BTX : Simple and Efficient Batch Threshold Encryption

Amit Agarwal, Sourav Das, Babak Poorebrahim Gilkalaye, Peter Rindal, and Victor Shoup

Category Labs

Abstract

Batched threshold encryption (BTE) enables a committee of servers to jointly decrypt any chosen subset of ciphertexts from a large pool, while all remaining ciphertexts stay private. BTE is a key building block for *encrypted mempools*, where transactions are encrypted until block inclusion to mitigate *maximal extractable value* (MEV). Existing epochless BTE constructions either require user-chosen ciphertext indices that create coordination and censorship concerns or are computationally inefficient.

In this paper, we present BTX, a simple and concretely efficient BTE construction that is both epochless and collision-free: encryption does not require a user-chosen batch index. Our scheme achieves the shortest ciphertext size among all known BTE constructions having the same size as a standard elgamal ciphertext. By making the scheme amenable to FFT, we reduce the decryption cost to $O(B \log B)$ group exponentiations and $O(B)$ pairings, where B is the size of the dynamically chosen batch of ciphertexts.

We implement BTX and two baselines in a shared, aggressively optimized C++ codebase over BLS12-381 with AVX-512 vectorization, FFT-based backends where applicable, and additional low-level engineering throughout. At batch size $B = 512$, using a single core, BTX requires approximately 598 ms total for decryption, compared with 1197 ms for the FFT optimized version of partial-fraction evaluation baseline [7], an overall 2.0× improvement.

1 Introduction

Batched threshold encryption (BTE) enables a committee of servers to jointly decrypt an arbitrarily chosen subset (batch) of ciphertexts from a large pool, while all remaining ciphertexts stay private [11, 2, 10, 9]. The defining feature of BTE is scalability: the total server-to-server communication required for decryption grows sublinearly in the batch size and is independent of the overall pool size. This makes BTE a natural building block for *encrypted mempools*, where transactions are encrypted until they are included in a block. This is a key approach to mitigate *maximal extractable value* (MEV) [18, 26, 23]. Practical encrypted mempools operate under tight latency budgets. Reducing both the communication and the computation of batch decryption while supporting large batch sizes is therefore an important problem.

Recent work [11, 12, 2, 10, 9, 1, 27, 4] makes substantial progress toward this goal. Choudhuri et al. [11] introduced BTE and achieved low decryption communication, but their construction requires an MPC-based setup for each block. Later schemes [12, 2] remove this repeated setup cost. However, they require each ciphertext to be generated with respect to a unique decryption session ID; and each session can have a single batch decryption. In the encrypted mempool setting, the decryption session ID corresponds to a block ID, so each block has its own mempool of pending transactions and the transactions cannot roll over to the next block if they are not included. This is undesirable: it reduces flexibility and creates avenues for censorship.

BEAT-MEV [10] presents an alternative construction where ciphertexts are not tied to a decryption session. However, BEAT-MEV suffers from a collision problem: each ciphertext carries an index from a small namespace, and each decryption session can decrypt at most one ciphertext per index. Follow-up works [9, 1] improve the computation cost and support non-interactive setup, respectively, while retaining the same collision issues.

Two other works, Fernando et al. [15] and Boneh et al. [7], address all the above issues. However, Fernando et al. [15] require a large CRS proportional to the number of batches (decryption sessions) and a higher asymptotic computation cost (see Table 1), and Boneh et al. [7] incur higher concrete computation cost (see §8). We elaborate on related work in §2.

In this paper, we present BTX, a simple and concretely efficient BTE construction that addresses all the above issues while maintaining high efficiency. Similar to [7], to achieve these results we increase the secret key size of each decryptor to linear in the batch size. In contrast to [7], our scheme is conceptually much simpler without needing any partial-fraction-based techniques and also yielding better concrete costs (in terms of ciphertext size and decryption computation). A second key idea is to make the scheme amenable to *Fast Fourier Transform* (FFT) acceleration. We elaborate on these ideas in Section 3.

We summarize our contributions below.

- *A collision-free and epochless BTE.* We present a new BTE scheme where ciphertexts are not tied to any epoch (decryption session) and are collision-free, i.e., encryption does not require a user-chosen index.

