enc}(\text{pk}, g_2)$. Note that we could use an arbitrary group element instead of g_2 . We just choose g_2 to avoid introducing an additional constant.

Claim D.4 If the underlying homomorphic encryption scheme THE (which is ElGamal) is IND-CPA-secure in \mathbb{G}_2 then Game-B-IND-CCA_{4, \mathcal{A}} ^{BTE} is computationally indistinguishable from Game-B-IND-CCA_{3, \mathcal{A}} ^{BTE}. $|\Pr[\text{Game-B-IND-CCA}_{4, \mathcal{A}}^{\text{BTE}}(1^\lambda) = 1] - \Pr[\text{Game-B-IND-CCA}_{3, \mathcal{A}}^{\text{BTE}}(1^\lambda) = 1]| \leq \text{negl}(\lambda)$

Proof of Claim D.4. We proof this claim by reduction to the IND-CPA security of the underlying threshold homomorphic encryption scheme THE (Definition B.6). Let \mathcal{A} be a PPT adversary such that $|\Pr[\text{Game-B-IND-CCA}_{4, \mathcal{A}}^{\text{BTE}}(1^\lambda) = 1] - \Pr[\text{Game-B-IND-CCA}_{3, \mathcal{A}}^{\text{BTE}}(1^\lambda) = 1]| > \epsilon(\lambda)$ for a non-negligible ϵ . We construct a PPT reduction \mathcal{B} that runs in Game-IND-CPA _{\mathcal{B}} ^{THE} of THE and uses \mathcal{A} internally to break the IND-CPA-security of THE.

\mathcal{B} receives pk from Game-IND-CPA _{\mathcal{B}} ^{THE} and runs \mathcal{A} with (pp, pk) as input. It forwards the set of corrupted parties C to Game-IND-CPA _{\mathcal{B}} ^{THE}, passing the resulting secret key shares $\{\text{sk}_\ell\}_{\ell \in C}$ back to \mathcal{A} . It follows the steps from Game-B-IND-CCA₄ and Game-B-IND-CCA₃ up to the point where it is supposed to encrypt g_2^k or g_2 respectively. Instead, \mathcal{B} sends $(m_0 = g_2^k, m_1 = g_2)$ to Game-IND-CPA _{\mathcal{B}} ^{THE} and receives ct as a response. \mathcal{B} then continues following the steps from Game-B-IND-CCA₄ and Game-B-IND-CCA₃.

Oracle $O_2^{\text{b-dec}}(\ell, \{(c_i, \pi_i)\}_{i \in [B]})$	Oracle $O_3^{\text{b-dec}}(\ell, \{(c_i, \pi_i)\}_{i \in [B]})$
1: if $c^* \in \{c_i\}_{[B]}$ then return \perp	1: if $c^* \in \{c_i\}_{[B]}$ then return \perp
2: for $i \in [B]$ do	2: for $i \in [B]$ do
3: if $\text{Verify}(\text{pp}, \text{pk}, c_i, \pi_i) = 0$ then return \perp	3: if $\text{Verify}(\text{pp}, \text{pk}, c_i, \pi_i) = 0$ then return \perp
4: $(\tilde{k}_i, \tilde{u}_i) \stackrel{\$}{\leftarrow} \text{Extract}(\text{pp}, \text{pk}, c_i, \pi_i)$	4: $(\tilde{k}_i, \tilde{u}_i) \stackrel{\$}{\leftarrow} \text{Extract}(\text{pp}, \text{pk}, c_i, \pi_i, Q)$
5: $\tilde{k}'_i \leftarrow g_2^{x_i \tilde{k}_i}$	5: $\tilde{k}'_i \leftarrow g_2^{x_i \tilde{k}_i}$
6: if $\tilde{k}'_i \neq c_i \cdot k'$ then return \perp	6: if $\tilde{k}'_i \neq c_i \cdot k'$ then return \perp
7: $\tilde{c}t_i \leftarrow (g_2^{\tilde{u}_i}, \text{pk}^{\tilde{u}_i} g_2^{\tilde{k}_i})$	7: $\tilde{c}t_i \leftarrow (g_2^{\tilde{u}_i}, \text{pk}^{\tilde{u}_i} g_2^{\tilde{k}_i})$
8: if $\tilde{c}t_i \neq c_i \cdot ct$ then return \perp	8: if $\tilde{c}t_i \neq c_i \cdot ct$ then return \perp
9: $\text{CT} \leftarrow \prod_{i=1}^B ct_i$	9: $\tilde{u} \leftarrow \sum_{i=1}^B \tilde{u}_i$
10: $d_\ell \leftarrow \text{THE.Dec}(\text{sk}_\ell, \text{CT})$	10: $\tilde{d}_\ell \leftarrow \text{pk}^{\tilde{u}}$
11: return d_ℓ	11: return \tilde{d}_ℓ

