# Weighted Batched Threshold Encryption with Applications to Mempool Privacy

Amit Agarwal[1], Kushal Babel[1], Sourav Das[1], Babak Poorebrahim Gilkalaye[1], Arup Mondal[2], Benny Pinkas[3], Peter Rindal[1], and Aayush Yadav[4]

[1]Category Labs
[2]Ashoka University
[3]Bar-Ilan University
[4]George Mason University

**Abstract**

A Batched Threshold Encryption (BTE) scheme enables a committee of servers to perform *lightweight* (in terms of communication and computation) threshold decryption of an arbitrary *batch* of ciphertexts from a larger pool, while ensuring the privacy of ciphertexts that are outside the batch. Such a primitive has a direct application in designing encrypted mempools for MEV protection in modern blockchains. Bormet et al. (USENIX 2025) recently proposed a BTE scheme called "BEAT-MEV" which is concretely efficient for small to moderate batch sizes.

In this work, we improve and extend the BEAT-MEV scheme in multiple ways. First, we improve the computational cost from quadratic to quasilinear in the batch size thus making it practical for *large* batch sizes. This improvement is achieved by substituting the key-homomorphic punctured PRF used in BEAT-MEV with an FFT-friendly alternative. Second, we extend the ideas in their scheme to the *weighted* setting, where each server in the committee has an associated 'weight' value (e.g. stake weight of validators in PoS blockchains), while crucially ensuring that the communication cost remains *independent* of the weights. In contrast, BEAT-MEV with naive virtualization would incur communication cost linear in the total weight. Third, to handle the small failure rate inherent in BEAT-MEV scheme due to index collisions across different clients at the time of encryption, we propose a generalization of their suggested approach which offers an option to tradeoff between ciphertext size and server communication cost.

We implement and evaluate our scheme and compare it with BEAT-MEV to demonstrate our concrete improvement. In the unweighted setting, we improve the computational cost (without increasing the communication cost) by $\approx 6\times$ for a batch size of 512 ciphertexts. In the weighted setting, we improve the communication cost (without compromising computation time), over BEAT-MEV with naive virtualization, by $\approx 50\times$ for 100 validators with total stake weight 5000 distributed as per the latest Solana stake distribution.

## 1 Introduction

A batched threshold encryption (BTE) scheme enables a group of servers to jointly decrypt (in a threshold fashion) any chosen subset/batch of ciphertexts from a large public pool, while keeping all remaining ciphertexts private [CGPP24, SAS24, AFP25, CGPW25, BFOQ25, BCF+25]. The defining feature of BTE is scalability: the total server-to-server communication required for decryption grows sublinearly with the batch size and ideally completely independently of it. This ability to selectively and efficiently decrypt *only* a batch of ciphertexts is useful in many contexts, such as:

- In *time-locked or event-conditioned releases*, servers can jointly decrypt only after a prescribed condition – such as a timestamp, or a price trigger – has been met [GMR23, ABDG25].

- In *fair multi-party computation*, outputs of many concurrent MPC sessions can be released fairly and simultaneously once a threshold of participants consent [CGJ+17, AFP25].
- In *privacy-preserving auctions and derivatives*, only winning bids or exercised options are revealed, keeping all others bids private [AN20, PPA+20].

**Mempools and MEV.** While these abstract uses illustrate BTE's generality, the most immediate and high-impact application today lies in *blockchain mempool privacy and MEV mitigation*. In existing blockchains, users broadcast transactions to a public queue (the mempool) before they are finalized in blocks. Because these transactions are visible, sophisticated actors and miners or validators who are in control of block production can frontrun, reorder, replicate or censor mempool transactions for profit, at the expense of ordinary users. This phenomenon is called "Miner or Maximal Extractable Value" (MEV) [DGK+20, BDKJ23]. MEV undermines decentralization [DGK+20], and has become one of the most persistent structural issues in decentralized finance, with hundreds of millions of USD extracted [QZG22] and massive lawsuits underway [New25].

Recent works have proposed to reduce MEV through *encrypted mempools* [BO22, KLJD23]. In encrypted mempools, users encrypt their transactions under a public key shared by the validator set. Once a block is finalized, the validators collectively decrypt only the transactions included in that block, while all unconfirmed transactions remain private. BTE naturally fits this setting: it enables threshold decryption of arbitrary subsets of ciphertexts (i.e., transactions included in each block) with communication that is independent of the batch size (i.e., the block size), while preserving semantic security for all pending transactions outside the batch (i.e., mempool transactions not included in the blocks).

Despite many recent works (see Section 1.2), the current state-of-the-art BTE scheme BEAT-MEV [BFOQ25] has several limitations that prevent it from being practically deployable.

- **High decryption time**: In BEAT-MEV, decrypting a batch of $B$ ciphertexts requires $B^2$ bilinear pairing operations, resulting in impractical decryption times for large batch sizes and orders of magnitude more than the block interval times of modern blockchains.
- **No (efficient) support for weights**: In a weighted setting, which is the natural setting for proof-of-stake blockchains, each server performing the threshold decryption is associated with a weight (e.g. its stake in the blockchain). BEAT-MEV natively does not support this setting and a naive extension, via virtualization technique (see Section 2.3), leads to an inter-server communication cost which grows linearly with the *total weight* across all the servers.
- **Collision issue:** Each ciphertext in BEAT-MEV must be formed w.r.t an index from a prespecified domain of size $n$, where $n$ controls the size of the setup phase. If multiple ciphertexts use the same index, then the subset can only include one of them (at a time) for batched decryption. This becomes a limitation in some applications, such as mempool privacy, where clients sending encrypted transactions need to ensure asynchronously that such an index collision doesn't occur. The proposed sub-batching solution in BEAT-MEV to reduce this collision probability either leads to a large increase in inter-server communication cost or a high "failure rate" (meaning excluding ciphertexts that have index collision from the subset) which can become prohibitive in some settings. We note that this particular limitation of index collision in an *application specific* limitation - it only affects those applications of BTE (e.g., mempool privacy) where the ciphertexts that will be a part of the subset are generated by entities that cannot co-ordinate, via other means such as smart contracts, to ensure index uniqueness.

## 1.1 Contributions

In this paper, we address these limitations, and make several advances toward realizing a practical, and weighted BTE scheme suitable for deployment in blockchain systems. More specifically, we start with the BEAT-MEV construction, and introduce the following changes:

- **Quasilinear decryption time via FFT-structured KH-PPRFs (Section 2.2).** The dominant computation cost during decryption in BEAT-MEV arises from summing $B^2$ evaluations of a key-homomorphic

puncturable PRF (KH-PPRF). As instantiated in BEAT-MEV, this overhead appears to be inherent. Therefore, we take an alternate approach and replace the KH-PPRF used in BEAT-MEV with a different KH-PPRF scheme from [DGM24a]. We observe that in the batched setting, the new KH-PPRF evaluations form a hidden convolution structure that can be computed efficiently using a Fast Fourier Transform (FFT). We believe that prior work likely missed this observation, as it requires opening up the KH-PPRF construction and doesn't apply generically to an arbitrary KH-PPRF. Using this FFT optimization, decrypting a batch of $B$ ciphertexts in our scheme requires $O(B \log B)$ time, instead of the $O(B^2)$ time required by BEAT-MEV.

- **Adding (efficient) support for weights (Section 2.3).** We present the first practical BTE scheme that supports server weights while ensuring that the inter-server communication cost is *independent* of the weights. Our construction algebraically couples a simple weighted but *unbatched* threshold encryption scheme with the KH-PPRF primitive where both primitives share correlated randomness. The resulting scheme requires a nuanced security proof due to the various interactions between these primitives.
- **Modern hashing for collision-resilience (Section 2.4).** To handle index collisions gracefully, we employ cuckoo hashing with bins capacity greater than one [DW07, MP23, EMM06], a rarely used variant which is particularly well suited for our BTE scheme. This allows us to handle large batches of ciphertexts while simultaneously ensuring a low inter-server communication and low failure rate.
- **Comprehensive evaluation (Section 5).** We implement our scheme along with optimizations and provide detailed benchmarks demonstrating concrete runtime and bandwidth improvements over two baselines: original BEAT-MEV scheme and a simple improved variant of it. For a batch of $B = 512$, $N = 100$ servers with equal weight, our scheme requires 6.8 seconds on a single thread, 15.7× faster the BEAT-MEV. Increasing the total weight to 500, we observe a 6× improvement in running time over BEAT-MEV with a total time of 17.6 seconds and communication cost of 11.2 kB which is 5× better than BEAT-MEV. We also observe that our scheme is trivial to parallelize.

We note that our optimizations and extensions retain all the desirable properties of the original BEAT-MEV scheme, such as a one-time setup phase, a single key generation protocol for a fixed set of servers, and batch-independent ciphertexts (see Section 1.2 for more details). In summary, we make non-trivial progress towards reducing the existing gap between theoretical BTE constructions and practical blockchain requirements. Our design achieves near-linear scalability, high collision resilience, and supports weighted servers, properties not previously realized in a single design.

## 1.2 Related Work

We succinctly summarize the works relevant to Batched Threshold Encryption (BTE) and classify them into the following three broad categories:

1. **Communication *inefficent* approaches:** This category includes approaches where we naively perform an individual threshold decryption operation, using for example the classical threshold Elgamal encryption scheme [ElG85] or its weighted variant [BO22], for each ciphertext in the batch. The obvious downside of these approaches is that it incurs a communication cost that scales linearly with the batch size $B$ which becomes quite prohibitive in practice for large batch sizes.
2. **Communication *efficient* but batch *dependent* ciphertexts based approaches:** These approaches [AFP25, CGPW25, CGPP24, DFL24, MGZ22, DHMW23, CDK+22, GMR23, CCN+23] manage to achieve a communication cost which is completely independent of the batch size $B$ but have a downside that each ciphertext is tied to a *batch-specific* label. In other words, each batch has a label $\ell$ (for example, this label can be the block number in a typical blockchain). The encryption algorithm takes $\ell$ as an input and produces a ciphertext $\mathsf{ct}_\ell$ that can only be decrypted when batch $\ell$ is processed for decryption. Suppose $P_\ell$ is the pool of ciphertexts w.r.t batch label $\ell$ out of which an arbitrary subset $S \subseteq P_\ell$ of ciphertexts need to be decrypted. This category of solutions either break the privacy of ciphertexts outside the set $S$ [DFL24, MGZ22, DHMW23, CDK+22, GMR23, CCN+23] or have a restriction that the ciphertexts

outside the set $S$ cannot become a part of any other batch decryption subsets [AFP25,CGPW25,CGPP24]. Both these caveats limit their applications, especially in the mempool privacy application where it is essential that encrypted transactions that don't get included in a particular block maintain their privacy and remain potential candidates for future block inclusion.

3. **Communication *efficient* and batch *independent* ciphertexts based approaches:** These approaches [BFOQ25, BCF+25, BLT25] manage to achieve a communication cost which is completely independent of the batch size $B$ while having ciphertexts that are not tied to a batch label. In other words, the encryption algorithm does not require any batch label $\ell$ as an input and produces a ciphertext $\mathsf{ct}$. The ciphertext $\mathsf{ct}$ can be a part of any future batch and the scheme ensures that the privacy of $\mathsf{ct}$ is maintained until it becomes part of some batch. From a practical perspective, this category of solutions have the properties that are necessary for typical applications and our work falls in this category. The state-of-the-art in this regime, in terms of efficiency metrics (such as decryption time, ciphertext size and communication cost), is BEAT-MEV [BFOQ25] (this is starting point for our work).

Finally, none of the existing works, to the best of our knowledge, have considered the question of constructing an efficient *weighted* version of Batched Threshold Encryption, which we address in our work.

# 2 Technical Overview

In this section, we provide a comphresenvie overview of our contributions. To describe our ideas, we will use a pairing-friendly group with a base group $\mathbb{G}$ and a target group $\mathbb{G}_{\mathsf{T}}$[1] of prime order $p$ with group operation $+$ and a pairing operation $\circ : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_{\mathsf{T}}$. For the base group $\mathbb{G}$ with a generator $g$, we will use the notation $[\![x]\!]$ to represent the group element $x \cdot g$ in the group $\mathbb{G}$ where $x \in \mathbb{Z}_p$ and similarly $[\![x]\!]_{\mathsf{T}}$ for target group representation.

## 2.1 BEAT-MEV as a starting point

We start by recalling the (un-weighted) BTE scheme called BEAT-MEV [BFOQ25], which can be seen as a generic template that makes black-box use of two main ingredients:

- A threshold homomorphic public-key encryption scheme ($\mathsf{THE}$): an extension of a standard public-key encryption scheme where (1) the decryption process is distributed among $N$ servers (with security against $t$ colluding servers); and (2) two ciphertexts can be combined to yield a new ciphertext which encrypts the sum (over the group of the message space) of the underlying messages.

- A key-homomorphic puncturable pseudorandom function ($\mathsf{KH\text{-}PPRF}$): an extension of the standard notion of a PRF with (1) key-homomorphism, meaning that for any input $x$ and any two keys $K_1, K_2$, the sum of the outputs computed with the keys $K_1$ and $K_2$ is equal to the output with the key $K_1 + K_2$. Formally, we have $\mathsf{PRF.Eval}(K_1, x) + \mathsf{PRF.Eval}(K_2, x) = \mathsf{PRF.Eval}(K_1 + K_2, x)$, where $\mathsf{Eval}$ is the PRF evaluation function; and (2) puncturing, meaning that for any key $K$ and input $i$, it is possible to derive a "punctured" key $K_i^*$ which enables PRF evaluation on all inputs *except* for the punctured point $i$. Formally, we have $\mathsf{PRF.Eval}(K, x) = \mathsf{PRF.PEval}(K_i^*, x)$ for all $x \neq i$ where $\mathsf{PEval}$ is the punctured evaluation function.

To encrypt a message $\mathsf{m}_i \in \{0,1\}^*$ in BEAT-MEV, the $i^{\text{th}}$ client samples a $\mathsf{PRF}$ key $K_i \in \mathbb{G}$, punctures the key $K_i$ at point $i$ to derive $K_i^* := \mathsf{PRF.Puncture}(K_i, i)$ and constructs the following ciphertext triple[2]

$$\mathsf{ct}_i := \Big( \underbrace{\mathsf{m}_i \oplus \mathsf{H}(\mathsf{PRF.Eval}(K_i, i))}_{\mathsf{ct}_i^{\mathsf{first}}}, \underbrace{\mathsf{THE.Enc}(K_i)}_{\mathsf{ct}_i^{\mathsf{second}}}, K_i^* \Big) \tag{1}$$

[1] Our scheme works over both symmetric and asymmetric pairings. Here we use symmetric pairing for ease of illustration and refer the readers to Fig. 2 for the formal construction over arbitrary pairing group.

[2] It is additionally assumed that the key space of the $\mathsf{PRF}$ and the message space of the $\mathsf{THE}$ are both some group $\mathbb{G}$. Additionally, the domain of $\mathsf{PRF}$ is $[B]$ where $B$ denotes the batch size.

The ciphertext includes a one-time-pad encryption $\mathsf{ct}_i^{\mathsf{first}}$ of the message $m_i$ with the hash of PRF evaluation w.r.t. key $K_i$ at the punctured point $i$, a THE ciphertext of $K_i$, and the punctured key $K_i^*$.

Once the committee of servers $\{S_i\}_{i\in[t+1]}$ decides to threshold-decrypt a batch of $B$ ciphertexts $\{\mathsf{ct}_i\}_{i\in[B]}$, they proceed as follows. First, they combine all the $\mathsf{THE}$ ciphertext components of $\{\mathsf{ct}_i\}_{i\in[B]}$, by leveraging the homomorphic property of $\mathsf{THE}$, to derive $\mathsf{ct}' := \sum_i \mathsf{ct}_i^{\mathsf{second}} = \mathsf{THE.Enc}(K_1 + \ldots + K_B)$ which is the encrypted aggregate key. They perform a threshold decryption of $\mathsf{ct}'$ and retrieve the aggregate key $K = \sum_i K_i$.

Then, each server can locally decrypt all the messages $\mathsf{m}_1, \ldots, \mathsf{m}_B$ in the following way:

$$\mathsf{m}_i \coloneqq \mathsf{ct}_i^{\mathsf{first}} \oplus \mathsf{H}\Big(\underbrace{\mathsf{PRF.Eval}(K, i) - \sum_{j\in[B]\setminus\{i\}} \mathsf{PRF.PEval}(K_j^*, i)}_{z_i}\Big) \tag{2}$$

Correctness follows from $K = \sum_{j\in[B]} K_j$, and that

$$\mathsf{PRF.Eval}(K, i) = \sum_{j\in[B]} \mathsf{PRF.Eval}(K_j, i) \tag{3}$$

$$= \mathsf{PRF.Eval}(K_i, i) + \sum_{j\in[B]\setminus\{i\}} \mathsf{PRF.PEval}(K_j^*, i) \tag{4}$$

where Eq. (3) is from the key-homomorphism property and Eq. (4) is from the puncturing property. Therefore, the $z_i$ query to $\mathsf{H}$ in Eq. (2) matches the $\mathsf{H}$ query used during encryption as shown in Eq. (1), i.e.

$$\mathsf{PRF.Eval}(K, i) - \sum_{j\in[B]\setminus\{i\}} \mathsf{PRF.PEval}(K_j^*, i) = \mathsf{PRF.Eval}(K_i, i)$$

Intuitively, the aggregated key $K$ is used to "patch" the punctured point which in turn allows one to decrypt the ciphertext. For the same reason, ciphertexts outside the batch remain fully encrypted as $K$ is not a sum of the corresponding PRF key and therefore can not be used to patch it's punctured key.

## 2.2 Lowering computational cost of decryption

The major computational bottleneck in BEAT-MEV comes from the aforementioned decryption process which, for a batch of $B$ ciphertexts, involves a total of $B^2$ many PRF evaluations, as each $z_i$ term in Eq. (2) requires $B$ PRF evaluations and there are $B$ such $z_i$ terms that need to be computed, one for each message in the batch. Note that all PRF evaluations that happen while computing the term $z_i$ are with respect to the same point $i$; hence the PRF evaluations for decrypting the $i^{\text{th}}$ message $\mathsf{m}_i$ cannot be "re-used" for decrypting the $j^{\text{th}}$ message $\mathsf{m}_j$ for $j \neq i$.

Furthermore, the most efficient known construction of $\mathsf{KH\text{-}PPRF}$ is the pairing-based scheme from [DGM24b] where the authors present two variants, one based on the Decisional Bilinear Diffie-Hellman assumption (used to instantiate BEAT-MEV) and another based on the Decisional $B$-Power Diffie-Hellman assumption[3]. Notably, the PRF evaluation in both these constructions requires a pairing operation, thus reducing the number of PRF evaluations during decryption is of special interest. Concretely, for a modest batch size $B = 512$ and assuming a pairing operation takes $\approx 0.45$ms for the BLS12-381 pairing friendly curve, this leads to a running time of $\approx 2$ minutes! The first driving motivation of this work is to address this bottleneck.

**Warm-up optimization.** As a warmup, we first note that this cost can be improved by using the well-known multi-pairing optimization[4] in the following way. Note that the $z_i$ term in Eq. (2) requires computing $B$ PRF evaluations and then adding them up. Each such PRF evaluation is done via pairing operation of two publicly known group elements. Hence, instead of computing $B$ pairings individually and then adding them, we can perform a single multi-pairing operation of size $B$. With this simple optimization, the cost

[3]The latter variant has a smaller CRS size than the former variant but this difference is irrelevant to our present discussion.

[4]Given two sets of $B$ group elements, $(a_1, \ldots, a_B)$ and $(b_1, \ldots, b_B)$, a multi-pairing operation computes $\sum_{i\in[n]} a_i \circ b_i$ more efficiently. It leverages a homomorphism that allows $\sum_i f(m(a_i, b_i)) = f(\sum_i m(a_i, b_i))$, where $a \circ b = f(m(a, b))$ is the pairing function and $f$ is the expensive "final exponentiation" step and $m$ is the "Miller loop."

reduces from $\approx 2$ minutes to $\approx 44$s for $B = 512$. While this is already a $3\times$ improvement, this cost is still far beyond the requirement of modern blockchain systems which operate on the order of few seconds or even less. For instance, Ethereum produces a block every 12 seconds.