- *Quasi-linear decryption with dynamic batch sizing.* We improve the computation cost of our scheme by making it amenable to FFT/NTT, and achieve a decryption cost of $O(B \log B)$ group exponentiations and $O(B)$ pairings. Here, B is the actual batch size. This is an improvement over the prior-best cost of $O(B_{\mathsf{max}} \log B_{\mathsf{max}})$ exponentiations and $O(B_{\mathsf{max}})$ pairings [1], where $B_{\mathsf{max}} \geq B$ is a protocol parameter indicating the maximum supported batch size.

- *Shortest ciphertext size.* Our scheme achieves the shortest ciphertext size compared to all prior schemes. The ciphertext in our scheme has exactly the same size as a standard elgamal encryption - one additional source group element besides the usual additive overhead due to message length and Schnorr-based NIZK proof (for CCA security).

- *Evaluation.* We implement BTX, BEAT++, and the partial-fraction evaluation baseline (PFE) in a shared, aggressively optimized C++ codebase with AVX-512 vectorization [22] over BLS12-381. This includes FFT-based backends where applicable, optimized MSM/multipairing paths, and additional micro-optimizations across the stack, so the comparison is against best-effort implementations of the strongest prior protocols rather than against lightly optimized reference code. At batch size $B = 512$, using a single core, BTX requires ≈ 598 ms total for decryption, compared with 1197 ms for PFE— an overall 2.0× improvement.

Paper organization. The rest of the paper is organized as follows. We discuss related work in §2 and present a technical overview of BTX in §3. We describe preliminaries and formalize the BTE definitions in §4. We present our construction in detail in §5 and describe our computational optimizations in §6. We analyze correctness and security in §7 and present our evaluation in §8.

2 Related Work

We summarize our comparison with prior work in Table 1 and describe them next.

Table 1: Comparison of BTE schemes. Here, B denotes the actual batch size and B_{max} denotes the maximum supported batch size. CR denotes "censorship resistance" and signifies that the scheme is index-independent and not vulnerable to censorship attack due to index collision among clients generating ciphertexts. The decryption complexity is in terms of number of group operations.

Scheme	CR	Epochless	Decryption complexity	Ciphertext size
Batched IBE [1]	✓	✗	$O(B \log^2 B)$	$3\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$
	✗	✗	$O(B_{\mathsf{max}} \log B_{\mathsf{max}})$	$3\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$
Fernando et al. [15]	✓	✓	$O(B \log^2 B)$	$2\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$
BEAT-MEV [10]	✗	✓	$O(B^2)$	$3\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$
Gong et al. [19]	✓	✗	$O(B \log^2 B)$	$2\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$
	✗	✗	$O(B_{\mathsf{max}} \log B_{\mathsf{max}})$	$2\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$
Agarwal et al. (BEAT++) [1]	✗	✓	$O(B_{\mathsf{max}} \log B_{\mathsf{max}})$	$2\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$
Boneh et al. (PFE) [7]	✓	✓	$O(B_{\mathsf{max}} \log B_{\mathsf{max}})^*$	$2\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$
Our scheme (BTX)	✓	✓	$O(B \log B)$	$\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$

*In the original presentation of Boneh et al., the decryption complexity is $O(B^2)$. However, we observe that their computation can be implemented in $O(B_{\mathsf{max}} \log B_{\mathsf{max}})$ time using FFT techniques. This optimization has a larger constant factor and is about 2× slower than our scheme in practice; see Table 2 for a detailed comparison.

Threshold encryption for mempool privacy. A natural approach to mempool privacy is to encrypt transactions under a threshold encryption scheme, where the decryption key is secret-shared among the validator set [3, 21]. However, this approach incurs a communication cost of $O(N \cdot B)$ for decrypting B ciphertexts with a committee of N servers. For large batches and large validator sets, this cost is prohibitive.

Mempool privacy using threshold identity-based encryption. Shutter [14] and Fairblock [24] use *threshold identity-based encryption* (IBE) [6] to encrypt transactions for a specific epoch (block). Users encrypt their transactions to a target block using an IBE scheme, and the decryptors release an $O(1)$-size decryption key for that epoch, which enables decryption of all ciphertexts encrypted to that epoch. This achieves $O(N)$ total communication cost. However, releasing the epoch key decrypts *all* ciphertexts in that epoch, so ciphertexts not included in the batch lose their privacy. Dötting et al. [13] present a similar approach with the same all-or-nothing privacy limitation.