Figure 9: Game hop from Game-B-IND-CCA₂ to Game-IND-CCA₃. The changes to the $O^{\text{b-dec}}$ oracle are highlighted in grey.

Note that \mathcal{B} can simulate the batch decryption oracle $O_3^{\text{b-dec}} = O_4^{\text{b-dec}}$ to \mathcal{A} , as it no longer depends on the ElGamal secret key share sk_ℓ . This dependency was removed in the previous game hop.

Analysis. Let b' be the internal bit of Game-IND-CPA $_{\mathcal{B}}^{\text{THE}}$. If $b' = 0$, then Game-IND-CPA $_{\mathcal{B}}^{\text{THE}}$ returns an encryption of g_2^k and \mathcal{B} simulates Game-B-IND-CCA $_{3, \mathcal{A}}^{\text{BTE}}$ to \mathcal{A} . If $b' = 1$, then Game-IND-CPA $_{\mathcal{B}}^{\text{THE}}$ returns an encryption of g_2 and \mathcal{B} simulates Game-B-IND-CCA $_{4, \mathcal{A}}^{\text{BTE}}$ to \mathcal{A} . Hence, \mathcal{B} wins Game-IND-CPA $_{\mathcal{B}}^{\text{THE}}$ if \mathcal{A} wins Game-B-IND-CCA $_{\mathcal{A}}^{\text{BTE}}$, which is assumed to be non-negligible. Denote \mathcal{B} 's output in Game-IND-CPA $_{\mathcal{B}}^{\text{THE}}$ with d .

$$\begin{aligned}
& \left| \Pr \left[\text{Game-IND-CPA}_{\mathcal{B}}^{\text{THE}}(1^\lambda) = 1 \right] \right| \\
&= \left| \frac{1}{2} \Pr [d = 0 \mid b' = 0] + \frac{1}{2} \Pr [d = 1 \mid b' = 1] \right| \\
&= \left| \frac{1}{2} (1 - \Pr [d = 1 \mid b' = 0]) + \Pr [d = 1 \mid b' = 1] \right| \\
&= \frac{1}{2} + \frac{1}{2} \left| -\Pr [\text{Game-B-IND-CCA}_{3, \mathcal{A}}^{\text{BTE}}(1^\lambda) = 1] \right. \\
&\quad \left. + \Pr [\text{Game-B-IND-CCA}_{4, \mathcal{A}}^{\text{BTE}}(1^\lambda) = 1] \right| \\
&> \frac{1}{2} + \frac{\varepsilon(\lambda)}{2}
\end{aligned}$$

This contradicts the IND-CPA security of THE. \blacksquare

Game-B-IND-CCA $_{5, \mathcal{A}}^{\text{BTE}}$: In this game, we replace the result of the PRF-evaluation during encryption with a random group element from \mathbb{G}_T . Instead of computing $\gamma \leftarrow m_b + \text{PRF}(k, i)$ we set $\gamma \leftarrow m_b + r$ for a random $r \stackrel{\$}{\leftarrow} \mathbb{G}_T$.