**Leveraging FFT.** We observe that the decryption process can be made even more efficient by using an alternate key-homomorphic PRF construction from [DGM24b]. For a PRF domain $[B]$, this variant has a CRS which resembles the well-known "powers-of-tau" setup albeit with a missing element:

$$\mathsf{crs} := \underbrace{[\![\tau^1]\!]}_{h_1}, \ldots, [\![\tau^B]\!], \cancel{[\![\tau^{B+1}]\!]}, [\![\tau^{B+2}]\!], \ldots, \underbrace{[\![\tau^{2B+1}]\!]}_{h_{2B+1}} \tag{5}$$

To sample a PRF key and create its punctured version w.r.t some point $j \in [B]$, we sample $k_j \leftarrow\!\!\$\ \mathbb{Z}_p$, set $K_j := [\![k_j]\!]$ as the key and $K_j^* := k_j \cdot [\![\tau^j]\!]$ as the punctured key. PRF evaluation $\mathsf{Eval}$ (resp. punctured evaluation $\mathsf{PEval}$) is done by pairing the key (resp. punctured key) with an appropriate element in the CRS as specified below.

$$\mathsf{PRF.Eval}(K_j, i) = K_j \circ h_{B+1+i} \;//\; = [\![k_j \tau^{B+1+i}]\!] \; \forall i \in [B]$$
$$\mathsf{PRF.PEval}(K_j^*, i) = K_j^* \circ h_{B+1+i-j} \;//\; = [\![k_j \tau^{B+1+i}]\!], i \neq j$$

Plugging the above evaluation expression into the expression for $z_i$ term in Eq. (2), we get the following:

$$\begin{aligned} z_i &= \mathsf{PRF.Eval}(K, i) - \sum_{j \in [B] \setminus \{i\}} \mathsf{PRF.PEval}(K_j^*, i) \\ &= K \circ h_{B+1+i} - \sum_{j \in [B] \setminus \{i\}} K_j^* \circ h_{B+1+i-j} = \sum_j \mathbf{A}_j \circ \mathbf{B}_{B+2+i-j} \end{aligned}$$

where $\mathbf{A} := (K, -K_1^*, \ldots, -K_B^*)$ is the aggregate and punctured keys, $\mathbf{B} := (h_1, \ldots, h_B, [\![0]\!], h_{B+2}, \ldots, h_{2B+1})$ is the $\mathsf{crs}$ (Eq. (5)) with $[\![0]\!]$ inserted for the missing $\cancel{h_{B+1}}$.

Now, our main observation is that $z_i$ above is a term in the polynomial convolution $\mathbf{C} := \mathbf{A} \star \mathbf{B}$, where $\mathbf{A}, \mathbf{B}$ are the coefficient vectors[5], i.e., $\mathbf{C}_i = \sum_j \mathbf{A}_j \circ \mathbf{B}_{i-j+1}$. In particular, $z_i$ is the $(B+i)$-th coefficient of the product polynomial, i.e., $z = (\mathbf{C}_{B+1}, ..., \mathbf{C}_{2B})$. Note that $\mathbf{B}_{B+1} = [\![0]\!]$ critically ensures that the convolution omits the contributions from evaluating each punctured key at its puncture point. This convolution can be computed efficiently using FFT "in the exponent." More concretely, the parties compute $\mathbf{C} := \mathsf{iFFT}(\mathsf{FFT}(\mathbf{A}) \odot \mathsf{FFT}(\mathbf{B}))$ where we use $\odot$ to refer to component-wise pairing in the "evaluation domain.'

The upshot is that we can use FFT to compute all the coefficients of the product polynomial $\mathbf{C}$ using $O(B \log B)$ group exponentiations and $O(B)$ pairings. Additionally, we observe that the coefficients $\mathbf{B}$ only contain elements of the CRS that are independent of the batch. Hence, a part of the FFT based convolution, where we transform $\mathbf{B}' := \mathsf{FFT}(\mathbf{B})$ from coefficient form to evaluation form, can be precomputed once and reused across all the batches. With our FFT optimization, the concrete computational cost of batched decryption reduces to just $\approx 6$s for a batch size $B = 512$. This is a $20\times$ improvement over the original BEAT-MEV approach and $7\times$ improvement over the warm up multi-pairing optimization described earlier.

One can further optimize this computation in several ways. First, the FFT of $\mathbf{A}$ can be computed before $K = \mathbf{A}_1$ is threshold decrypted. Its contribution can be added later as the constant term, i.e. add $K$ to each coefficient. This allows the FFT to be computed concurrently with the threshold decryption, reducing the latency. In addition, many low level FFT optimizations exist, such as using higher radix FFT [Goo58, CT65] and pruned FFT [DH84, SB02]. We leave the application of them as future work. Hereafter, we will refer to our FFT optimized scheme, where we leverage the alternative $\mathsf{PRF}$ scheme in [DGM24b] and exploit it in a non-black-box way to do batch threshold decryption in quasilinear (in the batch size $B$) time, as $\mathsf{BEAT\text{-}MEV}^{++}$.

[5] I.e. interpret $\mathbf{A}, \mathbf{B}$ as polynomials $a(Y) := \sum_i \mathbf{A}_i Y^{i-1}, b(Y) := \sum_i \mathbf{B}_i Y^{i-1}$, respectively. Then, $\mathbf{C}$ consists of coefficients of the product polynomial $c := ab$ with coefficient multiplication defined as pairing.

## 2.3 Extending to the weighted setting

In a weighted BTE scheme, each server $j \in [N]$ performing the threshold decryption has a weight, say $w_j \in \mathbb{N}$. Security and robustness must hold against any colluding subset $S \subseteq [N]$ whose combined weight $\sum_{j \in S} w_j$ is up to threshold $t$.

**Virtualization and its limitations.** A simple way to convert an unweighted threshold encryption scheme into a weighted one is via *virtualization*: each real server $j$ with weight $w_j$ is treated as $w_j$ "virtual" servers of weight 1. The unweighted scheme is then executed on this expanded set. Clearly, this approach incurs communication costs proportional to the total weight $W = \sum_j w_j$, which can be prohibitively large in practice (e.g., $10 - 50\times$ of the total server count even after quantizing the weights). Our goal is to design a weighted BTE scheme whose server communication cost is *independent* of the weights.

**Background: threshold ElGamal encryption and Ferveo.** In the threshold ElGamal encryption scheme, a secret key $\mathsf{msk} \leftarrow\!\$\ \mathbb{Z}_p$ is threshold secret-shared using the Shamir scheme among $N$ servers with corruption threshold $t$, where the $j$-th server receives its secret key share $\mathsf{msk}_j$. The public key is defined as $\mathsf{mpk} := [\![\mathsf{msk}]\!]$. To encrypt a message $\mathsf{m} \in \mathbb{G}$, sample $r \leftarrow\!\$\ \mathbb{Z}_p$ and set the ciphertext as

$$\mathsf{ct} := (\mathsf{ct}_1, \mathsf{ct}_2) := ([\![r]\!], \mathsf{m} + r \cdot \mathsf{mpk}) \in (\mathbb{G} \times \mathbb{G})$$

To decrypt $\mathsf{ct}$, each server $j$ reveals its decryption share $\mathsf{msk}_j \cdot \mathsf{ct}_1$. Given $t+1$ decryption shares, say $\{\mathsf{msk}_j \cdot \mathsf{ct}_j\}_{j \in [t+1]}$, they can be linearly combined via Lagrange interpolation, using Lagrange coefficients $\lambda_j$, to derive

$$\sum_j \lambda_j \cdot \mathsf{msk}_j \cdot \mathsf{ct}_1 = \mathsf{msk} \cdot \mathsf{ct}_1 = r \cdot \mathsf{mpk}.$$

The original message is then obtained as $\mathsf{m} = \mathsf{ct}_2 - \mathsf{msk} \cdot \mathsf{ct}_1$.

Ferveo [BO22] proposes a modification of this scheme to support weighted servers by operating over a pairing-friendly group. At a high level, Ferveo reduces the partial decryption size in the naive virtualization-based approach to a single group element per server by introducing a server-specific masking secret. To illustrate, consider $N = 2$ servers with weights $w_1 = 2$ and $w_2 = 3$. During setup, we sample secret keys $\mathsf{sk}_1, \mathsf{sk}_2 \leftarrow\!\$\ \mathbb{Z}_p$ and send them to servers 1 and 2, respectively. The master public key $\mathsf{mpk}$ is published in the target group, along with additional elements in the CRS:

$$\mathsf{crs} := \begin{cases} \mathsf{mpk} := [\![\mathsf{msk}]\!]_{\mathsf{T}} \\ [\![\mathsf{sk}_1 \cdot \mathsf{msk}_1]\!], [\![\mathsf{sk}_1 \cdot \mathsf{msk}_2]\!] \\ [\![\mathsf{sk}_2 \cdot \mathsf{msk}_3]\!], [\![\mathsf{sk}_2 \cdot \mathsf{msk}_4]\!], [\![\mathsf{sk}_2 \cdot \mathsf{msk}_5]\!] \end{cases}$$

where $\mathsf{msk}_j$ is the $j^{\text{th}}$ Shamir share of $\mathsf{msk}$ w.r.t $W = w_1 + w_2 = 5$ virtual servers. To encrypt a message $\mathsf{m} \in \mathbb{G}_{\mathsf{T}}$, sample $r \leftarrow\!\$\ \mathbb{Z}_p$ and set the ciphertext as

$$\mathsf{ct} := (\mathsf{ct}_1, \mathsf{ct}_2) := ([\![r]\!], \mathsf{m} + r \cdot \mathsf{mpk}) \in (\mathbb{G} \times \mathbb{G}_{\mathsf{T}})$$

To decrypt $\mathsf{ct}$, unlike the virtualization based approach, each server $j$ reveals a *single* group element $\mathsf{sk}_i^{-1} \cdot \mathsf{ct}_1 \in \mathbb{G}$ as its decryption share. Anyone can then convert these partial decryption shares into partial decryption shares in the virtualization-based approach by pairing them with appropriate elements in the $\mathsf{crs}$. In our example, given $\mathsf{sk}_1^{-1} \cdot \mathsf{ct}_1$, the partial decryption share of server 1, we pair it with $[\![\mathsf{sk}_1 \cdot \mathsf{msk}_1]\!]$ and $[\![\mathsf{sk}_1 \cdot \mathsf{msk}_2]\!]$ to derive $\mathsf{msk}_1 \cdot \mathsf{ct}_1$, and $\mathsf{msk}_2 \cdot \mathsf{ct}_2 \in \mathbb{G}_{\mathsf{T}}$, respectively (and similarly for server 2). It is possible to then linearly combine these derived shares via Lagrange interpolation to derive $\mathsf{msk} \cdot \mathsf{ct}_1 = r \cdot \mathsf{mpk} \in \mathbb{G}_{\mathsf{T}}$, which can then be used to retrieve the original $\mathsf{m}$. Note that this construction preserves the homomorphic property of $\mathsf{THE}$ w.r.t messages in the target group (instead of the source group as in the classical ElGamal scheme).

**Weighted BEAT-MEV$^{++}$.** A seemingly straightforward way to design a weighted version of BEAT-MEV$^{++}$ is to simply instantiate the thresholdizable homomorphic public-key encryption scheme $\mathsf{THE}$ used in BEAT-MEV$^{++}$ with a *weighted* $\mathsf{THE}$ scheme, say Ferveo [BO22]. While this approach makes sense, there is a subtle type mismatch that needs to be dealt with when taking into account the instantiation the key-homomorphic puncturable PRF.

Recall that in $\mathsf{BEAT\text{-}MEV}^{++}$, the PRF key resides in the *source* group and is encrypted using $\mathsf{THE}$ (see Eq. (1)). Consequently, the message space of $\mathsf{THE}$ needs to be the *source* group – something that Ferveo does not satisfy. To fix this type mismatch, we algebraically couple the PRF evaluation in $\mathsf{BEAT\text{-}MEV}^{++}$ with the decryption protocol of Ferveo in a non black-box manner. Doing so requires us to also couple, in a sense that will become clear shortly, the CRS elements of the PRF of $\mathsf{BEAT\text{-}MEV}^{++}$ with the CRS elements of Ferveo.

Recall that in $\mathsf{BEAT\text{-}MEV}^{++}$, the ciphertext of the $i^{\text{th}}$ client is encrypted using $\mathsf{H}(\mathsf{PRF.Eval}(K_i, i))$ as the one-time-pad (Eq. (1)) where $K_i = [\![k_i]\!]$ for $k_i \leftarrow\$ \mathbb{Z}_p$ and $\mathsf{PRF.Eval}(K_i, i) = [\![k_i \cdot \tau^{B+1+i}]\!]_{\mathsf{T}}$. In our new scheme, we will redefine the PRF evaluation to be $\mathsf{PRF.Eval}(K_i, i) = [\![k_i \cdot \mathsf{msk} \cdot \tau^{B+1+i}]\!]_{\mathsf{T}}$ where $\mathsf{msk}$ is the secret key of Ferveo. Similarly, we will redefine the structure of the punctured key $K_i^*$ w.r.t. point $i \in [B]$ from $[\![k_i \cdot \tau^i]\!]$ to $[\![k_i \cdot \mathsf{msk} \cdot \tau^i]\!]$.

Given this change, we need to add support for two key aspects *without* leaking $\mathsf{msk}$: (1) enable $i$-th client to perform PRF evaluation and puncturing on its key $k_i$; and (2) enable the servers to perform a weighted threshold PRF evaluation, w.r.t. all points in the PRF domain, on the aggregate key $k_1 + \ldots + k_B$ with performance overhead that is independent of batch size and weights. To this end, we add some $\delta$ and $\gamma$ terms (highlighted in blue in the following equation), in the existing CRS of [DGM24b] PRF, to address the first and second challenge respectively. For ease of illustration, we will continue the example of $N = 2$ servers with weights $w_1 = 2$ and $w_2 = 3$.

$$\mathsf{crs} := \begin{cases} \underbrace{[\![\tau^1]\!]}_{h_1}, \ldots, \cancel{[\![\tau^{B+1}]\!]}, \ldots, \underbrace{[\![\tau^{2B+1}]\!]}_{h_{2B+1}} \\ \underbrace{h_1 \cdot \mathsf{msk}}_{\delta_1}, \ldots, \underbrace{h_B \cdot \mathsf{msk}}_{\delta_B} \\ \underbrace{\{h_{B+1+i} \cdot \mathsf{sk}_1 \cdot \mathsf{msk}_1\}_{i\in[B]}}_{\gamma_{i,1,1}}, \underbrace{\{h_{B+1+i} \cdot \mathsf{sk}_1 \cdot \mathsf{msk}_2\}_{i\in[B]}}_{\gamma_{i,1,2}} \\ \underbrace{\{h_{B+1+i} \cdot \mathsf{sk}_2 \cdot \mathsf{msk}_3\}_{i\in[B]}}_{\gamma_{i,2,3}}, \underbrace{\{h_{B+1+i} \cdot \mathsf{sk}_2 \cdot \mathsf{msk}_4\}_{i\in[B]}}_{\gamma_{i,2,4}} \\ \underbrace{\{h_{B+1+i} \cdot \mathsf{sk}_2 \cdot \mathsf{msk}_5\}_{i\in[B]}}_{\gamma_{i,2,5}} \end{cases}$$

Here, the newly added $\delta$ and $\gamma$ terms in the CRS correlates the "powers-of-tau" elements of the $\mathsf{PRF}$ with the terms depending on $\mathsf{sk}_i$'s and $\mathsf{msk}$ of Ferveo. We are now ready to describe our weighted BTE scheme. In our scheme, the $i^{\text{th}}$ client will sample a key $k_i \leftarrow\$ \mathbb{Z}_p$ and derive its punctured version w.r.t point $i \in [B]$ as

$$K_i^* := k_i \cdot \delta_i \qquad // = [\![k_i \cdot \mathsf{msk} \cdot \tau^i]\!]$$

Note that the client can evaluate the PRF at any $\ell \in [B]$ by selecting any $u \in [B] \setminus \{\ell\}$ and computing the following:

$$\mathsf{PRF.Eval}(k_i, \ell) = k_i \delta_u \circ h_{B+1+\ell-u} \; // = [\![k_i \cdot \mathsf{msk} \cdot \tau^{B+1+\ell}]\!]_{\mathsf{T}} \tag{6}$$

Moreover, the punctured key $K_i^*$ can be used to evaluate the PRF at all points $\ell \neq i$ as follows (this is not needed by the client but will be used on the server side during decryption process)

$$\mathsf{PRF.PEval}(K_i^*, \ell) = K_i^* \circ h_{B+1+\ell-i} \; // = [\![k_i \cdot \mathsf{msk} \cdot \tau^{B+1+\ell}]\!]_{\mathsf{T}} \tag{7}$$

To encrypt a message $\mathsf{m}_i$, a client computes the ciphertext triple $\mathsf{ct}_i$ as

$$\mathsf{ct}_i := \Big( \underbrace{\mathsf{m}_i \oplus \mathsf{H}(\mathsf{PRF.Eval}(k_i, i))}_{\mathsf{ct}_i^{\mathsf{first}}}, \underbrace{[\![k_i]\!]}_{K_i}, K_i^* \Big)$$

The ciphertext includes (i) a one-time-pad encryption of the message $\mathsf{m}_i$ using the hash of the PRF evaluation w.r.t. $k_i$ at the punctured point $i$, (ii) a group representation of $k_i$, and (iii) the corresponding punctured key $K_i^*$.

Once the committee decides to threshold-decrypt a batch of $B$ ciphertexts $\{\mathsf{ct}_i\}_{i\in[B]}$, the servers proceed as follows. First, each server locally computes the aggregate client key $K := K_1 + \ldots + K_B$. Then, each server $j$ outputs a *single* group element $\sigma_j := \mathsf{sk}_j^{-1} \cdot K$. Here, $\sigma_j$ is conceptually analogous to the "partial decryption share" in Ferveo but serves a different role: it enables servers to locally evaluate the PRF w.r.t aggregate key $K$.

In our example with $N = 2$ servers, each server receives $\sigma_1 = \mathsf{sk}_1^{-1} \cdot K$ and $\sigma_2 = \mathsf{sk}_2^{-1} \cdot K$. To evaluate the PRF w.r.t. $K$ at point $\ell \in [B]$, a server computes the following terms by leveraging the $\gamma$ terms from the $\mathsf{crs}$:

$$\begin{cases} \sigma_1 \circ \gamma_{\ell,1,1} \\ \sigma_1 \circ \gamma_{\ell,1,2} \\ \sigma_2 \circ \gamma_{\ell,2,3} \\ \sigma_2 \circ \gamma_{\ell,2,4} \\ \sigma_2 \circ \gamma_{\ell,2,5} \end{cases} \quad \textit{// equivalent to} \quad \begin{cases} [\![k \cdot \tau^{B+1+\ell} \cdot \mathsf{msk}_1]\!]_{\mathsf{T}} \\ [\![k \cdot \tau^{B+1+\ell} \cdot \mathsf{msk}_2]\!]_{\mathsf{T}} \\ [\![k \cdot \tau^{B+1+\ell} \cdot \mathsf{msk}_3]\!]_{\mathsf{T}} \\ [\![k \cdot \tau^{B+1+\ell} \cdot \mathsf{msk}_4]\!]_{\mathsf{T}} \\ [\![k \cdot \tau^{B+1+\ell} \cdot \mathsf{msk}_5]\!]_{\mathsf{T}} \end{cases}$$

Next, servers linearly combine these terms via Lagrange interpolation to derive $[\![k \cdot \mathsf{msk} \cdot \tau^{B+1+\ell}]\!]_{\mathsf{T}}$ which matches$\mathsf{PRF.Eval}(k, \ell)$. Note that regardless of the PRF evaluation point $\ell$, each server sends a *single* group element during the decryption protocol.

In summary, the ability to (i) compute the PRF on aggregate key $K$ at any point $\ell$, which we will succinctly denote as $\mathsf{PRF.Eval}(K, \ell)$, and (ii) evaluate any punctured key $K_i^*$ at the non-punctured points (Eq. (7)), lets each server locally decrypt all messages $\mathsf{m}_1, \ldots, \mathsf{m}_B$ in the same manner as in the original BEAT-MEV scheme (see Eq. (2)). This completes the description of our weighted BTE scheme.

**Alternative construction.** In Appendix B , we also describe an alternative construction for weighted BTE, which fixes the type mismatch that we pointed out earlier, in a different way. This alternative construction has a much larger ciphertext size currently hence we defer it to the appendix with the hope that future works can investigate and optimize it.

## 2.4 Handling collisions more efficiently

**Background: collision issue in BEAT-MEV.** In the BEAT-MEV template (Section 2.1), encryption takes as input an index $i \in [n]$, where $n$ is the domain size of the PRF, and forms a ciphertext $\mathsf{ct}_i$ as per Eq. (1). The default setting is to set $n = B_{\mathsf{max}}$ where $B_{\mathsf{max}}$ is the maximum batch size that the application of BTE demands. As noted in BEAT-MEV [BFOQ25], if two or more ciphertexts are created w.r.t. the same index $i$, then the batch decryption process can include at most one of them. In other words, these ciphertexts must be decrypted as part of two separate batches. This particular limitation of index collision is an *application specific* limitation – it only affects those applications of BTE, such as mempool privacy, where ciphertexts are generated by entities that cannot coordinate to ensure index uniqueness.

**Challenges in collision resolution.** In applications where index coordination is not possible, typically the best solution is to let the encryptors sample their indices uniformly at random from a domain of size $n$ and hope that collisions occur with low probability. Making $n$ too large is not feasible as the CRS size grows at least linearly with $n$. Alternatively, using a KH-PPRF that natively supports an exponential domain size using a moderate CRS size has practical efficiency issues, as these KH-PPRF schemes rely on lattices [DGM24b] and are computationally quite costly.

**Known approaches and its limitations.** To mitigate this collision issue, [DGM24b] uses a solution based on perfect matching, and BEAT-MEV [BFOQ25] uses a sub-batching solution. For large batch sizes, the former approach leads to large ciphertext sizes (as argued in [BFOQ25]) whereas the latter approach leads to a large increase in the server communication when ensuring collision probability $\epsilon \leq 2^{-40}$. We manage to simultaneously achieve a low ciphertext size and low server communication by generalizing these approaches via *cuckoo hashing* [PR01] over bins with capacity greater than 1.