Epoch-bound BTE. Choudhuri et al. [11] introduced *batched threshold encryption* (BTE), achieving $O(N)$ total decryption communication while preserving the privacy of ciphertexts outside the selected batch. However, their construction requires a separate setup for every decryption session (i.e., every block in the mempool application). Concurrent works [12, 1, 27] remove the per-block setup requirement while achieving $O(N)$ total communication cost. Moreover, these constructions incur $O(B_{\mathsf{max}} \log B_{\mathsf{max}})$ group exponentiations and $O(B_{\mathsf{max}})$ pairings to decrypt any batch of $B \leq B_{\mathsf{max}}$ ciphertexts. However, both approaches bind each ciphertext to a specific decryption session or batch identity (block or epoch label). A ciphertext can be decrypted only within its designated session. In the encrypted mempool setting, this gives each transaction only one inclusion opportunity, limiting flexibility and creating avenues for censorship. Very recently, [19] give a BTE construction in the plain model with cost similar to prior works.

Epochless BTE. BEAT-MEV [10] introduced the first epochless BTE with concrete efficiency and no ciphertext-to-block binding, while requiring only a reusable one-time setup. However, BEAT-MEV requires each ciphertext to carry a user-chosen index from a small set, with the constraint

that each decryption session can decrypt at most one ciphertext per index. This creates a coordination problem and a corresponding censorship surface. BEAST-MEV [9] extends BEAT-MEV with silent (non-interactive) setup. Agarwal et al. [1] extend BEAT-MEV to support weighted validators and lower the computation cost to $O(B_{\mathsf{max}} \log B_{\mathsf{max}})$ exponentiations and $O(B_{\mathsf{max}})$ pairings, compared with $O(B^2)$ pairings in BEAT-MEV. Both of these works retain the same indexed design and collision issue.

Collision-free BTE. Fernando et al. [15] and Boneh et al. [7] each address both the collision and session-binding issues. Fernando et al. [15] achieve collision-free BTE but require a CRS whose size grows with the number of decryption sessions, which limits long-lived deployments, and a higher asymptotic computation cost (see Table 1). Boneh et al. [7] present a collision-free BTE based on partial-fraction techniques. Their construction avoids the large-CRS limitation but incurs higher concrete computation cost (see §8 for a concrete comparison).

Other research. Bhattacharyya et al. [4] study traceable threshold encryption for mempool privacy, focusing on accountability of colluding decryptors rather than selective batch decryption with pending transaction privacy.

3 Technical Overview

Notation. We use $[\![x]\!]_1, [\![x]\!]_2, [\![x]\!]_T$ to denote group elements in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ respectively, and $\circ$ for the bilinear pairing, so that $[\![a]\!]_1 \circ [\![b]\!]_2 = [\![ab]\!]_T$. Let B_{max} denote the maximum supported batch size and $B \le B_{\mathsf{max}}$ the actual batch size in the current decryption session. We summarize our construction in Fig. 2 (single-server setting) and Fig. 3 (threshold setting), and describe the key ideas next. For better exposition, we present the construction in three steps.

Step 1 — A single-server BTE scheme. We begin with the simplest setting: a single server holds the full secret key and decrypts the entire batch. This version illustrates the core algebraic mechanism without threshold machinery. The protocol we describe below is only CPA secure. We describe the CCA secure scheme later.

Setup and encryption. The secret key sk is a scalar $\tau \leftarrow\$ \mathbb{Z}_p$ and the public key is a *punctured powers-of-τ* where dk below is the public decryption key and ek is an encryption key:

$$\mathsf{dk} = \{\underbrace{[\![\tau^i]\!]_2}_{h_i}\}_{i \in [2B_{\mathsf{max}}] \setminus \{B_{\mathsf{max}}+1\}}, \qquad \mathsf{ek} = [\![\tau^{B_{\mathsf{max}}+1}]\!]_T.$$

That is, all powers $\tau, \tau^2, \ldots, \tau^{2B_{\mathsf{max}}}$ in $\mathbb{G}_2$ are published except the middle power $\tau^{B_{\mathsf{max}}+1}$. The corresponding target-group element $\mathsf{ek} = [\![\tau^{B_{\mathsf{max}}+1}]\!]_T$ serves as the public encryption key.