Claim D.5 If PRF is pseudorandom (Definition B.2) then Game-B-IND-CCA $_{5, \mathcal{A}}^{\text{BTE}}$ is computationally indistinguishable from Game-B-IND-CCA $_{4, \mathcal{A}}^{\text{BTE}}$. $|\Pr[\text{Game-B-IND-CCA}_{5, \mathcal{A}}^{\text{BTE}}(1^\lambda) = 1] - \Pr[\text{Game-B-IND-CCA}_{4, \mathcal{A}}^{\text{BTE}}(1^\lambda) = 1]| \leq \text{negl}(\lambda)$

Proof Sketch of Claim D.5. We proof this claim by reduction to the pseudorandomness Game-PR of PRF. The PPT reduction \mathcal{B} playing in Game-PR $_{\mathcal{B}}$ sends index i to the pseudorandomness game and receives k' and value r as a response. \mathcal{B} then computes γ as $\gamma \leftarrow m_b + r$. If the r is sampled uniformly random by the pseudorandomness game ($b = 0$), \mathcal{B} simulates Game-B-IND-CCA $_{5, \mathcal{A}}^{\text{BTE}}$ to \mathcal{A} . If r is the result of the PRF evaluation ($b = 1$), \mathcal{B} simulates Game-B-IND-CCA $_{4, \mathcal{A}}^{\text{BTE}}$ to \mathcal{A} . Hence, if \mathcal{A} can distinguish between Game-B-IND-CCA $_{5, \mathcal{A}}^{\text{BTE}}$ and Game-B-IND-CCA $_{4, \mathcal{A}}^{\text{BTE}}$, then \mathcal{B} can distinguish between $b = 0$ and $b = 1$. \blacksquare

Game-B-IND-CCA $_{6, \mathcal{A}}^{\text{BTE}}$: In this game we directly sample γ from \mathbb{G}_T instead of computing it as $m_b + r$ for a random $r \stackrel{\$}{\leftarrow} \mathbb{G}_T$.

Claim D.6 Both games are identically distributed and thus it holds that $\Pr [\text{Game-B-IND-CCA}_{6, \mathcal{A}}^{\text{BTE}}(1^\lambda) = 1] = \Pr [\text{Game-B-IND-CCA}_{5, \mathcal{A}}^{\text{BTE}}(1^\lambda) = 1]$

Proof of Claim D.6. The only difference between the games is the way γ is computed. In Game-B-IND-CCA $_{5, \mathcal{A}}^{\text{BTE}}$, γ is computed as $m_b + r$ for a random $r \stackrel{\$}{\leftarrow} \mathbb{G}_T$. In Game-B-IND-CCA $_{6, \mathcal{A}}^{\text{BTE}}$, γ is directly sampled from \mathbb{G}_T . Clearly, γ is identically distributed in both games. \blacksquare

We conclude the proof by arguing that the adversaries view in Game-B-IND-CCA $_{6, \mathcal{A}}^{\text{BTE}}$ no longer depends on the

internal bit b . Hence $\Pr\left[\text{Game-B-IND-CCA}_{6,\mathcal{A}}^{\text{BTE}}(1^\lambda) = 1\right] = \frac{1}{2}$. Further, we have derived $\text{Game-B-IND-CCA}_{6,\mathcal{A}}^{\text{BTE}}$ from $\text{Game-B-IND-CCA}_{0,\mathcal{A}}^{\text{BTE}}$ through a series of game-hops where each game is computationally indistinguishable from the previous one. We conclude that for all PPT adversaries \mathcal{A} there exists a negligible function $\text{negl}(\lambda)$ such that $\Pr[\text{Game-B-IND-CCA}_{0,\mathcal{A}}^{\text{BTE}}(1^\lambda) = 1] \leq \Pr[\text{Game-B-IND-CCA}_{6,\mathcal{A}}^{\text{BTE}}(1^\lambda) = 1] + \text{negl}(\lambda) = \frac{1}{2} + \text{negl}(\lambda)$ which satisfies Definition 3.2. ■