**A cuckoo hashing based approach.** Cuckoo hashing is a data structure that maps $B$ items, $x_1, ..., x_B \in \mathcal{X}$, each into one of $n$ buckets.The hashing algorithm uses $\mathsf{c}$ random functions $h_1, ..., h_{\mathsf{c}} : \mathcal{X} \to [n]$, and each

item $x_i$ can be placed in one of the buckets indexed by $h_1(x_i), ..., h_{\mathsf{c}}(x_i) \in [n]$. Most instantiations in the literature assume at most one item can reside in any bucket. We will consider a well-known but rarely used generalization where at most $d$ items can reside in any given bucket [DW07, MP23, EMM06]. (For a thorough discussion of generalized cuckoo hashing, see, e.g., [Wie17].)

For BTE, when using cuckoo hashing with $\mathsf{c}$ hash functions and bucket capacity $d$, each message to be encrypted w.r.t $\mathsf{c}$ random indices in the range $[n]$ (which are interpreted as bucket indices in cuckoo hashing).[6] The parameter $d$ defines the *maximum* number of ciphertexts in a batch that can be assigned the same index. Given all the ciphertexts and their indices, the servers run a predetermined cuckoo hashing algorithm to map each ciphertext to one of its bucket indices (for example, using the perfect construction algorithms described in [Yeo23] which are guaranteed to find a mapping if it exists), such that no index has more than $d$ ciphertexts associated with it. Once this mapping is done, the servers derive $d$ many sub-batches where each sub-batch includes at most one ciphertext from each indexed bucket, and then run a BTE decryption protocol individually on each sub-batch. This has the following effect on BTE efficiency: ciphertext size is proportional to $\mathsf{c}$, server communication is proportional to $d$ and CRS size is proportional to $n$.

We observe that the known approaches of perfect matching in [DGM24b] and sub-batching in BEAT-MEV [BFOQ25] are simply extreme special cases of the above idea instantiated with parameters $(d = 1, \mathsf{c} > 1)$ and $(d > 1, \mathsf{c} = 1)$, respectively. It turns out that a much better tradeoff between different BTE efficiency metrics can be achieved when we simultaneously set $d > 1$ and $\mathsf{c} > 1$. We show, via numerical analysis in Section 5.2, that $\mathsf{c} = 2$ and $d = 8, 16$ (that same choices of $d$ as were proposed in BEAT-MEV) enables us to handle a much larger batch of ciphertexts, compared to BEAT-MEV, while ensuring less than $\epsilon = 2^{-40}$ failure probability and close to full capacity utilization of $n \times d$.

**Other extensions.** In the next version , we will discuss how to handle situations where adversarial clients can deliberately try to perform denial of service attacks via collisions (again a known but unresolved issue in BEAT-MEV). While the real-world feasibility and incentive of such an attack is debatable, and depends on the exact application, we propose individual clients to dynamically choose the number of hash functions to use for encryption. As the number of hash functions increases, the adversary's advantage becomes negligible, while fewer hash functions result in greater efficiency. These ideas can be formalized via the notion of *adversarially robust cuckoo hashing schemes* [Yeo23] and weighted bipartite matching.

# 3 Preliminaries

**Notation.** We use $\lambda$ to denote a computational security parameter, $[a, b]$ to represent the set of integers $\{a, a+1, \ldots, b\}$ and $[n] := [1, n]$, $x \leftarrow\!\!\$\ S$ to denote that $x$ is an element sampled uniformly at random from set $S$. We use bold-letters to indicate vectors and matrices. For a vector $\mathbf{v}$ of length $n$, we use the notation $v_i$ to indicate the $i^{\text{th}}$ element of $\mathbf{v}$ where $i \in [n]$. By $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$, we mean the class $\lambda^{O(1)}$ and $\frac{1}{\lambda^{\omega(1)}}$. For security parameter $\lambda$, we use PPT to denote probabilistic $\mathsf{poly}(\lambda)$-time Turing Machines with $\mathsf{poly}(\lambda)$-sized advice. Also, we use the notation $v \leftarrow \Pi\text{-}\star(u, \langle x_j \rangle)$ to denote an execution of a secure multi-party computation protocol among a fixed set of parties, where the $j^{\text{th}}$ party has a *private* input $x_j$ (demarcated withing angle brackets) and a common *public* input $u$, where each (honest) party produces a common public output $v$ (which can be $\perp$). We summarize the notations in Table 1

## 3.1 Bilinear Groups

We follow the notation used in [GKPW24, Section 3.1]. A bilinear group is a set of three groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathsf{T}})$ of prime order $p$, with a (non-degenerate) bilinear map or pairing, denoted as $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_{\mathsf{T}}$. The groups $(\mathbb{G}_1, \mathbb{G}_2)$ are referred to as the source groups, while $\mathbb{G}_{\mathsf{T}}$ is the target group. The groups $\mathbb{G}_1, \mathbb{G}_2$ have generators $g_1, g_2$, and we use the notation $[\![x]\!]_b$ to represent $x \cdot g_b$[7] (meaning the group operation applied to $g_b$ for $x$

[6] This can be done efficiently using hybrid encryption where the message is encrypted with a random key that is encrypted w.r.t $\mathsf{c}$ indices

[7] The term $x$ is usually called as "scalar" or "exponent" in the literature.

Table 1: Parameters relevant to the BTE scheme

| Notation | Meaning |
|---|---|
| $N, t$ | Number of servers and corruption threshold |
| $w_j, W$ | Weight of the $j$-th server and total weight |
| $n$ | Index space |
| $B, B_{\mathsf{max}}$ | Batch size and maximum batch size |
| $d$ | Number of sub-batches |

times) in the group $\mathbb{G}_b$, for $b \in \{1, 2, \mathsf{T}\}$, where $x \in \mathbb{Z}_p$. The generator of $\mathbb{G}_\mathsf{T}$ is $g_\mathsf{T} = e(g_1, g_2)$. The group operation is additive, and therefore $[\![x]\!]_b + [\![y]\!]_b = [\![x + y]\!]_b$.

We denote the pairing operation $e([\![x]\!]_1, [\![y]\!]_2)$ as $[\![x]\!]_1 \circ [\![y]\!]_2 = [\![y]\!]_2 \circ [\![x]\!]_1 = [\![x \cdot y]\!]_\mathsf{T}$. (This notation makes it easier to write expressions which compose pairings with linear operations.) The operation $\circ$ is commutative and can be applied to vectors of equal length. For example, given $\boldsymbol{u} = ([\![u_1]\!]_1, \ldots, [\![u_n]\!]_1) \in \mathbb{G}_1^n, \boldsymbol{v} = ([\![v_1]\!]_2, \ldots, [\![v_n]\!]_2) \in \mathbb{G}_2^n$, we have that $\mathbf{u}^T \circ \mathbf{v} = [\![u_1 v_1 + \cdots u_n v_n]\!]_\mathsf{T}$. It is further possible to use this notation for matrix-vector multiplication. If $\mathbf{A} \in \mathbb{G}_1^{n \times m}$ and $\mathbf{b} \in \mathbb{G}_2^m$, then $\mathbf{A} \circ \mathbf{b}$ is the vector in $\mathbb{G}_\mathsf{T}^n$ with the coordinates $(\mathbf{A}_1 \circ \mathbf{b}, \ldots, \mathbf{A}_n \circ \mathbf{b})$ where $\mathbf{A}_i$ is the $i^{\text{th}}$ row vector of the matrix $\mathbf{A}$.

## 3.2 Non-interactive Zero-knowledge Proofs

**Definition 3.1** (Non-interactive zero knowledge)**.** A non-interactive zero knowledge proof system for a relation $\mathcal{R}$ consists of the following polynomial time algorithms:

- $\mathsf{crs} \leftarrow \mathsf{Setup}(\lambda)$. The probabilistic setup algorithm takes as input the security parameter $\lambda$, and outputs a common reference string $\mathsf{crs}$.
- $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \chi, \omega)$. The prover algorithm takes as input a common reference string $\mathsf{crs}$, an instance $\chi \in \mathcal{L}_\mathcal{R}$, and a witness $\omega$ and outputs a proof $\pi$.
- $1/0 \leftarrow \mathsf{Verify}(\mathsf{crs}, \chi, \pi)$. The verification algorithm takes as input a common reference string $\mathsf{crs}$, an instance $\chi$, and a proof $\pi$, and outputs 1 (accept) or 0 (reject).

A NIZK proof system for relation $\mathcal{R}$ must further satisfy the following properties:

**Definition 3.2** (NIZK Completeness)**.** For every common reference string $\mathsf{crs} \leftarrow \mathsf{Setup}(\lambda)$, any instance $\chi \in \mathcal{L}_\mathcal{R}$ with corresponding witness $\omega$,

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, \chi, \pi) = 1 \ : \ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \chi, \omega)\right] = 1.$$

**Definition 3.3** (NIZK Soundness)**.** For every PPT adversary $\mathcal{A}$, we require the following probability to be negligible in $\lambda$,

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{crs}, \chi, \pi) = 1 \\ \nexists\ \omega \ \ (\chi, \omega) \in \mathcal{R} \end{array} \ : \ \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(\lambda) \\ (\chi, \pi) \leftarrow \mathcal{A}(\lambda, \mathsf{crs}) \end{array}\right]$$

**Definition 3.4** (Zero-knowledge)**.** A NIZK proof system is zero-knowledge if for all $(x, w) \in \mathcal{R}$, there exists a PPT simulator $\mathcal{S}$ such that for every PPT adversary $\mathcal{A}$, the following probability is negligible in $\lambda$

$$\left|\Pr\left[\begin{array}{l} \\ \mathcal{A}(\pi_b) = b \end{array} \ : \ \begin{array}{r} b \leftarrow\$ \{0,1\} \\ \mathsf{crs}_0 \leftarrow \mathsf{Setup}(\lambda), \mathsf{crs}_1 \leftarrow \mathcal{S}(\lambda) \\ (\chi, \omega) \leftarrow \mathcal{A}(\lambda, \mathsf{crs}_b) \\ \pi_0 \leftarrow \mathsf{Prove}(\mathsf{crs}_0, \chi, \omega) \\ \pi_1 \leftarrow \mathcal{S}(\mathsf{crs}_1, \chi) \end{array}\right] - \frac{1}{2}\right|$$

**Definition 3.5** (Simulation extractability)**.** A NIZK proof system is simulation extractable if for every PPT adversary $\mathcal{A}$, there exists a PPT extractor $\mathcal{E}_{\mathcal{A}}$ with access to $\mathcal{A}$'s internal state $\mathsf{st}$, such that the following probability is negligible in $\lambda$

$$\Pr\left[\begin{array}{l|r} \mathsf{Verify}(\mathsf{crs},\chi,\pi)=1 & \mathsf{crs}\leftarrow\mathsf{Setup}(\lambda) \\ \wedge\ (\chi,\omega)\notin\mathcal{R} & : \ (\chi,\pi)\leftarrow\mathcal{A}^{\mathcal{O}_{\mathsf{SimProve}}}(\mathsf{crs};\mathsf{st}) \\ \wedge\ (\chi,\pi)\notin\mathcal{Q} & \omega\leftarrow\mathcal{E}_{\mathcal{A}}(\mathsf{crs},\chi,\pi,\mathcal{Q},\mathsf{st}) \end{array}\right]$$

where the oracle $\mathcal{O}_{\mathsf{SimProve}}$ on input $\chi$, outputs a simulated proof $\pi\leftarrow\mathcal{S}(\mathsf{crs},\chi)$ and inserts $(\chi,\pi)$ into the set $\mathcal{Q}$.

**Batch Schnorr proofs for discrete logarithm equality.** Our BTE scheme relies on a NIZK protocol for verifying a batch of discrete logarithm equality. The instance $\chi$ consists of two group elements $[\![1]\!], [\![\alpha]\!]$, and two vectors $\{[\![x_1]\!], [\![x_2]\!], \ldots, [\![x_k]\!]\}$ and $\{[\![y_1]\!], [\![y_2]\!], \ldots, [\![y_k]\!]\}$. The prover $\mathcal{P}$ wants to convince a verifier $\mathcal{V}$ that for each $i \in [k]$, $y_i = \alpha \cdot x_i$. Note that for $k = 1$, this is the standard discrete logarithm equality instance [CP93].

Concretely, we use the protocol from [APB+04] with the following interface. Given an instance $\chi$ and witness $r$, the prover $\mathcal{P}$ uses $\mathsf{DLEq.Prove}(\chi, r)$ to generate the NIZK proof $\pi$. The verifier $\mathcal{V}$ then uses $\mathsf{DLEq.Verify}(\chi, \pi)$ and either accepts or rejects the proof.

## 3.3 Cryptographic Hardness Assumptions

**Assumption 3.1** ($q$-Strong Diffie-Hellman [BS04])**.** *Let* $\mathsf{GrpGen}$ *be an asymmetric bilinear group generator. The q-SDH assumption is hard if for all PPT adversary* $\mathcal{A}$ *and degree bound* $q \in \mathbb{N}$*, and for uniformly random* $x \leftarrow\$ \mathbb{Z}_p$*, the following advantage* $\mathsf{Adv}^{\mathsf{SDH}}_{\mathcal{B}}$ *is negligible in* $\lambda$*:*

$$\Pr\left[(c, [\![(x+c)^{-1}]\!]_1) \leftarrow \mathcal{A}\left(\{[\![x^i]\!]_1, [\![x^i]\!]_2\}_{i\in[0,q]}\right)\right]$$

*where* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathsf{T}}, [\![1]\!]_1, [\![1]\!]_2, [\![1]\!]_{\mathsf{T}}, p, \circ) \leftarrow \mathsf{GrpGen}(1^\lambda)$.

**Assumption 3.2** (Discrete Logarithm)**.** *Let* $\mathsf{GrpGen}$ *be an asymmetric bilinear group generator. The discrete-logarithm* ($\mathsf{DL}$) *problem is hard if for all PPT adversary* $\mathcal{A}$ *the following advantage is negligible in* $\lambda$*.*

$$\Pr\left[x \leftarrow \mathcal{A}\left([\![1]\!]_1, [\![1]\!]_2, [\![x]\!]_1, [\![x]\!]_2\right) \ : \ x \leftarrow\$ \mathbb{Z}_p\right] = \mathsf{Adv}^{\mathsf{DL}}_{\mathcal{B}}$$

*where* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathsf{T}}, [\![1]\!]_1, [\![1]\!]_2, [\![1]\!]_{\mathsf{T}}, p, \circ) \leftarrow \mathsf{GrpGen}(1^\lambda)$.

**Algebraic Group Model (AGM).** In the Algebraic Group Model (AGM), all algorithms are modeled as algebraic, i.e., whenever the algorithm outputs a group element $H$, the algorithm also outputs an representation of $H$ with respect to all the group elements it has seen so far. More formally,

**Definition 3.6** (Algebraic Algorithm)**.** Let $\mathsf{GrpGen}$ be an asymmetric bilinear group generator, and let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathsf{T}}, [\![1]\!]_1, [\![1]\!]_2, [\![1]\!]_{\mathsf{T}}, p, \circ) \leftarrow \mathsf{GrpGen}(1^\lambda)$. Let $\mathcal{A}_{\mathsf{Alg}}$ be a probabilistic algorithm run on initial inputs including the description of the bilinear group. During its execution $\mathcal{A}_{\mathsf{Alg}}$ receives further inputs including obliviously sampled group elements (which it cannot sample directly).

Let $\boldsymbol{G} = [G_1, \ldots, G_n] \in \mathbb{G}_b^n$ for $b \in \{0, 1, \mathsf{T}\}$ be a list of all group elements $\mathcal{A}_{\mathsf{Alg}}$ has been given so far. We call $\mathcal{A}_{\mathsf{Alg}}$ an *algebraic algorithm* if for any group element $H \in \mathbb{G}_b$ it outputs, $\mathcal{A}_{\mathsf{Alg}}$ also outputs a vector $\boldsymbol{a} = [a_1, \ldots, a_n] \in \mathbb{Z}_p^n$ such that $H = \sum_{i\in[n]} a_i \cdot G_i$. The coefficients $a_i$ are called the *representation* of $H$ with respect to $\boldsymbol{G}$.

**Assumption 3.3** (Decisional $n$-Power Diffie–Hellman ($n$-$\mathsf{DH}$))**.** *Let* $\mathsf{GrpGen}$ *be an asymmetric bilinear group generator. The asymmetric n-power Diffie–Hellman problem is hard for* $\mathsf{GrpGen}$ *if the following two distributions* $\mathcal{D}_0$ *and* $\mathcal{D}_1$ *are computationally indistinguishable:*

$$\mathcal{D}_0 = \left(\{[\![x^i]\!]_1\}_{i\in[n]}, \{[\![x^i]\!]_2\}_{i\in[2n+1]\setminus\{n+1\}}, [\![y]\!]_1, [\![x^{n+1}\cdot y]\!]_{\mathsf{T}}\right),$$

$$\mathcal{D}_1 = \left(\{[\![x^i]\!]_1\}_{i\in[n]}, \{[\![x^i]\!]_2\}_{i\in[2n+1]\setminus\{n+1\}}, [\![y]\!]_1, [\![z]\!]_{\mathsf{T}}\right),$$

*where* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathsf{T}}, [\![1]\!]_1, [\![1]\!]_2, [\![1]\!]_{\mathsf{T}}, p, \circ) \leftarrow \mathsf{GrpGen}(1^\lambda)$ *and* $(x, y, z) \leftarrow\$ \mathbb{Z}_p^*$.

**Assumption 3.4** (Decisional $n$-Power Diffie–Hellman+ ($n$-DH$^+$))**.** *Let* GrpGen *be an asymmetric bilinear group generator. The asymmetric $n$-power Diffie–Hellman problem is hard for* GrpGen *if the following two distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ are computationally indistinguishable:*

$$\mathcal{D}_0 = \Big(\{[\![x^i]\!]_1\}_{i\in[-n,n]}, \{[\![x^i]\!]_2\}_{i\in[3n+1]\setminus\{n+1\}}, [\![y]\!]_1, [\![x^{n+1}\cdot y]\!]_\mathsf{T}\Big),$$

$$\mathcal{D}_1 = \Big(\{[\![x^i]\!]_1\}_{i\in[-n,n]}, \{[\![x^i]\!]_2\}_{i\in[3n+1]\setminus\{n+1\}}, [\![y]\!]_1, [\![z]\!]_\mathsf{T}\Big),$$

*where* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, [\![1]\!]_1, [\![1]\!]_2, [\![1]\!]_\mathsf{T}, p, \circ) \leftarrow \mathsf{GrpGen}(1^\lambda)$, $(x, y, z) \leftarrow\!\!\$\ \mathbb{Z}_p^*$, *and* $[-n, n] = \{-n, -(n-1), \ldots, n\}$.

**Lemma 3.7.** *$n$-DH$^+$ is hard if and only if $n$-DH is hard.*

*Sketch.* Given an adversary $\mathcal{A}$ that breaks the $n$-DH$^+$ assumption, we give a reduction $\mathcal{B}$ that breaks the $\tilde{n}$-DH assumption with $n = \lfloor \tilde{n}/2 \rfloor$. Specifically, given the $\tilde{n}$-DH instance $(\{[\![\tilde{x}^i]\!]_1\}_{i\in[\tilde{n}]}, \{[\![\tilde{x}^i]\!]_2\}_{i\in[2\tilde{n}+1]\setminus\{\tilde{n}+1\}}, [\![y]\!]_1, [\![z]\!]_\mathsf{T})$, $\mathcal{B}$ constructs a $n$-DH$^+$ instance as per the map $\tilde{x}^{n+1+i} \mapsto x^i$ if $n$ is even and $\tilde{x}^{n+i} \mapsto x^i$ if $n$ is odd for each $i \in [-n, 3n+1]$. It is easy to check that this yields a valid $n$-DH$^+$ instance of the form $(\{[\![x^i]\!]_1\}_{i\in[-n,n]}, \{[\![x^i]\!]_2\}_{i\in[3n+1]\setminus\{n+1\}}$, $[\![y]\!]_1, [\![z]\!]_\mathsf{T})$ since (for example for odd $n$) $[\![x^{-n}]\!]_1 = [\![\tilde{x}^1]\!]_1, [\![x^{-n+1}]\!]_1 = [\![\tilde{x}^2]\!]_1, \ldots, [\![x^n]\!]_1 = [\![\tilde{x}^{2n+1}]\!]_1 = [\![\tilde{x}^{\tilde{n}}]\!]_1$ and $[\![1]\!]_2 = [\![\tilde{x}^{n+1}]\!]_2, [\![x^1]\!]_2 = [\![\tilde{x}^{n+2}]\!]_2, \ldots, [\![x^{3n+1}]\!]_2 = [\![\tilde{x}^{4n+2}]\!]_2 = [\![\tilde{x}^{2\tilde{n}+1}]\!]_2$ with $[\![x^{n+1}]\!]_2 = [\![\tilde{x}^{2n+2}]\!]_2 = [\![\tilde{x}^{\tilde{n}+1}]\!]_2$ missing. Finally, when $\mathcal{A}$ outputs a bit $b$, $\mathcal{B}$ outputs $b$ as its response. It follows that if $\mathcal{A}$ solves $n$-DH$^+$ with non-negligible probability, $\mathcal{B}$ solves the $n$-DH problem with the same probability. The reverse direction can similarly be shown by taking the inverse map $x^i \mapsto \tilde{x}^{n+i}$ for each $i \in [-n, 3n+1]$. □

# 4 Weighted Batched Threshold Encryption

## 4.1 Formal Syntax and Definition

In this section, we provide a generic syntax and security definition for the notion of Weighted Batched Threshold Encryption (wBTE). We follow the definition in [BFOQ25] and adapt it to the weighted setting with the changes highlighted. The un-weighted setting is a special case of the weighted setting where the weight of all the $N$ servers is 1. Following [BFOQ25], our definition and construction will be specified in the "coordinated" setting, where each ciphertext in the batch has a unique index $i^*$ that is known at the time of encryption. There are known techniques [BFOQ25] to remove this coordination which we describe in Section 2.4.