To encrypt a message $m \in \mathbb{G}_T$, the sender samples $r \leftarrow\$ \mathbb{Z}_p$ and outputs

$$\mathsf{ct} = \big(\underbrace{[\![r]\!]_1}_{\mathsf{ct}_1}, \ \underbrace{m + r \cdot \mathsf{ek}}_{\mathsf{ct}_2}\big).$$

This is an additive ElGamal ciphertext in $\mathbb{G}_T$: the encryption pad is $r \cdot \mathsf{ek} = [\![r \cdot \tau^{B_{\mathsf{max}}+1}]\!]_T$. We emphasize that encryption does not require a batch index. This is a key difference from [10, 1] where each ciphertext carries a user-chosen index.

Batch decryption. Given an ordered batch $(\mathsf{ct}_1, \ldots, \mathsf{ct}_B)$, the server uses its knowledge of τ to compute the *batch secret key*:

$$\sigma = \sum_{\ell=1}^{B} \tau^\ell \cdot \mathsf{ct}_{\ell,1} = \sum_{\ell=1}^{B} [\![r_\ell \cdot \tau^\ell]\!]_1.$$

This is a single multi-scalar multiplication of size B, costing $O(B)$ scalar multiplications in $\mathbb{G}_1$. The value σ compactly encodes the contribution of the entire batch.

To open slot ℓ, the server pairs σ with the public element $h_{B_{\mathsf{max}}-\ell+1} = [\![\tau^{B_{\mathsf{max}}-\ell+1}]\!]_2$:

$$\alpha_\ell := \sigma \circ h_{B_{\mathsf{max}}-\ell+1} = \sum_{k=1}^{B} [\![r_k \cdot \tau^{k+B_{\mathsf{max}}-\ell+1}]\!]_T.$$

This sum contains the desired term $[\![r_\ell \cdot \tau^{B_{\mathsf{max}}+1}]\!]_T$ (when $k = \ell$), which is exactly the encryption pad $r_\ell \cdot \mathsf{ek}$ for message m_ℓ. However, it also contains cross-terms from every other ciphertext in the batch.

The key observation is that these cross-terms are publicly computable. Define

$$\beta_\ell := \sum_{\substack{i=1 \\ i\neq\ell}}^{B} \mathsf{ct}_{i,1} \circ h_{B_{\mathsf{max}}-\ell+i+1} = \sum_{\substack{i=1 \\ i\neq\ell}}^{B} [\![r_i \cdot \tau^{B_{\mathsf{max}}-\ell+i+1}]\!]_T.$$

The exponent $B_{\mathsf{max}} - \ell + i + 1$ equals $B_{\mathsf{max}} + 1$ only when $i = \ell$. Since the sum excludes $i = \ell$, every required element $h_{B_{\mathsf{max}}-\ell+i+1}$ is present in the CRS. The omitted middle power is precisely what separates the desired term from the cross-terms. Subtracting gives

$$\alpha_\ell - \beta_\ell = [\![r_\ell \cdot \tau^{B_{\mathsf{max}}+1}]\!]_T = r_\ell \cdot \mathsf{ek},$$

and the server recovers $m_\ell = \mathsf{ct}_{\ell,2} - (\alpha_\ell - \beta_\ell)$.

Cost. Computing the α terms requires B pairings (one per slot). Each β_ℓ term sums over $B-1$ pairings, so the total cost is $B + B(B-1) = O(B^2)$ pairings.

Step 2 — Reducing the cross-term cost to $O(B \log B)$. The $O(B^2)$ cost comes entirely from the β_ℓ computation. The key observation is that β_ℓ depends only on the *relative offset* $i - \ell$: the CRS element used is $h_{B_{\mathsf{max}}+(i-\ell)+1}$. Consequently, the vector $(\beta_1, \ldots, \beta_B)$ is not an arbitrary quadratic collection of terms; it is a contiguous coefficient window of a bilinear polynomial product — a *middle-product* computation in the classical sense of Hanrot et al. [20].

Based on this observation regarding the middle product, FFT acceleration is natural. One input polynomial comes from the ciphertext batch and changes from block to block; the other is determined entirely by the fixed CRS powers and can be preprocessed once. The result is an implementation of the cross-term computation using $O(B \log B)$ group exponentiations and only $O(B)$ pairings, reducing the dominant cost from quadratic to quasi-linear. We describe the formal reduction in §6.