D.2 Proof of Theorem 5.2

Proof of Theorem 5.2. We prove rogue ciphertext security (Definition 3.3) by reduction to the soundness of the proof-system Π (Definition C.2). Let \mathcal{A} be a PPT adversary against rogue ciphertext security of BTE. We construct an adversary \mathcal{B} against the soundness of Π , which runs \mathcal{A} internally, simulating $\text{Game-Rogue}_{\mathcal{A}}$ to \mathcal{A} . \mathcal{B} simulates $\text{Game-Rogue}_{\mathcal{A}}$ exactly as described in Figure 1. If \mathcal{A} wins $\text{Game-Rogue}_{\mathcal{A}}$ in \mathcal{B} 's simulation, then \mathcal{B} learns $\{(c_j, \pi_j)\}_{j \in [B]}$ along the way. \mathcal{B} picks a random $r \xleftarrow{\$} [B]$ and outputs statement $\chi = (\text{pp}, \text{pk}, c_R)$ along with proof π_r .

In order to break soundness (χ, π_r) must satisfy the following properties:

1. $\text{Verify}(\chi, \pi_r) = 1$.
2. $\forall (k, u)$ it holds that $c_r.k^* \neq g_2^{x_r k}$ or $c_r.ct \neq (g_2^u, \text{pk}^u \cdot g_2^k)$.

We claim that whenever \mathcal{A} wins $\text{Game-Rogue}_{\mathcal{A}}$, there always exists at least one $r \in [B]$ such that both properties above are satisfied. Property 1 is necessarily satisfied for all $\ell \in [B]$ whenever \mathcal{A} wins $\text{Game-Rogue}_{\mathcal{A}}$. We argue further that Property 2 must be satisfied for at least one $r \in [B]$ whenever \mathcal{A} wins $\text{Game-Rogue}_{\mathcal{A}}$ because of the correctness of BTE. Assume that \mathcal{A} sets $c_r.k^*$ and $c_r.ct^*$ for all $\ell \in [B]$ such that Property 2 is *not* satisfied. This in turn means that the punctured keys and ElGamal ciphertext for all ciphertexts in the batch are correct (i.e. $\exists (k_\ell, u_\ell)$ such that $c_\ell.k^* = g_2^{x_\ell k_\ell}$ and $c_\ell.ct = (g_2^{u_\ell}, \text{pk}^{u_\ell} \cdot g_2^{k_\ell})$). Observe now that BatchDec only used the ElGamal ciphertext, so the resulting partial decryptions are correct for the batch and the resulting combined key is also correct $K = g_2^{\sum k_\ell}$. Further, when we look at the Combine step, we can see that the challenge message m_i is computed from the decryption shares as follows:

$$m_i = \gamma_i + \sum_{\ell \neq i}^{[B]} \text{PRF.PEval}(c_{\ell}.k^*, \ell) - \text{PRF.ExpEval}(K, i)$$

Clearly, this only relies on the punctured keys $c_{\ell}.k^*$ and the combined key K , which in turn relies on the ElGamal ciphertexts. Hence, the resulting message m_i is correct because of the perfect correctness of BTE. By contradiction, we have shown that there must exist at least one index $r \in [B]$, which

also satisfies Property 2. The reduction \mathcal{B} will pick this index with probability of at least $1/B_{\max}$, and we get

$$\Pr[\mathcal{B} \text{ breaks soundness}] \geq \frac{1}{B_{\max}} \cdot \Pr[\text{Game-Rogue}_{\mathcal{A}}(1^\lambda) = 1]$$

■

D.3 Probability Analysis of 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:

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

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

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:

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

According to Stirling's approximation, $(\alpha+1)! \geq \sqrt{2\pi} \sqrt{\alpha+1} \left(\frac{\alpha+1}{e}\right)^{\alpha+1}$. 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:

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

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

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

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

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

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

$$(10)$$

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.