**Definition 4.1** (Weighted Batched Threshold Encryption)**.** A weighted batched threshold encryption scheme wBTE consists of the following PPT algorithms and protocols:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^{B_{\mathsf{max}}})$: The randomized setup algorithm takes as input the security parameter $\lambda$ and the maximum batch size $B_{\mathsf{max}}$ and outputs public parameters pp which is an implicit input to all subsequent interfaces. The pp also contains an explicit description of an index space $[n]$.
- $(\mathsf{mpk}, \{\mathsf{sk}_j\}_{j\in[N]}) \leftarrow \mathsf{KeyGen}(N, t, \{w_i\}_{i\in[N]})$: The randomized key-generation algorithm takes as input the total number of servers $N$, the corruption threshold $t$, a set of $N$ weights $\{w_i\}_{i\in[N]}$, and outputs a master public key mpk to all, and the $j$-th secret key $\mathsf{sk}_j$ to server $j$.
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{m}, i^*)$: The randomized encryption algorithm takes as input a master public key mpk, a message m, and a batch index $i^* \in [n]$, and outputs a ciphertext ct. We require that ct contains an explicit description of the index $i^*$.
- $\{\widetilde{\mathsf{m}}_\ell\}_{\ell\in[B]} \leftarrow \Pi\text{-}\mathsf{BDec}(\mathsf{mpk}, \{\mathsf{ct}_\ell\}_{\ell\in[B]}, \langle\mathsf{sk}_i\rangle)$: This is an interactive protocol among all $[N]$ servers for batch decryption, where the $i^{\text{th}}$ server has a private input $\mathsf{sk}_i$, and all servers have common input mpk and a batch of ciphertexts $\{\mathsf{ct}_\ell\}_{\ell\in[B]}$ where $B \le B_{\max}$. The protocol outputs the original decrypted messages $\{\mathsf{m}_\ell\}_{\ell\in[B]}$ where each $\mathsf{m}_\ell$ is either a valid message or $\perp$.

**Game** $\mathsf{Expt}^{(\mathsf{w})\mathsf{BTE\text{-}CCA}}_{\mathcal{A},b}(N, t, \{w_j\}_{j\in[N]}, \lambda, B_{\mathsf{max}})$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathcal{Q}_{\mathsf{BDec}} \leftarrow \emptyset$
$\mathcal{C} \leftarrow \mathcal{A}(\mathsf{pp})$
$(\mathsf{mpk}, \{\mathsf{sk}_j\}_{j\in[N]}) \leftarrow \mathsf{KeyGen}(N, t, \{w_j\}_{j\in[N]})$
$(\mathsf{m}_0, \mathsf{m}_1, i^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{BDec}}}(\mathsf{mpk}, \{\mathsf{sk}_j\}_{j\in\mathcal{C}})$
$\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{m}_b, i^*)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{BDec}}}(\mathsf{ct}^*)$
**if** $|\mathcal{C}| \le t \ \wedge \ \mathsf{ct}^* \notin \mathcal{Q}_{\mathsf{BDec}}$ :
  **return** $b'$
**return** 0

**Oracle** $\mathcal{O}_{\mathsf{BDec}}(\{\mathsf{ct}_i\}_{i\in[B]})$:

*// Π-BDec is run among all $[N]$ parties where $\mathcal{A}$ controls the parties in the set $\mathcal{C}$.*

Let $\{\mathsf{m}_i\}_{i\in[B]} \leftarrow \Pi\text{-}\mathsf{BDec}(\mathsf{mpk}, \{\mathsf{ct}_i\}_{i\in[B]}, \langle\mathsf{sk}_j\rangle)$
$\mathcal{Q}_{\mathsf{BDec}} \leftarrow \mathcal{Q}_{\mathsf{BDec}} \cup \{\mathsf{ct}_i\}_{i\in[B]}$
**return** $\{\mathsf{m}_i\}_{i\in[B]}$

**Game** $\mathsf{Expt}^{\mathsf{BTE\text{-}rob}}_{\mathcal{A}}(N, t, \{w_j\}_{j\in[N]}, \lambda, B_{\mathsf{max}})$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathcal{C} \leftarrow \mathcal{A}(\mathsf{pp})$
**if** $|\mathcal{C}| > t$ : **return** 0
$(\mathsf{mpk}, \{\mathsf{sk}_j\}_{j\in[N]}) \leftarrow \mathsf{KeyGen}(N, t, \{w_j\}_{j\in[N]})$
$(B, S, \{\mathsf{m}_i\}_{i\in[B]\setminus S}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{BDec}}}(\mathsf{mpk}, \{\mathsf{sk}_j\}_{j\in[N]})$
**if** $B > B_{\mathsf{max}} \vee S \not\subset [B]$ : **return** 0
**for** all $i \in [B] \setminus S$ :
  $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{m}_i, i)$
$\{\mathsf{ct}_i\}_{i\in S} \leftarrow \mathcal{A}(\{\mathsf{ct}_i\}_{i\in[B]\setminus S})$ s.t. index of output $\mathsf{ct}_i$'s is set as $i$.
$\{\widetilde{\mathsf{m}}_i\}_{i\in[B]} \leftarrow \mathcal{O}_{\mathsf{BDec}}(\{\mathsf{ct}_i\}_{i\in[B]})$
**if** $\exists\ i \in [B] \setminus S$ s.t. $\widetilde{\mathsf{m}}_i \neq \mathsf{m}_i$ :
  **return** 1
**return** 0

Figure 1: Security and robustness game $\mathsf{Expt}^{(\mathsf{w})\mathsf{BTE\text{-}CCA}}_{\mathcal{A},b}$ and $\mathsf{Expt}^{\mathsf{BTE\text{-}rob}}_{\mathcal{A}}$, respectively, for a BTE scheme.

We require the wBTE scheme to satisfy the *correctness*, IND-CCA *security*, and *robustness* properties. We formalize these properties in Defs. 4.2 to 4.4 and describe them next.

**Correctness.** Naturally, the *correctness* property ensures that correctly encrypted ciphertexts successfully decrypt to the original encrypted messages. Formally,

**Definition 4.2** (Correctness of wBTE)**.** A batched threshold encryption scheme wBTE is correct for all $N \in \mathbb{N}, w_j \in \mathbb{N}, t \le \sum_{j\in[N]} w_j, \lambda \in \mathbb{N}, B_{\mathsf{max}} \in \mathbb{N}, B \le B_{\mathsf{max}}$, all messages $(\mathsf{m}_1, \mathsf{m}_2, \ldots, \mathsf{m}_B) \in \mathcal{M}^B$, where $\mathcal{M}$ is the message space, the following probability is 1

$$\Pr\left[\ \forall\ i \in [B] \ :\ \widetilde{\mathsf{m}}_i = \mathsf{m}_i \ \middle|\ \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^{B_{\mathsf{max}}}) \\ \left(\begin{array}{c}\mathsf{mpk},\\ \{\mathsf{sk}_j\}_{j\in[N]}\end{array}\right) \leftarrow \mathsf{KeyGen}(N, t, \{w_j\}_{j\in[N]}) \\ \{\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{m}_i, i)\}_{i\in[B]} \\ \{\widetilde{\mathsf{m}}_i\}_{i\in[B]} \leftarrow \Pi\text{-}\mathsf{BDec}\left(\begin{array}{c}\mathsf{mpk}, \{\mathsf{ct}_i\}_{i\in[B]},\\ \langle\mathsf{sk}_j\rangle_{j\in[N]}\end{array}\right) \end{array}\right]$$

IND-CCA **Security.** The CCA security for wBTE ensures that an adversary cannot tell apart between encryption of two equal-length messages, even if it can invoke the batch decryption protocol (Π-BDec) on arbitrary batches of its choice not containing the challenge ciphertext. We assume the $\{\mathsf{ct}_i\}_{i\in[B]}$ inputs being decrypted are delivered to the honest parties via a broadcast channel, e.g. a blockchain[8]. Security should hold even when the adversary statically corrupts servers with combined weight up to $t$. More formally,

**Definition 4.3** (CCA security of wBTE)**.** Let $\mathsf{Expt}^{(\mathsf{w})\mathsf{BTE\text{-}CCA}}_{\mathcal{A},b}$ be the game as defined in Fig. 1. A batched threshold encryption scheme wBTE is CCA secure if for all $N \in \mathbb{N}, w_j \in \mathbb{N}, t < \sum_{j\in[N]} w_j, B_{\mathsf{max}} \in \mathbb{N}$, and all stateful PPT adversaries $\mathcal{A}$, we have

$$\left|\Pr\left[\mathsf{Expt}^{(\mathsf{w})\mathsf{BTE\text{-}CCA}}_{\mathcal{A},0}(N, t, \lambda, B_{\mathsf{max}}) \Rightarrow 1\right] - \Pr\left[\mathsf{Expt}^{(\mathsf{w})\mathsf{BTE\text{-}CCA}}_{\mathcal{A},1}(N, t, \lambda, B_{\mathsf{max}}) \Rightarrow 1\right]\right| = \mathsf{negl}(\lambda)$$

**Robustness.** Finally, the *robustness* property prevents an adversary $\mathcal{A}$, that can corrupt parties with combined weight up to $t$, from invoking the batch decryption protocol (Π-BDec) on malformed ciphertexts to disrupt decryption of correctly generated ciphertexts. This is a generalization of the rouge-ciphertext security property defined in [BFOQ25], where $\mathcal{A}$ can not corrupt any of the decrypting parties. Formally,

[8]This assumption is implicit in all prior BTE schemes.

**Definition 4.4** (Robustness of wBTE)**.** Let $\mathsf{Expt}^{\mathsf{BTE\text{-}rob}}_{\mathcal{A}}$ be the game as defined in Fig. 1. A batched threshold encryption scheme wBTE is robust if for all $N \in \mathbb{N}, t < \left\lfloor \sum_{j\in[N]} w_j/2 \right\rfloor, B_{\mathsf{max}} \in \mathbb{N}$, we have the following:

$$\Pr[\mathsf{Expt}^{\mathsf{BTE\text{-}rob}}_{\mathcal{A}}(N, t, \lambda, B_{\mathsf{max}}) \Rightarrow 1] = \mathsf{negl}(\lambda) \ .$$

*Remark* (Efficiency of wBTE). As we discuss in Section 1, we are interested in wBTE schemes where the total communication complexity of $\Pi$-BDec, across all $N$ servers for any fixed $N$, is $o(B{+}W)$ meaning that it scales sub-linearly in the batch size $B$ and the total weight $W = \sum_{j\in[N]} w_j$.

## 4.2 Weighted BTE Design

We present the detailed design of our weighted BTE scheme in Fig. 2. We describe our scheme using an arbitrary pairing-friendly curve $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T})$ where the base groups $\mathbb{G}_1$ and $\mathbb{G}_2$ need not be identical. As in prior BTE schemes, our batch decryption protocol $\Pi$-BDec requires *one* round of communication. Accordingly, we split $\Pi$-BDec into two algorithms $(\Pi\text{-}\mathsf{BDec}_1, \Pi\text{-}\mathsf{BDec}_2)$. To decrypt a batch of ciphertexts, each server first runs $\Pi\text{-}\mathsf{BDec}_1$ locally and broadcasts its output to the other servers. Next, upon receiving these messages from other servers, each server completes the decryption protocol by executing $\Pi\text{-}\mathsf{BDec}_2$. The local variables, such as $U$ (the set of valid ciphertexts) and mpk (the master public key), are shared across both procedures. Moreover, all procedures including $\mathsf{Enc}, \Pi\text{-}\mathsf{BDec}_1, \Pi\text{-}\mathsf{BDec}_2$ have (implicit) access to the pp (public parameters) generated by the Setup procedure.

To ensure CCA security (Theorem 4.5) and robustness (Def. 4.4), we use Schnorr-style non-interactive zero-knowledge proofs [Sch90] to enforce the validity of ciphertexts (generated by the Enc algorithm) and partial decryption messages (generated by the $\Pi\text{-}\mathsf{BDec}_1$ algorithm). We highlight these steps in blue in the protocol. Concretely, we instantiate them using standard proof of equality of discrete logarithms from [CP93], which we abstract as (DLEq.Prove, DLEq.Verify).

## 4.3 Theoretical Analysis

The correctness of our scheme (Def. 4.2) follows straightforwardly as described in Section 2.3. We split our theoretical analysis into two parts, efficiency and security. In Section 5, we also detail our empirical analysis of efficiency based on concrete implementation.

### 4.3.1 Efficiency

In Table 2, we outline the concrete costs of different parts of our weighted BTE scheme and compare it with two baseline approaches: (i) the original BEAT-MEV [BFOQ25] with virtualization to support weights (see Section 2.3); and (ii) an improved variant where we apply known optimizations from the literature. We measure sizes in terms of group and field elements, and measure computation cost in terms of the different group operations.

The main highlight from Table 2 is that the server download cost of our scheme is *independent* of the total weight $W$ whereas it depends linearly in the baseline schemes. Later, in Section 5, we show how this concretely impacts the download cost for different values of $W$. Also, as a side benefit, the ciphertext size $|\mathsf{ct}|$ and the Enc computation cost of our scheme are also lower than in the baseline schemes. We note that our improvements come at a cost of a CRS of size $O(BW)$, where $B$ is the batch size, whereas BEAT-MEV and its improved variant have a CRS size of $O(B^2 + W)$ and $O(B + W)$, respectively. We also show the concrete computation cost of the servers for batched threshold decryption in the different schemes. Since it is hard to compare these costs analytically, we provide a concrete comparison in Table 3.

### 4.3.2 Security

**Theorem 4.5** (CCA Security)**.** *Assuming the hardness of discrete logarithm (Assumption 3.2), q-strong Diffie-Hellman (Assumption 3.1), n-power Diffie-Hellman (Assumption 3.3), and the zero-knowledge property*

**Building-blocks:**

- $N$ is the number of servers, $t$ is the corruption threshold, $w_j$ is the weight of the $j^{\text{th}}$ server and $W := \sum_{i\in[N]} w_i$.
- $\lambda$ denotes the security parameter, $B_{\mathsf{max}}$ is the maximum batch size, $B$ is the batch size and $n$ is the index space.
- A hash function $\mathsf{H} : \mathbb{G} \to \{0,1\}^*$
- Shamir secret sharing scheme having algorithms $\mathsf{Share}$ and $\mathsf{Lagrange}$
- A NIZK system $\mathsf{DLEq} = (\mathsf{Prove}, \mathsf{Verify})$ to prove DL equality w.r.t. an arbitrary group (Refer to [CP93]).

---

$\underline{\mathsf{Setup}(1^\lambda, 1^{B_{\max}})}$:

*// Defining the size of index space $n$*

1: $n := B_{\mathsf{max}}$
2: $\mathcal{G} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, [\![1]\!]_1, [\![1]\!]_2, [\![1]\!]_\mathsf{T}, p, \circ) \leftarrow \mathsf{GrpGen}(1^\lambda)$
3: $\tau \leftarrow\$ \mathbb{Z}_p$
4: **return** $\mathsf{pp} := n, \mathcal{G}, \{\underbrace{[\![\tau^i]\!]_1}_{g_i}\}_{i\in[n]}, \{\underbrace{[\![\tau^i]\!]_2}_{h_i}\}_{i\in[2n+1]\setminus\{n+1\}}$

$\underline{\mathsf{KeyGen}(1^\lambda, \{w_j\}_{j\in[N]}, t, \mathsf{pp})}$:

*// $\Omega$ is the secret sharing evaluation domain*

1: $W := \Sigma_{j\in[N]} w_j$; sample $\Omega \subseteq \mathbb{Z}_p$ s.t. $|\Omega| = W$
2: Partition $\Omega$ into $\Omega_1, \ldots, \Omega_N$ s.t. $|\Omega_j| = w_j \ \forall j \in [N]$
3: $\mathsf{msk} \leftarrow\$ \mathbb{Z}_p$
4: $\{\mathsf{msk}_\omega\}_{\omega\in\Omega} \leftarrow \mathsf{Share}(\mathsf{msk}, W, t, \Omega)$
5: $\{\mathsf{sk}_j\}_{j\in[N]} \leftarrow\$ \mathbb{Z}_p^N$
6: $\mathsf{mpk} := \begin{cases} \{\underbrace{h_{n+1+i} \cdot \mathsf{sk}_j \cdot \mathsf{msk}_\omega}_{\gamma_{i,j,\omega}}\}_{i\in[n], j\in[N], \omega\in\Omega_j} \\ \{\underbrace{g_i \cdot \mathsf{msk}}_{\delta_i}\}_{i\in[n]}, \{[\![\mathsf{sk}_j^{-1}]\!]_1\}_{j\in[N]} \end{cases}$
7: **return** $\mathsf{mpk}, \{\mathsf{sk}_j\}_{j\in[N]}$

$\underline{\mathsf{Enc}(\mathsf{mpk}, \mathsf{m}, i^*)}$:

1: $r \leftarrow\$ \mathbb{Z}_p$
2: Let $u \in [n] \setminus \{i^*\}$
3: $\mathsf{ct}_1 := [\![r]\!]_1$
4: $\mathsf{ct}_2 := r \cdot \delta_{i^*} \in \mathbb{G}_1$

*// NIZK proof $\pi^{\mathsf{client}}$ for CCA security*

5: $\chi^{\mathsf{client}} := ([\![1]\!]_1, \delta_{i^*}, \mathsf{ct}_1, \mathsf{ct}_2)$
6: $\pi^{\mathsf{client}} \leftarrow \mathsf{DLEq.Prove}(\chi^{\mathsf{client}}, r)$
7: $\mathsf{ct} := (\mathsf{ct}_1, \mathsf{ct}_2, i^*, \mathsf{m} \oplus \mathsf{H}(r \cdot \delta_u \circ h_{n+1+i^*-u}), \pi^{\mathsf{client}})$
8: **return** $\mathsf{ct}$

$\underline{\Pi\text{-}\mathsf{BDec}_1(\mathsf{mpk}, \{\mathsf{ct}_\ell\}_{\ell\in[B]}, \langle\mathsf{sk}_j\rangle)}$:

1: **for** all $\ell \in [B]$, parse $\mathsf{ct}_\ell$ as $(\mathsf{ct}_{\ell,1}, \ldots, \mathsf{ct}_{\ell,4}, \mathsf{ct}_{\ell,5})$
2: $U := [B]$

*// $U$ contains the indices of only valid ciphertexts*

3: **for** $\ell \in [B]$, $\chi_\ell^{\mathsf{client}} := ([\![1]\!]_1, \delta_{\mathsf{ct}_{\ell,3}}, \mathsf{ct}_{\ell,1}, \mathsf{ct}_{\ell,2})$
4: $U := \{\ell \mid \ell \in [B] \wedge \mathsf{DLEq.Verify}(\chi_\ell^{\mathsf{client}}, \mathsf{ct}_{\ell,5}) = 1\}$
5: $\sigma_j := \mathsf{sk}_j^{-1} \cdot \Sigma_{\ell\in U} \mathsf{ct}_{\ell,1}$

*// NIZK proof $\pi_j^{\mathsf{server}}$ for the correctness of $\sigma_j$*

6: $\chi_j^{\mathsf{server}} := ([\![1]\!]_1, \Sigma_{\ell\in U} \mathsf{ct}_{\ell,1}, [\![\mathsf{sk}_j^{-1}]\!]_1, \sigma_j)$
7: $\pi_j^{\mathsf{server}} \leftarrow \mathsf{DLEq.Prove}(\chi_j^{\mathsf{server}}, \mathsf{sk}_j^{-1})$
8: **return** $\sigma_j, \pi_j^{\mathsf{server}}$

$\underline{\Pi\text{-}\mathsf{BDec}_2(\{\sigma_j, \pi_j^{\mathsf{server}}\}_{j\in[N]})}$:

1: $S := [N]$

*// Update $S$ to include the indices of only those servers who send a valid message*

2: $S := \{j \mid j \in S \wedge \mathsf{DLEq.Verify}(\chi_j^{\mathsf{server}}, \pi_j^{\mathsf{server}}) = 1\}$

*// Select $V \subseteq S$ of servers whose combined weight exceeds the corruption threshold $t$*

3: Select $V \subseteq S$ s.t. $\sum_{j\in V} w_j > t$

*// Defining the share interpolation domain $\Omega'$ and lagrange coefficients $L_\omega$*

4: $\Omega' := \bigcup_{j\in V} \Omega_j$ and $\{L_\omega\}_{\omega\in\Omega'} := \mathsf{Lagrange}(\Omega', t)$

*// Defining the set $E \subseteq [n]$ of indices used by the ciphertext batch*

5: $E := \{\mathsf{ct}_{\ell,3}\}_{\ell\in U}$
6: **for** all $i \in E$, $\alpha_i := \sum_{j\in V} \sum_{\omega\in\Omega_j} \sigma_j \circ \gamma_{i,j,\omega} \cdot L_\omega$
7: **for** all $i \in E$, $\beta_i := \sum_{\ell\in U, \mathsf{ct}_{\ell,3}\neq i} \mathsf{ct}_{\ell,2} \circ h_{n+1+i-\mathsf{ct}_{\ell,3}}$

*// Decrypting all the valid ciphertexts*

8: **for** all $\ell \in U$, $\widetilde{\mathsf{m}_\ell} := \mathsf{ct}_{\ell,4} \oplus \mathsf{H}(\alpha_{\mathsf{ct}_{\ell,3}} - \beta_{\mathsf{ct}_{\ell,3}})$

*// Default output for all the invalid ciphertexts*

9: **for** all $\ell \in [B] \setminus U$, $\widetilde{\mathsf{m}_\ell} := \perp$
10: **return** $\{\widetilde{\mathsf{m}_\ell}\}_{\ell\in[B]}$

Figure 2: Construction of $\mathsf{wBTE}$ scheme for $N$ servers, where server $i^{\text{th}}$ has weight $w_i$, and supporting a maximum batch size $B_{\mathsf{max}}$. We highlight parts needed for robustness and CCA security (against any number of malicious clients and upto $t$ malicious servers running $\Pi$-BDec) in blue.