This structure also yields a clean latency decomposition. The β_ℓ values depend only on the ciphertext batch and not on the batch secret key σ. Therefore, the entire cross-term computation can run as a Precompute step as soon as the batch is fixed. After σ is computed, the remaining Open step requires only B pairings for the α_ℓ terms, followed by lightweight subtraction and unmasking. Moreover, the FFT size adapts to the actual batch size B, not the provisioned maximum B_{max}, so smaller batches incur proportionally less work.

Step 3 — Extension to threshold decryption. The single-server scheme requires the server to know τ. In a decentralized setting, no single party should hold τ.

The extension to a threshold setting is immediate. The batch secret key $\sigma = \sum_{\ell=1}^{B} \tau^\ell \cdot \mathsf{ct}_{\ell,1}$ is a *linear function* of $(\tau, \tau^2, \ldots, \tau^{B_{\mathsf{max}}})$ for any fixed batch of ciphertexts. If we Shamir-share each

power τ^ℓ among N servers with reconstruction threshold $t+1$, then each server j can compute its share of σ locally:

$$\sigma_j = \sum_{\ell=1}^{B} \langle \tau^\ell \rangle_j \cdot \mathsf{ct}_{\ell,1},$$

where $\langle \tau^\ell \rangle_j$ denotes server j's Shamir share of τ^ℓ. This is one multi-scalar multiplication using only the server's own shares and the public ciphertexts. Each server sends a single $\mathbb{G}_1$ element σ_j to the combiner, so communication is $O(1)$ per server, independent of B.

The combiner reconstructs σ from any $t+1$ shares via Lagrange interpolation: $\sigma = \sum_{j \in V} L_j \cdot \sigma_j$. By the linearity of Shamir reconstruction, this gives the same σ as in the single-server setting. Everything else — α_ℓ, β_ℓ, and the unmasking — remains unchanged. The resulting protocol is a single-round threshold scheme. We present the full multi-server protocol in Fig. 3.

Step 4 — CCA security and Robustness. The construction described above is only CPA-secure. Moreover, in the above scheme, a malicious server can submit a malformed decryption share to prevent honest servers from decrypting the batch of ciphertexts. We address these issues using standard techniques, as follows.

- *CCA security.* We make the scheme CCA secure by requiring each encryption to include a (simulation-extractable) non-interactive zero-knowledge (NIZK) proof of knowledge of the encryption randomness.
- *Malicious servers.* To protect against malicious decryptors, we augment the decryption key with commitments to each server's shares: $\mathsf{pk}_j^i = [\![\langle \tau^i \rangle_j]\!]_2$ for each server j and power $i \in [B_{\mathsf{max}}]$. The combiner verifies σ_j via a single batched pairing check:

$$\sum_{\ell \in U} \left(\mathsf{ct}_{\ell,1} \circ \mathsf{pk}_j^\ell\right) \stackrel{?}{=} \sigma_j \circ [\![1]\!]_2.$$

  Servers that fail this check are excluded.

We defer the formal security analysis to §7.

4 Preliminaries and BTE Definition

Notations. We use λ 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 sampled uniformly at random from set S. We use bold letters to indicate vectors. For a vector $\mathbf{v}$ of length n, we use v_i to denote the i^{th} element where $i \in [n]$. By $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$, we mean the classes $\lambda^{O(1)}$ and $\frac{1}{\lambda^{\omega(1)}}$. We use PPT to denote probabilistic $\mathsf{poly}(\lambda)$-time Turing Machines.

4.1 Bilinear Groups

We follow the notation used in [17]. A bilinear group $\mathcal{BG}$ is a set of three groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of prime order p, with a (non-degenerate) bilinear map or pairing e. This map takes one element from $\mathbb{G}_1$ and one element from $\mathbb{G}_2$ and produces an element in $\mathbb{G}_T$. The groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are called the source groups, and $\mathbb{G}_T$ is the target group.

The groups $\mathbb{G}_1, \mathbb{G}_2$ have generators g_1, g_2, and we use the notation $[\![x]\!]_s$ to represent $x \cdot g_s$ in the group $\mathbb{G}_s$ for $s \in \{1, 2, T\}$, where $x \in \mathbb{Z}_p$ and the generator of $\mathbb{G}_T$ is $g_T = e(g_1, g_2)$. The group operation is additive, and therefore $[\![x]\!]_s + [\![y]\!]_s = [\![x + y]\!]_s$.