*of the proof system* $\mathsf{DLEq}$ *(Def. 3.4) the construction in Fig. 2 is a CCA secure weighted BTE scheme as per Def. 4.3.*

*Proof.* We prove this theorem using a sequence of hybrids, with the goal of removing any dependence on the choice of the bit $b$ of Def. 4.3. Recall from $\mathsf{Expt}^{(\mathsf{w})\mathsf{BTE\text{-}CCA}}_{\mathcal{A},b}$ of Fig. 1, the adversary corrupts the set $\mathcal{C}$ and obtains the corresponding key shares $\{\mathsf{sk}_j\}_{j\in\mathcal{C}}$. Their goal is to generate $m_0, m_1, i^*$ and guess $b$ given honestly generated $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, m_b, i^*)$. The adversary can invoke the protocol via $\mathcal{O}_{\mathsf{BDec}}$ at any time

Table 2: Communication (in terms of $\mathbb{G}$ and $\mathbb{Z}_p$ elements) and computation (in terms of $\mathbb{G}$ operations) cost of different parts involved in $\mathsf{wBTE}$ schemes. We have $N$ servers with total weight $W$, and a batch of $B = B_{\mathsf{max}}$ ciphertexts. $N'$ is the size of an arbitrary subset of servers whose combined weight exceeds the corruption threshold $t$ ($N'$ is the size of set $V$ in $\Pi\text{-}\mathsf{BDec}_2$ in Fig. 2). We split all the costs into two parts where we highlight the overhead incurred due to NIZK proofs in blue.

| **Metrics** | Our work (Fig. 2) | BEAT-MEV*+ virtualization | BEAT-MEV [BFOQ25] +virtualization |
|---|---|---|---|
| CRS size ($\lvert\mathsf{pp}\rvert + \lvert\mathsf{mpk}\rvert$) | $(2B{+}N)\lvert\mathbb{G}_1\rvert + B(2+W)\lvert\mathbb{G}_2\rvert$ | $(B+1{+}W)\lvert\mathbb{G}_1\rvert + 2B\lvert\mathbb{G}_2\rvert$ | $(B+1{+}W)\lvert\mathbb{G}_1\rvert + B^2\lvert\mathbb{G}_2\rvert$ |
| Ciphertext size ($\mathsf{ct}$) | $\lvert\mathsf{m}\rvert + 2\lvert\mathbb{G}_1\rvert{+}2\lvert\mathbb{Z}_p\rvert$ | $\lvert\mathsf{m}\rvert + 3\lvert\mathbb{G}_1\rvert{+}3\lvert\mathbb{Z}_p\rvert$ | $\lvert\mathsf{m}\rvert + 3\lvert\mathbb{G}_1\rvert{+}3\lvert\mathbb{G}_1\rvert + 2\lvert\mathbb{Z}_p\rvert$ |
| Client computation ($\mathsf{Enc}$) | $3\ \mathbb{G}_1.\mathsf{mult} + \mathbb{G}.\mathsf{pair}{+}2\ \mathbb{G}_1.\mathsf{mult}$ | $4\ \mathbb{G}_1.\mathsf{mult} + \mathbb{G}.\mathsf{pair}{+}4\ \mathbb{G}_1.\mathsf{mult}$ | $4\ \mathbb{G}_1.\mathsf{mult} + \mathbb{G}.\mathsf{pair}{+}4\ \mathbb{G}_1.\mathsf{mult}$ |
| Per server download ($\Pi\text{-}\mathsf{BDec}_2$ input size) | $N\lvert\mathbb{G}_1\rvert{+}2N\lvert\mathbb{Z}_p\rvert$ | $W\lvert\mathbb{G}_1\rvert{+}2N\lvert\mathbb{Z}_p\rvert$ | $W\lvert\mathbb{G}_1\rvert{+}2W\lvert\mathbb{Z}_p\rvert$ |
| Computation cost of server $i$ ($\Pi\text{-}\mathsf{BDec}$) | $B\Big(\sum_{j=1}^{N'} \mathbb{G}_2.\mathsf{msm}(w_j) + \mathbb{G}.\mathsf{mpair}(N')\Big)$ $+1.5B\log_2(3B)\ (\mathbb{G}_1.\mathsf{mult} + \mathbb{G}_\mathsf{T}.\mathsf{mult})$ $+3B\ \mathbb{G}.\mathsf{pair}{+}4(B+N)\ \mathbb{G}_1.\mathsf{mult}$ | $w_i\ \mathbb{G}_1.\mathsf{mult} + \mathbb{G}_1.\mathsf{msm}(t)$ $+B\ \mathbb{G}.\mathsf{mpair}(B){+}\sum_{j=1}^{N} 2\mathbb{G}_1.\mathsf{msm}(w_j)$ $+(7B+4N)\ \mathbb{G}_1.\mathsf{mult}$ | $w_i\ \mathbb{G}_1.\mathsf{mult} + \mathbb{G}_1.\mathsf{msm}(t)$ $+B^2\ \mathbb{G}.\mathsf{pair}$ $+(7B+4W)\ \mathbb{G}_1.\mathsf{mult}$ |

* This is an improved variant of the original BEAT-MEV where we use (i) the PRF from [DGM24b] with linear CRS size, (ii) multipairing optimization to reduce server computation (see "Warmup optimization" in Section 2.2), (iii) batch Schnorr proofs (see Section 3.2) to reduce server communication and computation, and (iv) apply known Schnorr optimization (see Section 3.2) to reduce the ciphertext size.

but is not allowed to have the honest parties (simulated by the challenger) decrypt $\mathsf{ct}^*$.

**$\mathbf{H}_0$.** This is the original CCA security experiment.

**$\mathbf{H}_1$.** This is the same as $\mathbf{H}_0$, except that instead of generating the proof $\pi^{\mathsf{client}}$ as part of $\mathsf{ct}^*$ honestly, we generate a simulated proof using the simulator $\mathcal{S}$ for $\mathsf{DLEq}$.

**Lemma 4.6.** *Assuming* $\mathsf{DLEq}$ *satisfies zero-knowledge,* $\mathbf{H}_1$ *is indistinguishable from* $\mathbf{H}_0$.

*Proof.* This follows directly via a reduction to the zero-knowledge property of $\mathsf{DLEq}$. Hence, the overall advantage $\left|\mathsf{Adv}^1_{\mathcal{A}}(\lambda) - \mathsf{Adv}^0_{\mathcal{A}}(\lambda)\right| \leq \mathsf{Adv}^{\mathsf{ZK}}_{\mathcal{B}}(\lambda)$ is is negligible. □

**$\mathbf{H}_2$.** This is the same as $\mathbf{H}_1$, except that on calls to the batch decryption oracle $\mathcal{O}_{\mathsf{BDec}}$, instead of running $\Pi\text{-}\mathsf{BDec}\left(\mathsf{mpk}, \{\mathsf{ct}_\ell\}_{\ell\in[B]}, \langle P_j\rangle_{j\in[N]}\right)$ honestly, the challenger generates a simulated proof $\pi^{\mathsf{server}}_j$ for each $j \in [N]$ in Step 7 using the simulator $\mathcal{S}$ for $\mathsf{DLEq}$.

**Lemma 4.7.** *Assuming* $\mathsf{DLEq}$ *satisfies zero-knowledge,* $\mathbf{H}_2$ *is indistinguishable from* $\mathbf{H}_1$.

*Proof.* This follows straightforwardly via a reduction to the zero-knowledge property of $\mathsf{DLEq}$ and taking the union over all $j \in [N]$. So that the overall advantage is negligible, i.e.,

$$\left|\mathsf{Adv}^2_{\mathcal{A}}(\lambda) - \mathsf{Adv}^1_{\mathcal{A}}(\lambda)\right| \leq N \cdot \mathsf{Adv}^{\mathsf{ZK}}_{\mathcal{B}}(\lambda)\ .$$ □

**$\mathbf{H}_3$.** This hybrid is the same as $\mathbf{H}_2$, except we change the way we compute the response of the batch decryption oracle $\mathcal{O}_{\mathsf{BDec}}$. In particular, for each call to the batch decryption oracle $\mathcal{O}_{\mathsf{BDec}}$ with a batch of ciphertext $\{\mathsf{ct}_\ell = (\llbracket r_\ell \rrbracket_1, \cdots)\}_{\ell\in[B]}$, we first extract the subset with valid $\mathsf{DLEq}$ proofs $\{r_\ell\}_{\ell\in[U]}$ (Step 3) using the algebraic nature of $\mathcal{A}$ (more on this below). Next, for each $j \in [N]$, compute $\sigma_j = \llbracket \mathsf{sk}_j^{-1} \rrbracket_1 \cdot \sum_{\ell\in[B]} r_\ell$.

*Extracting* $\{r_\ell\}_{\ell\in[U]}$. Recall that, $\mathcal{A}$ is algebraic, i.e., with every group element $\llbracket a \rrbracket$ it outputs, it outputs a coefficient representations $(a_0, a_1, ...)$ in terms of its input bases $G_1, G_2, ...$ (see Def. 3.6). For every ciphertext $\{\mathsf{ct}_\ell = (\llbracket r_\ell \rrbracket_1, \dots)\}_{\ell\in U}$ with a valid $\mathsf{DLEq}$ proof and extracted value $r_\ell$, let $r_{\ell,0}$ be the provided coefficient of

$[\![r_\ell]\!]_1$ for the base $G_1 = [\![1]\!]_1$, i.e., $[\![r_\ell]\!]_1 \coloneqq r_{\ell,0} \cdot [\![1]\!]_1 + r_{\ell,1} \cdot G_2 + \cdots$. Then in this hybrid we set $r_\ell \coloneqq r_{\ell,0}$, so that,

$$\sigma_j \coloneqq [\![\mathsf{sk}_j^{-1}]\!]_1 \cdot \sum_{\ell \in [B]} r_{\ell,0} \ .$$

**Lemma 4.8.** *Assuming hardness of q-strong Diffie-Hellman (Assumption 3.1) and discrete logarithm (Assumption 3.2), hybrid* $\mathbf{H}_2$ *and* $\mathbf{H}_3$ *are computationally indistinguishable.*

We prove Lemma 4.8 in Appendix A.

**$\mathbf{H}_4$.** This is the same as $\mathbf{H}_3$, except we change the way we compute the challenge ciphertext. In particular, instead of computing the challenge $\mathsf{ct}_4^*$ as $\mathsf{m}_b \oplus \mathsf{H}(r^* \cdot \delta_u \circ h_{n+1+i^*-u})$ for some $u \in [n] \setminus i^*$, we sample a uniform value $\rho \leftarrow\!\!\$\ \mathbb{Z}_p$, and set $\mathsf{ct}_4^* \coloneqq \mathsf{m}_b \oplus \mathsf{H}([\![\rho]\!]_\mathsf{T})$.

**Lemma 4.9.** *Assuming the* $n$-$\mathsf{DH}^+$ *problem (Assumption 3.4) is hard,* $\mathbf{H}_4$ *is indistinguishable from* $\mathbf{H}_3$.

We prove Lemma 4.9 in Appendix A.

**Lemma 4.10.** *An adversary has negligible advantage in* $\mathbf{H}_4$.

*Proof.* Since $\rho$ is sampled uniformly, $\mathsf{ct}_3^* \coloneqq \mathsf{m}_b \oplus \mathsf{H}([\rho]_\mathsf{T})$ information theoretically hides the message. Thus, $\mathsf{ct}^*$ is independent of the message. □

It therefore follows that

$$\mathsf{Adv}^{\mathsf{wBTE\text{-}CCA}}_{\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{ZK}}_{\mathcal{B}}(\lambda) + Q \cdot \mathsf{Adv}^{\mathsf{SimExt}}_{\mathcal{B}}(\lambda) + n \cdot \mathsf{Adv}^{n\text{-}\mathsf{DH}^+}_{\mathcal{B}}(\lambda) + \mathsf{negl}(\lambda). \qquad \square$$

Next, we prove the robustness of our scheme.

**Theorem 4.11** (Robustness)**.** *Assuming the soundness of the proof system* DLEq *(Definitions 3.3), the protocol in Fig. 2 is a robust weighted BTE scheme.*

*Proof.* Recall the robustness game $\mathsf{Expt}^{\mathsf{BTE\text{-}rob}}_{\mathcal{A}}$. In this game, $\mathcal{A}$ first outputs a batch of messages $\{\mathsf{m}_i\}_{i\in[B]\setminus S}$ for some $S \subset [B]$. Next, $\mathcal{A}$ receives encryption of these messages $\{\mathsf{ct}_i\}_{i\in[B]\setminus S}$ and outputs $|S|$ arbitrary ciphertexts $\{\mathsf{ct}_i\}_{i\in S}$ of its choice. Without loss of generality, let us assume that each ciphertext $\mathsf{ct} \in \{\mathsf{ct}_i\}_{i\in S}$ includes a valid DLEq proof. The game then decrypts the set $\{\mathsf{ct}_i\}_{i\in[B]}$ of ciphertexts using the batch decryption protocol. Let $\{\widetilde{\mathsf{m}}_i\}_{[B]}$ be the resulting decryptions. We say that $\mathcal{A}$ wins the $\mathsf{Expt}^{\mathsf{BTE\text{-}rob}}_{\mathcal{A}}$ game, if there exists at least one index $i \in [B] \setminus S$ with $\mathsf{m}_i \neq \widetilde{\mathsf{m}}_i$.

We give a reduction to the soundness of the DLEq protocol. To that end, let $\mathcal{A}$ be a PPT adversary that wins the robustness game (Def. 4.4) with some non-negligible advantage $\eta(\lambda)$, then we construct an efficient reduction $\mathcal{B}$ that uses $\mathcal{A}$ to break the soundness of DLEq with probability $\eta(\lambda)/(S + \mathcal{C})$. Here, $S$ defined above is the number of adversarial ciphertexts included in the challenge batch, and $\mathcal{C}$ is the set of corrupt parties in the $\mathsf{Expt}^{\mathsf{BTE\text{-}rob}}_{\mathcal{A}}$ game.

From the correctness property of our scheme, it is easy to see that for $\mathcal{A}$ to win the robustness game, it either (i) needs to create a malformed ciphertext $\mathsf{ct} \in \{\mathsf{ct}_i\}_{i\in S}$ with valid DLEq proof, or (ii) needs to output an invalid partial decryption on behalf of a malicious server $j \in \mathcal{C}$ with a valid DLEq proof. Otherwise, the decryption process will correctly recover the original message.

Based on the above insight, we build $\mathcal{B}$ as follows:

- $\mathcal{B}$ upon receiving the CRS from the DLEq soundness challenger, it embeds it as part of the public parameters pp, sends them to $\mathcal{A}$, and continues the rest of the game honestly.
- $\mathcal{B}$ waits for $\mathcal{A}$ to win. If $\mathcal{A}$ does not win, $\mathcal{B}$ aborts the game.
- Otherwise, $\mathcal{B}$ samples $k \leftarrow\!\!\$\ [|S| + |\mathcal{C}|]$, and

1. If $k \leq |S|$, let $\mathsf{ct}' = (\mathsf{ct}'_1, \ldots, \mathsf{ct}'_5)$ be the $k$-th ciphertext among the batch $\{\mathsf{ct}_i\}_{i \in S}$ of ciphertexts $\mathcal{A}$ outputs. $\mathcal{B}$ then returns the $\chi := ([\![1]\!]_1, \delta_{\mathsf{ct}'_3}, \mathsf{ct}'_1, \mathsf{ct}'_3)$ as the instance and proof $\pi_k^{\mathsf{client}} = \mathsf{ct}'_5$ to the DLEq soundness challenger.
2. If $|S| < k \leq |S| + |\mathcal{C}|$, let $j$ be the index of the $(k - |\mathcal{C}|)$-th corrupt server in $\mathcal{C}$. Then, $\mathcal{B}$ returns $\chi := ([\![1]\!]_1, \sum_{\ell \in [B]} \mathsf{ct}_\ell, [\![\mathsf{sk}_j^{-1}]\!]_1, \sigma_j)$ and the proof $\pi_j^{\mathsf{server}}$ to the soundness challenger.

In the above, reduction, when when $\mathcal{A}$ wins the game $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{BTE\text{-}rob}}$ with probability $\eta(\lambda)$, i.e., when at least one of the DLEq proof among the valid DLEq proof included as part of the ciphertext $\{\mathsf{ct}_i\}_{i \in S}$ or the proofs the corrupt servers output corresponds to a instance not in the DLEq language, then $\mathcal{B}$ wins the DLEq robustness game with probability $\eta(\lambda)/(|S| + |\mathcal{C}|)$. It follows that

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{BTE\text{-}rob}}(\lambda) \leq (|S| + |\mathcal{C}|) \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{Snd}}(\lambda) \ . \qquad \square$$

# 5 Experimental Evaluation

We implement and evaluate all schemes and baselines in `Rust`, and will release the code publicly.

**Setup.** We run all experiments on a MacBook Pro with an Apple M3 Max CPU with 48 GB RAM. For fair comparison, all runs are single-threaded both for our scheme and the baselines. We will discuss parallelism for our scheme later.

**Baselines.** We compare our results against BEAT-MEV [BFOQ25] and an improved version of it (see caption of Table 2), which we use as our baselines. We explained in Section 1.2 the reason for choosing this as the baseline.

**Pairing group details.** We use BLS12-381 pairing-friendly eliptic curve from `blstrs` library [Fil20,Sup20]. On this curve, the size of each element of $\mathbb{G}_1$,$\mathbb{G}_2$ and $\mathbb{G}_\mathsf{T}$ takes 48, 96, and 288 bytes respectively. Each $\mathbb{G}_1$ and $\mathbb{G}_2$ scalar multiplication takes 53 and 105 microseconds respectively, and each pairing operation takes 454 microseconds on average.

## 5.1 FFT optimization and weighted extension

In Table 3, we show the absolute costs of our scheme (with FFT optimization and weighted extension) for all the relevant metrics and varying number of batch sizes and total weight. We also show our improvement factor (large is better) w.r.t. both the original BEAT-MEV & optimized BEAT-MEV baseline approaches within $(\cdot)$ & $[\cdot]$, respectively.

In terms of batch decryption time, our work substantially outperforms the baselines for moderate to large batch sizes by a factor of $1.5\times - 22\times$. Similarly, the per server download size in our scheme doesn't grow with the total weight at all and shows a huge improvement $2.7\times - 50\times$ compared to the baselines. This communication cost reduction is particularly critical in the mempool privacy application. Encrypted mempools require running BTE once per block, at high frequency and under tight latency constraints. Here, network bandwidth between validators—not raw CPU—is often the bottleneck, especially when the validator set is large and block throughput is high. We also see a small side benefit of improvement in ciphertext size and encryption time.

**Parallelization.** Performing the FFT consumed more than 95% of the running time in our evaluations. Moreover, this step is trivial to parallelize. Using the Rayon library we executed each of the $\log B$ levels of the FFT, for batch size $B$, in parallel across 12 physical CPU cores (using 16 threads) and observe a linear improvement of $12\times$. In particular, for the combined time to compute the $\mathbb{G}_1, \mathbb{G}_2$ FFT, $B$ pairings and the $\mathbb{G}_\mathsf{T}$ inverse FFT with $B \in \{1024, 256\}$, we observe a 12 and $11.8\times$ reduction in running time respectively. This suggests that when decreased latency is required using multiple cores is a simple and effective solution.

Table 3: Concrete costs of our approach and baseline, for different batch size and total weight. Each cell contains $X + Y$ $(A\times)[B\times]$ where $X$ metric for the core protocol and $Y$ is additional overhead due to the NIZK proofs. $B$ is our multiplicative improvement over the original BEAT-MEV and $A$ denotes our improvement over an improved version of BEAT-MEV. See Section 5.1 for details.

| **Metrics** ↓ | **Batch size** ↓ | **Total weight** (as a multiple of the server count $N = 100$) | | | | |
|---|---|---|---|---|---|---|
| | | Unweighted | 5× | 10× | 20× | 50× |
| **Batch decryption time (Single core)** | 8 | 73+27 ms (0.37×) [0.56×] | 242+27 ms (0.59×) [0.56×] | 277+27 ms (0.63×) [0.91×] | 308+27 ms (0.63×) [1.4×] | 390+27 ms (0.59×) [2.7×] |
| | 32 | 301+35 ms (0.63×) [1.3×] | 950+35 ms (0.33×) [0.56×] | 975+35 ms (0.34×) [0.67×] | 1.25+0.03 s (0.29×) [0.71×] | 1.65+0.03 s (0.24×) [0.92×] |
| | 128 | 1.4+0.07 s (1.8×) [4.5×] | 4.32+0.07 s (0.67×) [1.5×] | 4.69+0.07 s (0.59×) [1.4×] | 5.31+0.07 s (0.53×) [1.3×] | 6.69+0.07 s (0.43×) [1.1×] |
| | 512 | 6.6+0.21 s (6.0×) [15.7×] | 17.41+0.21 s (2.3×) [6.0×] | 20.4+0.21 s (2.0×) [5.2×] | 22.3+0.21 s (1.8×) [4.7×] | 25.1+0.21 s (1.6×) [4.2×] |
| | 2048 | 28.4+0.80 s (22.6×) [58×] | 74.41+0.80 s (8.8×) [22.5×] | 81.2+0.80 s (8.0×) [20.1×] | 92.1+0.80 s (7.1×) [18.5×] | 120.02+0.80 s (5.4×) [14.2×] |
| **CRS size** | 8 | 2 kB +4.8 kB (1×) [1.6×] | 386+4.8 kB (0.067×) [0.078×] | 770+4.8 kB (0.065×) [0.07×] | 1.5 MB +4.8 kB (0.065×) [0.068×] | 3.8 MB +4.8 kB (0.064×) [0.065×] |
| | 32 | 7.8 kB +4.8 kB (1×) [8.3×] | 1.5 MB +4.8 kB (0.021×) [0.083×] | 3.0 MB +4.8 kB (0.019×) [0.049×] | 6.1 MB +4.8 kB (0.017×) [0.032×] | 15.3 MB +4.8 kB (0.016×) [0.022×] |
| | 128 | 31.2 kB +4.8 kB (1×) [44.5×] | 6 MB +4.8 kB (0.0092×) [0.27×] | 12.3 MB +4.8 kB (0.0064×) [0.13×] | 24.6 MB +4.8 kB (0.0052×) [0.068×] | 61.4 MB +4.8 kB (0.0044×) [0.03×] |
| | 512 | 157.8 kB +4.8 kB (1×) [155.1×] | 24 MB +4.8 kB (0.0076×) [1.05×] | 49.2 MB +4.8 kB (0.0042×) [0.53×] | 98.4 MB +4.8 kB (0.0026×) [0.26×] | 245.9 MB +4.8 kB (0.0016×) [0.1×] |
| | 2048 | 499.7 kB +4.8 kB (1×) [811.4×] | 98.8 MB +4.8 kB (0.0053×) [4.0×] | 197.1 MB +4.8 kB (0.0028×) [2.0×] | 393.8 MB +4.8 kB (0.0015×) [1.02×] | 983.6 MB +4.8 kB (0.00075×) [0.42×] |
| **Per server download size** | | 4.8+6.4 kB (1×) [1×] | 4.8+6.4 kB (2.7×) [5×] | 4.8+6.4 kB (4.8×) [10×] | 4.8+6.4 kB (9.1×) [20×] | 4.8+6.4 kB (22×) [50×] |
| **Ciphertext size** | | 144+96 B(1×) [1.4×] | 96+64 B (1.5×) [2.2×] | | | |
| **Encryption time** | | 632+226 $\mu$s(1×) [1×] | 555+117 $\mu$s (1.2×) [1.2×] | | | |

## 5.2 Evaluating cuckoo hashing

We instantiate the cuckoo hashing approach with $\mathsf{c} = 2$ hash functions and vary the number of bins $n$ (which translates to the PRF domain size in the BTE protocol) and the capacity of each bin $d$ (which translates to the number of sub-batches in the BTE protocol). The metric that we analyze is the following:

*For given values of $n$ and $d$, what is the maximum number of ciphertexts $B_{\mathsf{max}}$ that we can handle while ensuring less than $\epsilon = 2^{-40}$ failure probability ?*

Ideally, we would like $B_{\mathsf{max}}$ to be as close as possible to the maximum capacity $n \cdot d$. We analyze this metric based on the analysis provided in Lemma 1 of [DW07] which gives a tight upper bound on the failure probability $\epsilon$ for a given number of items $B_{\mathsf{max}}$, number of buckets $n$ and bucket capacity $d$. The proof of

Table 4: This table shows the maximum number of ciphertexts that our cuckoo hashing based approach can handle for two hash functions, $\epsilon = 2^{-40}$ statistical security, given number of sub-batches (bin capacity) and PRF domain size (number of bins). Each cell contains $X$ ($A\times$) where $X$ denote the number of ciphertexts and $A$ is the relative improvement over BEAT-MEV [BFOQ25]. Separately, we also give our ciphertext size in bytes for the weighted and unweighted scheme. Each cell contains $X+Y$ ($A\times$) where $X$ is the size without NIZK proofs and $Y$ is thier additional size. $A$ is the our multiplicative improvement over BEAT-MEV.

| **Metrics** $\rightarrow$ | **Maximum number of ciphertexts** ($\epsilon = 2^{-40}$ failure probability) | | | | | **Ciphertext size** | |
|---|---|---|---|---|---|---|---|
| Number of | PRF domain size | | | | | Unweighted | Weighted |
| sub-batches $\downarrow$ | 8 | 32 | 128 | 512 | 2048 | | |
| 8 | 9 (1.1×) | 65 (8.1×) | 830 (46.1×) | 4004 (77×) | 16117 (94.3×) | 384 B+480 B | 288 B+288 B |
| 16 | 69 (3.6×) | 488 (11×) | 2020 (14.1×) | 8157 (16.1×) | 32714 (17.7×) | (0.44×) | (0.67×) |

the lemma shows the following inequality:

$$\epsilon \leq \sum_{j=1}^{B_{\mathsf{max}}/d} \binom{n}{d} \left(\frac{B_{\mathsf{max}}(j/n)^2}{jd}\right)^{jd} \left(\frac{B_{\mathsf{max}} - B_{\mathsf{max}}(j/n)^2}{B_{\mathsf{max}} - jd}\right)^{B_{\mathsf{max}}-jd} \tag{8}$$

Since the above expression is not closed-form, the authors in [DW07] further approximate the above expression using various bounds in order to derive a succinct expression for the asymptotic analysis. However, as we are interested in a concrete analysis in this work, we directly use the expression in Eq. (8) and numerically compute the value of $\epsilon$. Specifically, to compute our metric of interest $B_{\mathsf{max}}$, we simply search over the possible choices of $B_{\mathsf{max}}$ for fixed parameters $n, d$ to find the maximum value of $B_{\mathsf{max}}$ for which the resulting $\epsilon$ is less than $2^{-40}$. We report these values in Table 4. For comparison, we also show the improvement factor (which is greater than $10\times$ for reasonable values of $B_{\mathsf{max}}$) w.r.t sub-batching approach suggested in BEAT-MEV which has the following failure probability.

$$\epsilon \leq 1 - \left(1 - \frac{1}{n^d}\right)^{\binom{B_{\mathsf{max}}}{d+1}} \tag{9}$$

Consider for example the choice of $n = 128$ buckets (which also governs the CRS size) with capacity $d = 8$ (corresponding to the number of sub-batches for decryption). Table 4 shows that these parameters enable batch decryption of 830 transactions, except with $2^{-40}$ probability. These parameters utilize $830/1024 = 0.81$ of the capacity of the hash table, and enable to encrypt $46.1\times$ more ciphertexts than BEAT-MEV at a mere cost of increasing the ciphertext size by $1.5 \times -2.25\times$. (And since the failure probability of cuckoo hashing is a decreasing function of the number of items $B$, we expect to have better results for larger batch sizes.)

## Acknowledgments

The authors would like to thank Michael Setrin for many helpful discussions related to this paper, and Wenhao Wang for early discussions on this project.

## References

[ABDG25] Amit Agarwal, Kushal Babel, Sourav Das, and Babak Poorebrahim Gilkalaye. Time-lock encrypted storage for blockchains. Cryptology ePrint Archive, Paper 2025/2048, 2025. 1

[AFP25] Amit Agarwal, Rex Fernando, and Benny Pinkas. Efficiently-thresholdizable batched identity based encryption, with applications. In *Annual International Cryptology Conference*. Springer, 2025. 1, 2, 3, 4

[AN20] Ramiro Alvarez and Mehrdad Nojoumian. Comprehensive survey on privacy-preserving protocols for sealed-bid auctions. *Computers & Security*, 88:101502, 2020. 2

[APB+04] Riza Aditya, Kun Peng, Colin Boyd, Ed Dawson, and Byoungcheon Lee. Batch verification for equality of discrete logarithms and threshold decryptions. In *ACNS 2004*, volume 3089 of *LNCS*, pages 494–508. Springer, Berlin, Heidelberg, June 2004. 12

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018. 31

[BCF+25] Jan Bormet, Arka Rai Choudhuri, Sebastian Faust, Sanjam Garg, Hussien Othman, Guru-Vamsi Policharla, Ziyan Qu, and Mingyuan Wang. Beast-mev: Batched threshold encryption with silent setup for mev prevention. *Cryptology ePrint Archive*, 2025. 1, 4, 31

[BDKJ23] Kushal Babel, Philip Daian, Mahimna Kelkar, and Ari Juels. Clockwork finance: Automated analysis of economic security in smart contracts. In *2023 IEEE Symposium on Security and Privacy (SP)*, 2023. 2

[BFOQ25] Jan Bormet, Sebastian Faust, Hussien Othman, and Ziyan Qu. {BEAT-MEV}: Epochless approach to batched threshold encryption for {MEV} prevention. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 3457–3476, 2025. 1, 2, 4, 9, 10, 13, 14, 15, 17, 19, 21

[BLT25] Dan Boneh, Evan Laufer, and Ertem Nusret Tas. Batch decryption without epochs and its application to encrypted mempools. *Cryptology ePrint Archive*, 2025. 4

[BO22] Joseph Bebel and Dev Ojha. Ferveo: Threshold decryption for mempool privacy in bft networks. *Cryptology ePrint Archive*, 2022. 2, 3, 7, 31

[BS04] Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel, editors, *ACM CCS 2004*, pages 168–177. ACM Press, October 2004. 12

[CCN+23] Andrea Cerulli, Aisling Connolly, Gregory Neven, Franz-Stefan Preiss, and Victor Shoup. vetkeys: How a blockchain can keep many secrets. *Cryptology ePrint Archive*, 2023. 3

[CDK+22] Matteo Campanelli, Bernardo David, Hamidreza Khoshakhlagh, Anders Konring, and Jesper Buus Nielsen. Encryption to the future: a paradigm for sending secret messages to future (anonymous) committees. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 151–180. Springer, 2022. 3

[CGJ+17] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017. 2

[CGPP24] Arka Rai Choudhuri, Sanjam Garg, Julien Piet, and Guru-Vamsi Policharla. Mempool privacy via batched threshold encryption: Attacks and defenses. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024. 1, 3, 4

[CGPW25] Arka Rai Choudhuri, Sanjam Garg, Guru Vamsi Policharla, and Mingyuan Wang. Practical mempool privacy via one-time setup batched threshold encryption. In *34th USENIX Security Symposium*, 2025. 1, 3, 4

[CP93] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, Berlin, Heidelberg, August 1993. 12, 15, 16, 32

[CT65] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965. 6

[DCX+23] Sourav Das, Philippe Camacho, Zhuolun Xiang, Javier Nieto, Benedikt Bünz, and Ling Ren. Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 356–370. ACM Press, November 2023. 24, 25

[DFL24] Stefan Dziembowski, Sebastian Faust, and Jannik Luhn. Shutter network: Private transactions from threshold cryptography. *Cryptology ePrint Archive*, 2024. 3

[DGK+20] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE symposium on security and privacy (SP)*, 2020. 2

[DGM24a] Jesko Dujmovic, Rachit Garg, and Giulio Malavolta. Time-lock puzzles with efficient batch solving. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 311–341. Springer, 2024. 3

[DGM24b] Jesko Dujmovic, Rachit Garg, and Giulio Malavolta. Time-lock puzzles with efficient batch solving. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 311–341. Springer, Cham, May 2024. 5, 6, 8, 9, 10, 17

[DH84] Pierre Duhamel and Henk Hollmann. 'split radix'fft algorithm. *Electronics letters*, 20(1):14–16, 1984. 6

[DHMW23] Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wohnig. Mcfly: verifiable encryption to the future made practical. In *International Conference on Financial Cryptography and Data Security*. Springer, 2023. 3

[DW07] Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theoretical Computer Science*, 380(1-2):47–68, 2007. 3, 10, 20, 21

[ElG85] Taher ElGamal. On computing logarithms over finite fields. In *Conference on the theory and application of cryptographic techniques*, pages 396–402. Springer, 1985. 3

[EMM06] Ulfar Erlingsson, Mark Manasse, and Frank McSherry. A cool and practical alternative to traditional hash tables. In *Proc. 7th Workshop on Distributed Data and Structures (WDAS'06)*, pages 1–6, 2006. 3, 10

[Fil20] Filecoin Project. blstrs: BLS12-381 pairing-friendly elliptic curve construction using the blst library as backend. `https://github.com/filecoin-project/blstrs`, 2020. Accessed: 2025-11-14. 19

[GKPW24] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold encryption with silent setup. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 352–386. Springer, Cham, August 2024. 10

[GMR23] Nicolas Gailly, Kelsey Melissaris, and Yolan Romailler. tlock: Practical timelock encryption from threshold bls. *Cryptology ePrint Archive*, 2023. 1, 3

[Goo58] Irving John Good. The interaction algorithm and practical fourier analysis. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 20(2):361–372, 1958. 6

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 305–326. Springer, 2016. 31

[KLJD23] Alireza Kavousi, Duc V Le, Philipp Jovanovic, and George Danezis. Blindperm: Efficient mev mitigation with an encrypted mempool and permutation. *Cryptology ePrint Archive*, 2023. 2

[Kur02] Kaoru Kurosawa. Multi-recipient public-key encryption with shortened ciphertext. In *International Workshop on Public Key Cryptography*, pages 48–63. Springer, 2002. 31

[MGZ22] 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. 3

[MP23] Brice Minaud and Charalampos Papamanthou. Generalized cuckoo hashing with a stash, revisited. *Information Processing Letters*, 181:106356, 2023. 3, 10

[New25] MSN News. Jurors ask pointed questions in mev bot trial as deliberations proceed, 2025. Accessed: 2025-11-13. 2

[PPA+20] Claudia Pop, Mirela Prata, Marcel Antal, Tudor Cioara, Ionut Anghel, and Ioan Salomie. An ethereum-based implementation of english, dutch and first-price sealed-bid auctions. In *2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 491–497, 2020. 2

[PR01] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *Algorithms - ESA 2001, 9th Annual European Symposium*, volume 2161 of *Lecture Notes in Computer Science*, pages 121–133. Springer, 2001. 9

[QZG22] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 198–214. IEEE, 2022. 2

[SAS24] Sora Suegami, Shinsaku Ashizawa, and Kyohei Shibano. Constant-cost batched partial decryption in threshold encryption. *Cryptology ePrint Archive*, 2024. 1

[SB02] Henrik V Sorensen and C Sidney Burrus. Efficient computation of the dft with only a subset of input or output points. *IEEE transactions on signal processing*, 41(3):1184–1200, 2002. 6

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, New York, August 1990. 15

[Sup20] Supranational. blst: Multilingual BLS12-381 signature library. `https://github.com/supranational/blst`, 2020. Accessed: 2025-11-14. 19

[Wie17] Udi Wieder. *Hashing, Load Balancing and Multiple Choice*. Foundations and Trends in Theoretical Computer Science. Now Publishers, 2017. 10

[Yeo23] Kevin Yeo. Cuckoo hashing in cryptography: Optimal parameters, robustness and applications. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 197–230. Springer, Cham, August 2023. 10

# A Security Proofs

To prove Lemma 4.8, we first argue that assuming hardness of $q$-strong Diffie-Hellman (Assumption 3.1) and discrete logarithm, $r_\ell = r_{\ell,0}$. For $r_\ell \neq r_{\ell,0}$, the adversary mush have found two representations $r_\ell [\![1]\!]_1 = r_{\ell,0}[\![1]\!]_1 + r_{\ell,1}G_2 + ...$ in the available bases, e.g. the CRS. To prove the equality of the representations, we will use the following helper lemma from [DCX+23, Theorem 5.1].

**Lemma A.1.** *[DCX+23, Theorem 5.1] Assuming the hardness of q-SDH, no PPT adversary $\mathcal{A}$ on input $\{[\![x^i]\!]_1\}_{i\in[0,q]}$ for some $q \in \mathbb{N}$ can output a non-zero polynomial $\alpha(X) \in \mathbb{Z}_p[X]$ of degree $\leq q$ such that $\alpha(x) = 0$.*

*Proof.* For the sake of contradiction, assume that $\mathcal{A}$ outputs (in the coefficient representation) a non-zero polynomial $\alpha(X)$ of degree $\leq q$ such that $\alpha(x) = 0$. Let $\alpha(X)$ be,

$$\alpha(X) = \alpha_0 + \alpha_1 X + \alpha_2 X^2 + \cdots + \alpha_q X^q$$

Now consider two exclusive possibilities: (i) $\alpha_0 \neq 0$ and (ii) $\alpha_0 = 0$. In the former case, we can write

$$x \cdot (\alpha_q \cdot x^{q-1} + \cdots + \alpha_1) = -\alpha_0 \implies \frac{\alpha_q \cdot x^{q-1} + \cdots + \alpha_1}{-\alpha_0} = \frac{1}{x} \tag{10}$$

Given Eq. (10), $\mathcal{A}$ can use $\{\alpha_i\}_{i\in[0,q]}$ and $\{[\![x^i]\!]_1\}_{i\in[0,q]}$ to efficiently compute $[\![x^{-1}]\!]_1$, and output $(0, [\![x^{-1}]\!]_1)$ as a valid $q$-SDH solution.

Next, in the latter case where $\alpha_0 = 0$, since $\alpha(X)$ is a non-zero polynomial, we can rewrite $\alpha(X)$ as $\alpha(X) = X^k \cdot \alpha'(X)$ for some $0 < k < q$ such that $\alpha'(x)$ has a non-zero constant term $\alpha'_0$. Then, $\mathcal{A}$ can follow the same approach as in case (i) with polynomial $\alpha'(X)$ instead of $\alpha(X)$ to output a $q$-SDH solution. □

Next, we will argue that assuming hardness of $q$-SDH, all the coefficients of the terms $[\![\tau^i]\!]_1$ for any $i > 0$ are zero.

**Lemma A.2.** *In hybrid $\mathbf{H}_3$, let $\mathsf{ct} = ([\![r]\!]_1, \ldots)$ be any ciphertext appearing in an input to the $\mathcal{O}_{\mathsf{BDec}}$ oracle with a valid $\mathsf{DLEq}$ proof. Assuming the hardness of q-SDH, then all coefficients of $[\![\tau^i]\!]_1$ in the representation of $[\![r]\!]_1$ are zero for every $i > 0$.*

*Proof.* Given an adversary $\mathcal{A}$ that outputs non-zero coefficient of $[\![\tau^i]\!]_1$, we can build an adversary $\mathcal{B}$ that breaks the $q$-SDH assumption. Upon receiving the $q$-SDH instance $([\![1]\!]_1, [\![1]\!]_2, \{[\![x^i]\!]_1, [\![x^i]\!]_2\}_{i\in[q]})$ for $q = 2n + 1$, $\mathcal{B}$ interacts with $\mathcal{A}$ as follows:

- *Simulating setup.* Send the group description along with $(\{[\![x^i]\!]_1\}_{i\in[n]}, \{[\![x^i]\!]_2\}_{i\in[2n+1]\setminus\{n+1\}})$ as the CRS.
- Simulate the *key generation* step and compute the *challenge ciphertext* as per the protocol specification.
- *Simulating $\mathcal{O}_{\mathsf{BDec}}$.* Upon receiving a batch decryption query $\{\mathsf{ct}_i = ([\![r_i]\!]_1, \cdots)\}_{i\in[B]}$, with attached valid $\mathsf{DLEq}$ proof, decrypt if the following holds.

  For each $i$, let $\boldsymbol{r}_i = [r_{i,0}, \ldots r_{i,q}]$ be the *effective*[9] representation of $r_i$ in the basis $\{[\![x^k]\!]_1\}_{k\in[0,q]}$, i.e.,

  $$r_i := \sum_{k\in[0,q]} r_{i,k} \cdot [\![x^k]\!]_1$$

  If $r_{i,k} = 0$ for all $i \in [B]$ and $k > 0$, decrypt $\{\mathsf{ct}_i\}_{i\in[B]}$ as per the protocol specification. Otherwise, compute a $q$-SDH solution as below.
- *Breaking q-SDH.* Let $\mathsf{ct} = ([\![r]\!]_1, \cdots)$ be the first ciphertext with $\boldsymbol{r} = [r_0, \ldots, r_q]$ the effective coefficients of $[\![r]\!]_1$ in the basis $\{[\![x^i]\!]_1\}_{i\in[0,q]}$, i.e.,

  $$[\![r]\!]_1 = \sum_{i\in[0,q]} r_i \cdot [\![x^i]\!]_1$$

  where $r_i \neq 0$ for $i > 0$.

[9] The *effective* coefficients is the coefficient representation $\mathcal{B}$ locally computes using its knowledge of $(\mathsf{msk}, \{\mathsf{msk}_\omega\}_{\omega\in[W]}, \{\mathsf{sk}_j\}_{j\in[N]})$, the randomness it uses to program random oracles. We will argue that $\mathcal{B}$ can represent all $\mathbb{G}_1$ elements it outputs to $\mathcal{A}$ in the basis $\{[\![x^i]\!]_1\}_{i\in[0,q]}$.

Similarly, let $(A, B, z) \in \mathbb{G}_1^2 \times \mathbb{Z}_p$ be the $\mathsf{DLEq}$ proof associated with $[\![r]\!]_1$, i.e., we have:

$$z \cdot [\![1]\!]_1 = A + c \cdot [\![r]\!]_1, \text{ for } c = \mathsf{G}(A, B, [\![1]\!]_1, [\![r]\!]_1, \cdots)$$

Let $A = [\![a]\!]_1$ for some $a \in \mathbb{Z}_p$, and let $\boldsymbol{a} = [a_0, \ldots, a_q]$ be its representation i.e.,

$$[\![a]\!]_1 = \sum_{i \in [0,q]} a_i \cdot [\![x^i]\!]_1$$

Combining the above, we get

$$z = \sum_{i \in [0,q]} \left( (a_i + c \cdot r_i) \cdot x^i \right)$$

Now, consider the polynomial $\alpha(X)$, where:

$$\alpha(X) = \sum_{i \in [0,q]} \left( (a_i + c \cdot r_i) \cdot X^i \right) - z$$

Next, if $\alpha(X) \neq 0$ as a polynomial, $\mathcal{B}$ break the $q$-SDH assumption using Lemma A.1. Otherwise, abort.

*Analysis.* We by proving the following claim.

**Claim 1** (Effective representation)**.** *For any $g \in \mathbb{G}_1$ output by $\mathcal{A}$, $\mathcal{B}$ can represent $g$ in the basis $\{[\![x^i]\!]_1\}_{i \in [0,q]}$. We refer to this as the* effective *representation of $g$.*

*Proof.* Note that by definition, each $\mathbb{G}_1$ element $\mathcal{A}$ outputs can be represented using its inputs as the basis. Hence, to prove this claim, it suffices to prove that $\mathcal{B}$ can represent all $\mathbb{G}_1$ elements $\mathcal{A}$ receives as input in the basis $\{[\![x^i]\!]_1\}_{i \in [0,q]}$. We argue this below.

- By design $\mathcal{B}$ can represent $\mathbb{G}_1$ elements the CRS and the public keys in the basis $\{[\![x^i]\!]_1\}_{i \in [0,q]}$.
- For all $\mathcal{O}_{\mathsf{BDec}}$ queries, $\mathcal{B}$ can use its knowledge of $\{\mathsf{sk}_j\}_{i \in [N]}$ to represent the partial decryptions $\{\sigma_j\}_{j \in [n]}$ in the basis $\{[\![x^i]\!]_1\}_{i \in [0,q]}$. Similarly, $\mathcal{B}$ can represent the simulated $\mathsf{DLEq}$ proofs of correct partial decryption in the basis $\{[\![x^i]\!]_1\}_{i \in [0,q]}$ using the representation of the ciphertext $\mathcal{A}$ submits and the local randomness $\mathcal{B}$ samples to simulate the NIZK proofs.
- The challenge ciphertext and the corresponding simulated $\mathsf{DLEq}$ proof of its correctness can also be described in the basis $\{[\![x^i]\!]_1\}_{i \in [0,q]}$ (see Section 3.2). □

From Lemma A.1, assuming hardness of $q$-SDH, the polynomial $\alpha(X) = 0$ as a polynomial. This implies that for each $i \in [q]$, $a_i + c \cdot r_i = 0$. For this to hold, there are two possibilities, either (i) $c = a_i / r_i$; or (ii) $(a_i, r_i) = (0, 0)$. Since we chose $c$ at random after receiving the representation of $(A, [\![r]\!]_1)$, i.e., after receiving $(a_i, r_i)$, the probability of $c = a_i / r_i$ is $1/p$, which is negligible. Thus, except with negligible probability, $(a_i, r_i) = (0, 0)$ for all $i \in [q]$. □

**Lemma A.3.** *In hybrid $\mathbf{H}_3$, let $\mathsf{ct} = ([\![r]\!]_1, \ldots)$ be any ciphertext appearing in an input to the $\mathcal{O}_{\mathsf{BDec}}$ oracle. Assuming the hardness of discrete logarithm ($\mathsf{DL}$), if $\mathsf{ct}$ includes a valid $\mathsf{DLEq}$ proof, then all coefficients of $[\![\mathsf{sk}_j^{-1}]\!]_1$ for each $j \in [N]$ in the effective representation of $[\![r]\!]_1$ are zero.*

*Proof.* Given an adversary $\mathcal{A}$ that outputs non-zero coefficient of $[\![\mathsf{sk}_j^{-1}]\!]_1$ for some $j \in [N]$, we can build an adversary $\mathcal{B}$ that breaks the $\mathsf{DL}$ assumption as follows.

$\mathcal{B}$ upon receiving the $\mathsf{DL}$ instance $([\![1]\!]_1, [\![1]\!]_2, [\![x]\!]_1, [\![x]\!]_2)$ interacts with $\mathcal{A}$ as follows.

- *Simulating setup.* Sample a uniform $\tau \leftarrow\!\!\$\ \mathbb{Z}_p$. Send the description of the group, along with $(\{[\![\tau^i]\!]_1\}_{i \in [n]}, \{[\![\tau^i]\!]_2\}_{i \in [2n+1] \setminus \{n+1\}})$ as the CRS.

- *Simulating key generation.*

  1. Let $\mathcal{C}$ be the set of corrupt parties.
  2. Sample $j' \leftarrow\$ [N] \setminus \mathcal{C}$ as a guess for the index for which $\mathcal{A}$ will output a non-zero coefficient.
  3. Implicitly set $\mathsf{sk}_{j'}^{-1} := x$. Next, for each $j \neq j' \in [N]$, sample $\mathsf{sk}_j \leftarrow \mathbb{Z}_p$.
  4. Sample $\varphi \leftarrow\$ \mathbb{Z}_p$ and $\theta_\omega \leftarrow\$ \mathbb{Z}_p$ for each $\omega \in [t]$. Let $\varphi := \mathsf{msk} \cdot \mathsf{sk}_{j'} = \mathsf{msk} \cdot x^{-1}$ for some $\mathsf{msk} \in \mathbb{Z}_p$, and let $\theta_\omega := \mathsf{msk}_\omega \cdot \mathsf{sk}_{j'} = \mathsf{msk}_\omega \cdot x^{-1}$ for some $\mathsf{msk}_\omega \in \mathbb{Z}_p$.
  5. For each $i \in [n]$, compute $\delta_i$ as
  $$\delta_i := (\varphi \cdot \tau^i) \cdot [\![x]\!]_1 = [\![\mathsf{msk} \cdot \tau^i]\!]_1$$
  6. For each $i \in [2n+1] \setminus \{n+1\}$, $j \in [N] \setminus \{j'\}$ and $\omega \in [t]$, compute $\gamma_{i,j}$ and $\gamma_{i,j,\omega}$ as:
  $$\gamma_{i,j} := (\mathsf{sk}_j \cdot \tau^i \cdot \varphi) \cdot [\![x]\!]_2 = [\![\mathsf{sk}_j \cdot \tau^i \cdot \mathsf{msk}]\!]_2$$
  $$\gamma_{i,j,\omega} := (\mathsf{sk}_j \cdot \tau^i \cdot \theta_\omega) \cdot [\![x]\!]_2 = [\![\mathsf{sk}_j \cdot \tau^i \cdot \mathsf{msk}_\omega]\!]_2$$
  Next, interpolate (in the exponent) the above to compute $\gamma_{i,j,\omega}$ for all $\omega \in [W]$.
  7. For each $i \in [2n+1] \setminus \{n+1\}$ and $\omega \in [t]$, compute $\gamma_{i,j'}$ and $\gamma_{i,j',\omega}$ as:
  $$\gamma_{i,j'} := (\tau^i \cdot \varphi) \cdot [\![1]\!]_2 = [\![\tau^i \cdot \mathsf{sk}_{j'} \cdot \mathsf{msk}]\!]_2$$
  $$\gamma_{i,j',\omega} := (\tau^i \cdot \theta_\omega) \cdot [\![1]\!]_2 = [\![\tau^i \cdot \mathsf{sk}_{j'} \cdot \mathsf{msk}_\omega]\!]_2$$
  Similarly as above, interpolate (in the exponent) the above to compute $\gamma_{i,j',\omega}$ for all $\omega \in [W]$.
  8. Send appropriate $\mathsf{mpk}$ and $\{\mathsf{sk}_j\}_{j \in \mathcal{C}}$ to $\mathcal{A}$.

- *Compute the challenge ciphertext* as per the protocol.

- *Simulating* $\mathcal{O}_{\mathsf{BDec}}$. Upon receiving a batch decryption query $\mathsf{ct}_i = ([\![r_i]\!]_1, \cdots)_{i \in [B]}$, verify the correctness of their $\mathsf{DLEq}$ proofs, and upon successful validation,

  1. Compute $\sigma_j$ for each $j \in [N] \setminus \{j'\}$ as per the honest protocol specification.
  2. Compute $\sigma_{j'}$ as:
  $$\sigma_{j'} := [\![x]\!]_1 \cdot \sum_{i \in [B]} r_{i,0} = [\![\mathsf{sk}_{j'}^{-1}]\!]_1 \cdot \sum_{i \in [B]} r_{i,0}$$
  Here, for each $i \in [B]$, $r_{i,0}$ be the *effective*[10] coefficient of $[\![r_i]\!]_1$ to the base $[\![1]\!]_1$.

- *Solving discrete logarithm.* Upon receiving (as part of the $\mathcal{O}_{\mathsf{BDec}}$ query) the *first* ciphertext $\mathsf{ct}$ that has a valid $\mathsf{DLEq}$ proof and an effective representation with non-zero coefficient of $[\![\mathsf{sk}_{j'}^{-1}]\!]_1$ (or equivalently $[\![x]\!]_1$), compute the discrete logarithm as follows:

  1. Let $\mathsf{ct} = ([\![r]\!]_1, \cdots)$, and $(A, \_, z) \in \mathbb{G}_1^2 \times \mathbb{Z}_p$ be the corresponding valid $\mathsf{DLEq}$ proof.
  2. Let $(a_0, a_1)$ and $(r_0, r_1)$ denote the *effective* coefficient representations of $A$ and $[\![r]\!]_1$ in the basis $([\![1]\!]_1, [\![x]\!]_1)$, respectively, i.e.,
  $$A = a_0 \cdot [\![1]\!]_1 + a_1 \cdot [\![x]\!]_1; \quad [\![r]\!]_1 = r_0 \cdot [\![1]\!]_1 + r_1 \cdot [\![x]\!]_1$$

[10] The *effective* coefficients is the coefficient representation $\mathcal{B}$ locally computes using its knowledge of $(\tau, \varphi, \{\theta_\omega\}_{\omega \in [t]}, \{\mathsf{sk}_j\}_{j \in [N] \setminus \{j'\}})$, and the randomness it uses to program random oracles. We will argue that $\mathcal{B}$ can represent all $\mathbb{G}_1$ elements it outputs to $\mathcal{A}$ in the basis $([\![1]\!]_1, [\![x]\!]_1)$.

3. Output $x'$ as the discrete logarithm of $[\![x]\!]_1$ as, where

$$x' := \frac{z - (a_0 + c \cdot r_0)}{a_1 + c \cdot z_1}, \text{ where } c = \mathsf{G}(A, [\![r]\!]_1, \cdots)$$

*Analysis.* We begin with the following simple claim, whose proof is analogous to that of Claim 1, and thus we omit it from here.

**Claim 2** (Effective representation)**.** *For any $g \in \mathbb{G}_1$ output by $\mathcal{A}$, $\mathcal{B}$ can locally represent $g$ in the basis $([\![1]\!]_1, [\![x]\!]_1)$. We refer to this as the* effective *representation of $g$.*

Note that when $a_1 + c \cdot z_1 \neq 0$, $x'$ is well defined and $x' = x$. Therefore, assuming hardness of $\mathsf{DL}$, $a_1 + c \cdot z_1 = 0$. Since we chose $c$ uniformly at random after receiving the representation of $(A, [\![r]\!]_1)$, i.e., after receiving $(a_1, r_1)$, the probability of $c = a_1/r_1$ is $1/p$, which is negligible. Thus, except with negligible probability, $(a_1, r_1) = (0, 0)$. □

**Lemma A.4.** *In hybrid $\mathbf{H}_3$, let $\mathsf{ct} = ([\![r]\!]_1, \ldots)$ be any ciphertext appearing in an input to the $\mathcal{O}_{\mathsf{BDec}}$ oracle. Assuming the hardness of discrete logarithm ($\mathsf{DL}$), if $\mathsf{ct}$ includes a valid $\mathsf{DLEq}$ proof, then all coefficient of $[\![\mathsf{msk}]\!]_1$ in the effective representation of $[\![r]\!]_1$ is zero.*

The proof of this lemma is similar to the proof of Lemma A.3.

*Proof sketch.* The proof of this lemma is similar to the proof of Lemma A.3 with the following changes:

1. $\mathcal{B}$ upon receiving the $\mathsf{DL}$ instance $([\![1]\!]_1, [\![1]\!]_2, [\![x]\!]_1, [\![x]\!]_2)$ implicitly uses $\mathsf{msk} := x$ while interactive with $\mathcal{A}$.
2. $\mathcal{B}$ samples $\tau \leftarrow\!\!\$\ \mathbb{Z}_p$ and uses it to compute the CRS.
3. $\mathcal{B}$ samples $\mathsf{msk}_\omega \leftarrow\!\!\$\ \mathbb{Z}_p$ for each $\omega \in [t]$, $\mathsf{sk}_j \leftarrow\!\!\$\ \mathbb{Z}_p$ for each $j \in [N]$ and uses them to compute $\mathsf{mpk}$.
4. $\mathcal{B}$ computes the challenge ciphertext, respond to $\mathcal{O}_{\mathsf{BDec}}$ queries, and solves $\mathsf{DL}$, using a procedure similar to the proof of Lemma A.3.

□

*Proof of Lemma 4.8.* All the $\mathbb{G}_1$ elements $\mathcal{A}$ receives in hybrid $\mathbf{H}_3$, can be represented using the basis $([\![1]\!]_1, \{[\![\tau^i]\!]_1\}_{i\in[n]}, \{\mathsf{sk}_j\}_{j\in[N]}, \{[\![\mathsf{msk} \cdot \tau_i]\!]_1\}_{i\in[n]})$. From Lemmas A.2 to A.4, for every ciphertext $\mathsf{ct}_\ell := ([\![r_\ell]\!]_1, \cdots)$ $\mathcal{A}$ queries the $\mathcal{O}_{\mathsf{BDec}}$ oracle with, their effective coefficients of $(\{[\![\tau^i]\!]_1\}_{i\in[n]}, \{\mathsf{sk}_j\}_{j\in[N]}, \{[\![\mathsf{msk} \cdot \tau_i]\!]_1\}_{i\in[n]})$ are zero. This implies that $r_\ell = r_{\ell,0}$, and hence the decryption outputs $\sigma_j$ for $j \in [N]$ matches in both hybrid $\mathbf{H}_2$ and $\mathbf{H}_2$. This implies that hybrid $\mathbf{H}_2$ and $\mathbf{H}_3$ are computationally indistinguishable. □

*Proof of Lemma 4.9.* We give a reduction to the hardness of the $n$-$\mathsf{DH}^+$ problem. To that end, suppose there exists a PPT adversary $\mathcal{A}$ that can distinguish between the two hybrids with some non-negligible advantage $\eta(\lambda)$, we construct an efficient reduction $\mathcal{B}$ that uses $\mathcal{A}$ to solve $n$-$\mathsf{DH}^+$ with probability $(1/n)\cdot\eta(\lambda)$ as follows:

- *Simulating* $\mathsf{Setup}$. The reduction algorithm $\mathcal{B}$ receives the $n$-$\mathsf{DH}^+$ challenge $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, [\![1]\!]_1, [\![1]\!]_2, [\![1]\!]_\mathsf{T}, \circ, \{[\![x^i]\!]_1\}_{i\in[-n,n]}, \{[\![x^i]\!]_2\}_{i\in[3n+1]\setminus\{n+1\}}, [\![y]\!]_1, Z)$ from the challenger, where $Z \in \mathbb{G}_\mathsf{T}$ is either $[\![x^{n+1} \cdot y]\!]_\mathsf{T}$ or uniform. $\mathcal{B}$ outputs $\mathsf{pp} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, [\![1]\!]_1, [\![1]\!]_2, [\![1]\!]_\mathsf{T}, \circ, \{\underbrace{[\![x^i]\!]_1}_{g_i}\}_{i\in[n]}, \{\underbrace{[\![x^i]\!]_2}_{h_i}\}_{i\in[2n+1]\setminus\{n+1\}})$ as the crs.
- *Simulating* $\mathsf{KeyGen}$. $\mathcal{B}$ simulates $\mathsf{KeyGen}$ as follows:
  1. Sample an index $i' \leftarrow\!\!\$\ [N]$ as its guess for $i^*$.
  2. Let $\mathcal{C}$ be the set of corrupt parties. Without loss of generality, let $\bigcup_{j\in\mathcal{C}} \Omega_j = [t]$.
  3. For each $\omega \in [t]$, sample $\mathsf{msk}_\omega \leftarrow\!\!\$\ \mathbb{Z}_p$.
  4. For each $j \in \mathcal{C}$, sample $\mathsf{sk}_j \leftarrow\!\!\$\ \mathbb{Z}_p$ and set $\gamma_{i,j,\omega} := h_{n+1+i} \cdot \mathsf{sk}_j \cdot \mathsf{msk}_\omega$ for all $i \in [n]$ and $\omega \in \Omega_j$.

5. Sample $\varphi \leftarrow\$ \mathbb{Z}_p$, and $\theta_j \leftarrow\$ \mathbb{Z}_p$ for each $j \in [N]\backslash\mathcal{C}$. For each $j \in [N]\backslash\mathcal{C}$, implicitly define $\mathsf{sk}_j := \theta_j \cdot x^n$. Also, let $\varphi := \mathsf{msk} \cdot x^{i'}$ for some $\mathsf{msk} \in \mathbb{Z}_p$.
6. For each $j \in [N] \setminus \mathcal{C}$ compute $[\![\mathsf{sk}_j^{-1}]\!]_1$ as

$$\theta_j^{-1} \cdot [\![x^{-n}]\!]_1 = (\mathsf{sk}_j^{-1} \cdot x^n)[\![x^{-n}]\!]_1 = [\![\mathsf{sk}_j^{-1}]\!]_1$$

7. For each $i \in [n]$, compute

$$\delta_i := \varphi \cdot [\![x^{i-i'}]\!]_1 = [\![\mathsf{msk} \cdot x^{i'} \cdot x^{i-i'}]\!]_1 = [\![\mathsf{msk} \cdot x^i]\!]_1 = \mathsf{msk} \cdot g_i$$

Note that $|i - i'| \leq n - 1$, and hence $\mathcal{B}$ has all $[\![x^{i-i'}]\!]_1$ terms to compute all $\delta_i$ values.
8. For each $i \in [n], j \in [N] \setminus \mathcal{C}$, compute

$$\begin{aligned}\gamma_{i,j} :=& [\![x^{2n+1+i-i'}]\!]_2 \cdot \theta_j \cdot \varphi \\ =& [\![x^{n+1+i+n-i'}]\!]_2 \cdot (\mathsf{sk}_j \cdot x^{-n}) \cdot (\mathsf{msk} \cdot x^{i'}) \\ =& [\![x^{n+1+i}]\!]_2 \cdot \mathsf{sk}_j \cdot \mathsf{msk} = h_{n+1+i} \cdot \mathsf{sk}_j \cdot \mathsf{msk}\end{aligned}$$

Note that $|i - i'| \leq n - 1$, which implies that $n + 2 \leq 2n + 1 + i - i' \leq 3n$. Hence, $\mathcal{B}$ has all $[\![x^{2n+1+i-i'}]\!]_1$ terms to compute all $\gamma_{i,j}$ values.
9. For all $i \in [n], j \in [N] \setminus \mathcal{C}$ and $\omega \in [t]$ compute

$$\begin{aligned}\gamma_{i,j,\omega} :=& [\![x^{2n+1+i}]\!]_2 \cdot \theta_j \cdot \mathsf{msk}_\omega \\ =& [\![x^{n+1+i}]\!]_2 \cdot (\theta_j \cdot x^n) \cdot \mathsf{msk}_\omega \\ =& [\![x^{n+1+i}]\!]_2 \cdot \mathsf{sk}_j \cdot \mathsf{msk}_\omega = h_{n+1+i} \cdot \mathsf{sk}_j \cdot \mathsf{msk}_\omega\end{aligned}$$

Note that $\mathcal{B}$ knows $\mathsf{msk}_\omega$ for each $\omega \in [t]$.
10. For each $i \in [n]$ and $j \in [N] \setminus \mathcal{C}$, interpolate (in the exponent) $(\gamma_{i,j}, \{\gamma_{i,j,\omega}\}_{\omega\in[t]})$ to compute:

$$\gamma_{i,j,\omega'} = h_{n+1+i} \cdot \mathsf{sk}_j \cdot \mathsf{msk}_{\omega'} \quad \forall \omega' \in \Omega_j$$

11. Send appropriate $\mathsf{mpk}$ and $\{\mathsf{sk}_j\}_{j\in\mathcal{C}}$ to $\mathcal{A}$.

- *Computing challenge ciphertext.* Upon receiving the challenge messages $(\mathsf{m}_0, \mathsf{m}_1, i^*)$ from $\mathcal{A}$, abort if $i' \neq i^*$. Otherwise, sample $b \leftarrow\$ \{0, 1\}$ and compute $\mathsf{ct}^*$ as:

$$\mathsf{ct}^* := ([\![y]\!]_1, \varphi \cdot [\![y]\!]_1, \mathsf{m}_b \oplus \mathsf{H}(\varphi \cdot Z), \pi)$$

here $\pi$ is the simulated NIZK proof. Note that $\varphi \cdot [\![y]\!]_1 = (\mathsf{msk} \cdot x^{i^*}) \cdot [\![y]\!]_1$.

- *Simulating $\mathcal{O}_{\mathsf{BDec}}$ queries.* Upon receiving a batch decryption query $\{\mathsf{ct}_i = ([\![r_i]\!]_1, \cdots)\}_{i\in[B]}$:
  1. Run the extractor $\mathcal{E}_\mathcal{A}$ for $\mathsf{DLEq}$ on each of the proofs $\pi_i^{\mathsf{client}}$ and obtain witness $r_i$.
  2. For any $j \in \mathcal{C}$, compute $\sigma_j := \mathsf{sk}_j^{-1} \cdot \left(\sum_{i\in[B]} [\![r_i]\!]_1\right)$.
  3. For any $j \in [N] \setminus \mathcal{C}$, compute $\sigma_j$ as:

$$\begin{aligned}\sigma_j :=& \theta_j^{-1} \cdot [\![x^{-n}]\!]_1 \cdot \sum_{i\in[B]} r_i \\ =& \left(\mathsf{sk}_j^{-1} \cdot x^n\right) [\![x^{-n}]\!]_1 \cdot \sum_{i\in[B]} r_i = [\![\mathsf{sk}_j^{-1}]\!]_1 \cdot \sum_{i\in[B]} r_i\end{aligned}$$

- *Guessing* $n$-$\mathsf{DH}^+$. When $\mathcal{A}$ outputs its guess bit $b'$, indicating whether the experiment corresponds to $\mathbf{H}_3$ or $\mathbf{H}_4$, output bit $b'$ as the guess to the $n$-$\mathsf{DH}^+$ challenger.

*Analysis.* Note that when $i' = i^*$ and $Z = [\![x^{n+1} \cdot y]\!]_{\mathsf{T}}$, we have $\mathsf{ct}_3^*$ as:

$$\begin{aligned}\mathsf{ct}_3^* &= \mathsf{m}_b \oplus \mathsf{H}(\varphi \cdot Z) \\ &= \mathsf{m}_b \oplus \mathsf{H}(\mathsf{msk} \cdot x^{i'} \cdot [\![x^{n+1} \cdot y]\!]_{\mathsf{T}}) \\ &= \mathsf{m}_b \oplus \mathsf{H}(\mathsf{msk} \cdot [\![y]\!]_1 \circ [\![x^{n+1+i'}]\!]_2)\end{aligned}$$

which is a valid encryption of $m_b$ with $y$ as its randomness. Hence, the interaction of $\mathcal{A}$ in this scenario is identically distributed as in $\mathbf{H}_3$.

Alternatively, when $Z$ is a uniform element in $\mathbb{G}_{\mathsf{T}}$, $Z$ is identically distributed as $[\![\rho]\!]_{\mathsf{T}}$ for uniformly random $\rho \leftarrow\!\!\$\ \mathbb{Z}_p$. Hence, when $i' = i^*$ and $Z$ is uniformly random in $\mathbb{G}_{\mathsf{T}}$, $\mathcal{A}$'s interaction with $\mathcal{B}$ is identically distributed as in the hybrid $\mathbf{H}_4$.

Since $\Pr[i^* = i'] = 1/n$, we get that if $\mathcal{A}$ distinguishes the two hybrids with probability $\eta(\lambda)$, then $\mathcal{B}$ can break the $n$-$\mathsf{DH}^+$ assumption with probability at least $(1/n) \cdot \eta(\lambda)$. Therefore, we have

$$\left|\mathsf{Adv}^4_{\mathcal{A}}(\lambda) - \mathsf{Adv}^3_{\mathcal{A}}(\lambda)\right| \leq n \cdot \mathsf{Adv}^{n\text{-}\mathsf{DH}^+}_{\mathcal{B}}(\lambda)\ . \qquad \square$$

# B Alternative construction of Weighted BTE scheme

Recall from our discussion in Section 2.3 where we mentioned that a seemingly straightforward way to design a weighted version of $\mathsf{BEAT\text{-}MEV}^{++}$, by simply instantiating the (unbatched) thresholdizable homomorphic public-key encryption scheme $\mathsf{THE}$ used in $\mathsf{BEAT\text{-}MEV}^{++}$ with a *weighted* (unbatched) $\mathsf{THE}$ scheme, say Ferveo [BO22], doesn't work because of a type mismatch issue. Specifically the PRF key in $\mathsf{BEAT\text{-}MEV}^{++}$ resides in the *source* group and is encrypted using $\mathsf{THE}$ (see Eq. (1)). Consequently, the message space of $\mathsf{THE}$ needs to be the *source* group – something that Ferveo does not satisfy as the message space in Ferveo is the *target* group. To fix this type mismatch, we proposed an approach in Section 2.3 where we algebraically coupled the PRF evaluation in $\mathsf{BEAT\text{-}MEV}^{++}$ with the decryption protocol of Ferveo in a non black-box manner. In this section, we will provide an alternative construction where we fix this type match in a different way which doesn't require non black-box coupling of the unbatched weighted $\mathsf{THE}$ scheme Ferveo with the PRF used in $\mathsf{BEAT\text{-}MEV}^{++}$. This construction is simpler to describe, has a smaller CRS size but a much higher ciphertext size and encryption time (as compared to our main construction).

The high-level idea of fixing the type mismatch issue in this alternative construction is the following. Suppose we have a target group element $[\![u]\!]_{\mathsf{T}}$ and we would like to convert it into its source group representation $[\![u]\!]$. One way to do this is to simply compute the discrete logarithm of $[\![u]\!]_{\mathsf{T}}$ (w.r.t to the generator $[\![1]\!]_{\mathsf{T}}$) to retrieve the scalar $u \in \mathbb{Z}_p$ and then derive $[\![u]\!] \coloneqq u \cdot [\![1]\!]$ in the source group. However, solving discrete logarithm efficiently in $\mathbb{G}_T$ is not possible when $u$ is a random element in $\mathbb{Z}_p$ without contradicting the hardness assumptions that underlies the security of the scheme. To get around this, we take the following approach: Split the scalar $u$ having $\lambda' = O(\lambda)$ bits into $m$ chunks $u_1, \ldots, u_m$ representing the radix $D = 2^{\lceil \lambda'/m \rceil}$ decomposition of $u$, where $u_1$ and $u_m$ are the LSB and MSB respectively. Then, instead of encoding the scalar $u \in \mathbb{Z}_p$ directly in the target group, we will encode its radix $R$ decomposition digits individually in the target group as $[\![u_1]\!]_{\mathsf{T}}, \ldots, [\![u_m]\!]_{\mathsf{T}}$. Since each $u_i \in [0, D)$, solving discrete logarithm of each $[\![u_i]\!]_{\mathsf{T}}$ will be efficient as long as $D = \mathsf{poly}(\lambda)$. When this is the case, we can efficiently solve discrete logarithm of each $[\![u_i]\!]_{\mathsf{T}}$ to retrieve $u_i$, recompose all $u_i$'s to get $u$ and then convert it into the source group representation $[\![u]\!]$. Increasing the number of chunks $m$ will increase the ciphertext size and decrease the time required for discrete logarithm computation, hence it needs to be tuned based on the specific application requirements.

We describe the formal construction in Fig. 3. The ciphertext $\mathsf{ct}$ consists of $\mathsf{ct}_1$ and $\mathsf{ct}_2$ where $\mathsf{ct}_1$ is the Ferveo style elgamal encryption of each chunk of the PRF key $r \in \mathbb{Z}_p$ and $\mathsf{ct}_2$ is the punctured version of $r$. Since we need to encrypt each chunk individually, the size of $\mathsf{ct}_1$ grows linearly with the number of chunks. To reduce this size, we use a known optimization from [Kur02] which allows us to reuse the randomness in Elgamal ciphertexts at the cost of using a separate public key for each ciphertext where the randomness is reused thereby increasing the CRS size by a multiplicative factor equal to the number of chunks. This optimization also makes the server communication to be independent of the number of chunks. The ciphertext size can be further optimized by encoding the second part of the elgamal ciphertexts in $\mathbb{G}_1$ instead of $\mathbb{G}_T$ by changing $[\![\mathsf{msk}^i]\!]_{\mathsf{T}}$ to $[\![\mathsf{msk}^i]\!]_1$ in $\mathsf{mpk}$. For this to be secure, we will need the absence of efficiently computable homomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$ (e.g. Type 3 pairing). Alternatively, the technique from [BCF+25] can be used to securely encode a $\mathbb{G}_T$ element as a *pair* of base group elements without any restriction to the type of pairing group. The client NIZK proof, in addition to proving the correctness of elgamal ciphertexts encrypting each chunk of the PRF key $r$ and the correctness of punctured key, requires proving that each chunk $r_i$ is within the range of radix $D$ being used. Concretely, this can be instantiated using efficient range proofs such as [BBB+18]. Alternatively, the entire NIZK (including range checks, elgamal correctness and punctured key correctness) can instantiated using [Gro16] to achieve the minimum possible overhead in ciphertext size. We leave these optimizations to future work.

**Requirements:**

- $N$ is the number of servers, $t$ is the corruption threshold, $w_j$ is the weight of the $j^{\text{th}}$ server and $W := \sum_{i\in[N]} w_i$.
- $B_{\mathsf{max}}$ is the maximum batch size, $B$ is the batch size, $n$ is the index space, $m$ is the number of chunks.
- Digit decomposition and recomposition algorithms $(\mathsf{Split}, \mathsf{Comb})$ w.r.t. a fixed radix $D$
- A hash function $\mathsf{H} : \mathbb{G} \to \{0,1\}^*$
- Shamir secret sharing scheme having algorithms $\mathsf{Share}$ and $\mathsf{Lagrange}$
- A NIZK system $\mathsf{NIZK} = (\mathsf{Prove}, \mathsf{Verify})$ for proving an arbitrary NP relation.
- A NIZK system $\mathsf{DLEq} = (\mathsf{Prove}, \mathsf{Verify})$ to prove DL equality w.r.t. an arbitrary group (Refer to [CP93]).

---

$\underline{\mathsf{Setup}(1^\lambda, 1^{B_{\max}})}$:

*// Defining the size of index space $n$*

$n := B_{\mathsf{max}}$
$\mathcal{G} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, [\![1]\!]_1, [\![1]\!]_2, [\![1]\!]_\mathsf{T}, p, \circ) \leftarrow \mathsf{GrpGen}(1^\lambda)$
$\tau \leftarrow\$ \mathbb{Z}_p$
**return** $\mathsf{pp} := n, \mathcal{G}, \{[\![\tau^i]\!]_1\}_{i\in[n]}, \{[\![\tau^i]\!]_2\}_{i\in[2n+1]\setminus\{n+1\}}$

$\underline{\mathsf{KeyGen}(1^\lambda, \{w_j\}_{j\in[N]}, t, \mathsf{pp})}$:

*// Defining the secret sharing evaluation domain $\Omega$*

$W := \Sigma_{j\in[N]} w_j$ and $\Omega \subseteq \mathbb{Z}_p$ s.t. $|\Omega| = W$
Partition $\Omega$ into $\Omega_1, \ldots, \Omega_N$ s.t. $|\Omega_j| = w_j \ \forall j \in [N]$
**for** all $i \in [m], \mathsf{msk}^i \leftarrow\$ \mathbb{Z}_p$
**for** all $i \in [m], \{\mathsf{msk}^i_\omega\}_{\omega\in\Omega} \leftarrow \mathsf{Share}(\mathsf{msk}^i, W, t, \Omega)$
**for** all $j \in [N], \mathsf{sk}_j \leftarrow\$ \mathbb{Z}_p$
$\mathsf{mpk} := \begin{cases} \{\underbrace{[\![1]\!]_2 \cdot \mathsf{sk}_j \cdot \mathsf{msk}^i_\omega}_{\gamma_{i,j,\omega}}\}_{i\in[m],j\in[N],\omega\in\Omega_j} \\ \{[\![\mathsf{msk}^i]\!]_\mathsf{T}\}_{i\in[m]}, \{[\![\mathsf{sk}_j]\!]_1\}_{j\in[N]} \end{cases}$
**return** $\mathsf{mpk}, \{\mathsf{sk}_j\}_{j\in N}$

$\underline{\mathsf{Enc}(\mathsf{mpk}, \mathsf{m}, i^*)}$:

$r \leftarrow\$ \mathbb{Z}_p$
$r_1, \ldots, r_m := \mathsf{Split}(r, m)$ *// Split $r$ into $m$ chunks*
$s \leftarrow \mathbb{Z}_p$
$\mathsf{ct}_1 := \underbrace{[\![s]\!]_1}_{\mathsf{ct}_1^{\mathsf{first}}}, \{\underbrace{[\![r_i]\!]_\mathsf{T} + s \cdot [\![\mathsf{msk}^i]\!]_\mathsf{T}}_{\mathsf{ct}_1^{i,\mathsf{second}}}\}_{i\in[m]}$
$\mathsf{ct}_2 := r \cdot g_{i^*}$
$\chi^{\mathsf{client}} := (\mathsf{pp}, \mathsf{mpk}, i^*, \mathsf{ct}_1, \mathsf{ct}_2)$
$\mathsf{wit}^{\mathsf{client}} := (s, r_1, \ldots, r_m)$
$\pi^{\mathsf{client}} \leftarrow \mathsf{NIZK.Prove}\left\{ \begin{array}{l|l} \chi^{\mathsf{client}}, & (\chi^{\mathsf{client}}, \mathsf{wit}^{\mathsf{client}}) \\ \mathsf{wit}^{\mathsf{client}} & \in \mathcal{R}_\mathcal{L} \end{array} \right\}$

where $\mathcal{R}_\mathcal{L} := \left\{ \begin{array}{l|l} & \mathsf{ct}_1^{\mathsf{first}} = s \cdot [\![1]\!]_1, \\ & \mathsf{ct}_2 = r \cdot g_{i^*}, \\ \chi^{\mathsf{client}}, & r = \mathsf{Comb}(r_1, \ldots, r_m), \\ \mathsf{wit}^{\mathsf{client}} & \forall i \in [m] : r_i \in [0, D), \\ & \forall i \in [m] : \ \mathsf{ct}_1^{i,\mathsf{second}} = \\ & r_i \cdot [\![1]\!]_\mathsf{T} + s \cdot [\![\mathsf{msk}^i]\!]_\mathsf{T} \end{array} \right\}$

$\mathsf{ct} := \left(\mathsf{ct}_1, \mathsf{ct}_2, i^*, \mathsf{m} \oplus \mathsf{H}(r \cdot [\![1]\!]_1 \circ h_{n+1+i^*}), \pi^{\mathsf{client}}\right)$
**return** $\mathsf{ct}$

$\underline{\Pi\text{-}\mathsf{BDec}_1(\mathsf{mpk}, \{\mathsf{ct}_\ell\}_{\ell\in[B]}, \langle \mathsf{sk}_j \rangle)}$:

**for** all $\ell \in [B]$, **parse** $\mathsf{ct}_\ell$ as $(\mathsf{ct}_{\ell,1}, \ldots, \mathsf{ct}_{\ell,4}, \pi^{\mathsf{client}}_\ell)$
$U := [B]$
*// $U$ includes indices of only valid ciphertexts*
**for** $\ell \in [B], \chi^{\mathsf{client}}_\ell := (\mathsf{pp}, \mathsf{mpk}, \mathsf{ct}_{\ell,3}, \mathsf{ct}_{\ell,1}, \mathsf{ct}_{\ell,2})$
$U := \{\ell \mid \ell \in [B] \wedge \mathsf{NIZK.Verify}(\chi^{\mathsf{client}}_\ell, \pi^{\mathsf{client}}_\ell) = 1\}$
$\sigma_j := \mathsf{sk}_j^{-1} \cdot \Sigma_{\ell\in U} \mathsf{ct}^{\mathsf{first}}_{\ell,1}$
*// NIZK proof $\pi^{\mathsf{server}}_j$ for the correctness of $\sigma_j$*
$\chi^{\mathsf{server}}_j := ([\![1]\!]_1, \sigma_j, [\![\mathsf{sk}_j]\!]_1, \Sigma_{\ell\in U} \mathsf{ct}^{\mathsf{first}}_{\ell,1})$
$\pi^{\mathsf{server}}_j \leftarrow \mathsf{DLEq.Prove}(\chi^{\mathsf{server}}_j, \mathsf{sk}_j)$
**return** $\sigma_j, \pi^{\mathsf{server}}_j$

$\underline{\Pi\text{-}\mathsf{BDec}_2(\{\sigma_j, \pi^{\mathsf{server}}_j\}_{j\in[N]})}$:

$S := [N]$
*// Update $S$ to include the indices of only those servers who send a valid message*
$S := \{j \mid j \in S \wedge \mathsf{DLEq.Verify}(\chi^{\mathsf{server}}_j, \pi^{\mathsf{server}}_j) = 1\}$
*// Select $V \subseteq S$ of servers whose combined weight exceeds the corruption threshold $t$*
Select $V \subseteq S$ s.t. $\sum_{j\in V} w_j > t$
*// Defining the share interpolation domain $\Omega'$ and lagrange coefficients $L_\omega$*
$\Omega' := \bigcup_{j\in V} \Omega_j$ and $\{L_\omega\}_{\omega\in\Omega'} := \mathsf{Lagrange}(\Omega', t)$
*// Reconstruct $\mu := \sum_{\ell\in U} r_\ell$*
**for** $i \in [m], R_i := (\sum_{\ell\in U} \mathsf{ct}^{i,\mathsf{second}}_{\ell,1}) - \sum_{j\in V} \sum_{\omega\in\Omega_j} \sigma_j \circ \gamma_{i,j,\omega} \cdot L_\omega$
**for** $i \in [m], \mu_i := \mathsf{DiscreteLog}(R_i)$
$\mu := \mathsf{Comb}(\mu_1, \ldots, \mu_m)$
*// Defining the set $E \subseteq [n]$ of indices used by the ciphertext batch*
$E := \{\mathsf{ct}_{\ell,3}\}_{\ell\in U}$
**for** *all* $i \in E, \alpha_i := \mu \cdot [\![1]\!]_1 \circ h_{n+1+i}$
**for** *all* $i \in E, \beta_i := \sum_{\ell\in U, \mathsf{ct}_{\ell,3}\neq i} \mathsf{ct}_{\ell,2} \circ h_{n+1+i-\mathsf{ct}_{\ell,3}}$
*// Decrypting all the valid ciphertexts*
**for** *all* $\ell \in U, \widetilde{\mathsf{m}}_\ell := \mathsf{ct}_{\ell,4} \oplus \mathsf{H}(\alpha_{\mathsf{ct}_{\ell,3}} - \beta_{\mathsf{ct}_{\ell,3}})$
*// Default output for all the invalid ciphertexts*
**for** *all* $\ell \in [B] \setminus U, \widetilde{\mathsf{m}}_\ell := \perp$
**return** $\{\widetilde{\mathsf{m}}_\ell\}_{\ell\in[B]}$

Figure 3: Alternative construction of wBTE scheme with parts needs for robustness and CCA security (against any number of malicious clients and upto $t$ malicious servers) highlighted in blue.