We represent the pairing operation $e([\![x]\!]_1, [\![y]\!]_2)$ as $[\![x]\!]_1 \circ [\![y]\!]_2 = [\![y]\!]_2 \circ [\![x]\!]_1 = [\![x \cdot y]\!]_T$. This notation composes naturally with linear operations. For vectors $\mathbf{u} \in \mathbb{G}_1^n, \mathbf{v} \in \mathbb{G}_2^n$, we have $\mathbf{u}^T \circ \mathbf{v} = [\![u_1 v_1 + \cdots + u_n v_n]\!]_T$.

4.2 Shamir Secret Sharing

We use a (t, N) Shamir secret sharing scheme over $\mathbb{Z}_p$. For any secret $x \in \mathbb{Z}_p$, the sharing procedure $\mathsf{Share}(x, N, t, \Omega)$ produces shares $(\langle x \rangle_1, \ldots, \langle x \rangle_N)$ such that any subset $T \subseteq [N]$ of size $|T| = t$ determines Lagrange coefficients $(\lambda_j)_{j \in T}$ satisfying $\sum_{j \in T} \lambda_j \langle x \rangle_j = x$. Here, $\Omega \in \mathbb{Z}_p^N$ is an evaluation domain of size N.

4.3 NIZKs

We will need a specific type of NIZK (non-interactive zero knowledge proof), namely, a *simulation-extractable* NIZK. Intuitively, this is a NIZK with the following properties. As usual, *zero-knowledge* means that there is a simulator that can simulate proofs of true statements without knowing a witness. *Simulation-extractable* means that there is an *online* extractor that can efficiently extract in an online fashion the witness from any valid proof of a statement, even when using the zero-knowledge simulator to generate proofs, so long as the statement/proof pair presented to the extractor is not identical to any statement/proof pair for a previously simulated proof. As a formal notion, we can restate the definition from [5].

Definition 4.1 (Non-Interactive Zero Knowledge Proofs)**.** Let $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{W}$ be a relation over sets $\mathcal{X}$ and $\mathcal{W}$. A NIZK scheme NIZK for $\mathcal{R}$ is a tuple of PPT algorithms $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ defined as follows:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$: given the security parameter λ, outputs a common reference string crs. The algorithms below implicitly take crs as input.
- $\mathsf{Prove}(x, w) \to \pi$: given a statement $x \in \mathcal{X}$ and a witness $w \in \mathcal{W}$, outputs a proof π.
- $\mathsf{Verify}(x, \pi) \to 0/1$: given a statement $x \in \mathcal{X}$ and a proof π, outputs 0 or 1.

Let $\mathcal{L}_\mathcal{R}$ denote the language defined by the relation $\mathcal{R}$. A NIZK scheme must satisfy the following properties.

Definition 4.2 (Completeness)**.** A NIZK scheme for $\mathcal{R}$ is complete if for all $(x, w) \in \mathcal{R}$, it holds that

$$\Pr[\mathsf{Verify}(x, \pi) = 1 \mid \pi \leftarrow \mathsf{Prove}(x, w)] \geq 1 - \mathsf{negl}(\lambda).$$

Definition 4.3 (Soundness)**.** A NIZK scheme for $\mathcal{R}$ is sound if for all $x \notin \mathcal{L}_\mathcal{R}$ and all PPT provers $\mathcal{P}^*$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\Pr[\mathsf{Verify}(x, \pi) = 1 \mid \pi \leftarrow \mathcal{P}^*(x)] \leq \mathsf{negl}(\lambda).$$

Definition 4.4 (Zero-Knowledge)**.** A NIZK scheme for $\mathcal{R}$ is zero-knowledge if there exists a PPT simulator Sim such that for all $(x, w) \in \mathcal{R}$,

$$(x, \mathsf{Sim}(x)) \approx_c (x, \mathsf{Prove}(x, w)).$$

Definition 4.5 (Simulation-Extractability)**.** A NIZK scheme for $\mathcal{R}$, $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$, is said to be simulation-extractable (or an SE-NIZK) if there exists a PPT extractor Ext and a PPT simulator $\mathsf{Sim} = (\mathsf{SimSetup}, \mathsf{SimProve})$ such that for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ for which the following holds:

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(x^*, \pi^*) = 1 \wedge & (\mathsf{crs}, \mathsf{td}_{\mathsf{sim}}, \mathsf{td}_{\mathsf{ext}}) \leftarrow \mathsf{SimSetup}(1^\lambda); \\ x^* \notin Q \wedge & Q \leftarrow \emptyset; \\ (x^*, w^*) \notin \mathcal{R} & (x^*, \pi^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{sim}}}(\mathsf{crs}); \\ & w^* \leftarrow \mathsf{Ext}(\mathsf{crs}, \mathsf{td}_{\mathsf{ext}}, x^*, \pi^*) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Here, $\mathcal{O}_{\mathsf{sim}}$ denotes the simulation oracle:

$$\mathcal{O}_{\mathsf{sim}}(x): \quad Q \leftarrow Q \cup \{x\}; \quad \text{return } \pi \leftarrow \mathsf{SimProve}(\mathsf{crs}, \mathsf{td}_{\mathsf{sim}}, x).$$

We will also require the extractor to be online (or straightline), meaning that the extractor does not rewind the adversary.

4.4 Hardness Assumptions

Assumption 4.1 (Decisional n-Power Diffie–Hellman(nDH))**.** *Let* GrpGen *be an asymmetric bilinear group generator. The* nDH *problem is hard for* GrpGen *if the following two distributions* $\mathcal{D}_0$ *and* $\mathcal{D}_1$ *are computationally indistinguishable:*

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

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

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

4.5 Batched Threshold Encryption: Definition

We formalize the syntax and security of *Batched Threshold Encryption* (BTE), following the definition in [1].

Definition 4.6 (Batched Threshold Encryption)**.** A batched threshold encryption scheme BTE consists of the following PPT algorithms and protocols:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$: The randomized setup algorithm takes as input the security parameter λ and outputs public parameters pp, which is an implicit input to all subsequent interfaces.
- $(\mathsf{ek}, \mathsf{dk}, \{\mathsf{sk}_j\}_{j\in[N]}) \leftarrow \mathsf{KeyGen}(1^{B_{\mathsf{max}}}, N, t)$: The randomized key-generation algorithm takes as input the total number of servers N and the corruption threshold t, along with the maximum batch size B_{max}, and outputs a public encryption key ek and a public decryption key dk to all, and the j-th secret key sk_j to server $j \in [N]$.
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{ek}, \mathsf{m})$: The randomized encryption algorithm takes as input an encryption key ek and a message m, and outputs a ciphertext ct.
- $\{\widetilde{\mathsf{m}_\ell}\}_{\ell\in[B]} \leftarrow \Pi\text{-}\mathsf{BDec}(\mathsf{dk}, \{\mathsf{ct}_\ell\}_{\ell\in[B]}, \langle\mathsf{sk}_j\rangle_{j\in[N]})$: This is a potentially interactive protocol among all $[N]$ servers for batch decryption, where the j^{th} server has a private input sk_j, and all servers have public decryption key dk and an *ordered* batch of ciphertexts $\{\mathsf{ct}_\ell\}_{\ell\in[B]}$ where $B \leq B_{\mathsf{max}}$. The protocol outputs the decrypted messages $\{\widetilde{\mathsf{m}_\ell}\}_{\ell\in[B]}$ in the same order, where each $\widetilde{\mathsf{m}_\ell}$ is either a valid message or $\perp$.

Game $\mathsf{Expt}^{\mathsf{BTE\text{-}CCA}}_{\mathcal{A},b}(N,t,\lambda,B_{\mathsf{max}})$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathcal{Q}_{\mathsf{BDec}} \leftarrow \emptyset$
$\mathcal{C} \leftarrow \mathcal{A}(\mathsf{pp})$
$(\mathsf{ek},\mathsf{dk},\{\mathsf{sk}_j\}_{j\in[N]}) \leftarrow \mathsf{KeyGen}(1^{B_{\mathsf{max}}},N,t)$
$(\mathsf{m}_0,\mathsf{m}_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{BDec}}}(\mathsf{ek},\mathsf{dk},\{\mathsf{sk}_j\}_{j\in\mathcal{C}})$
$\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{ek},\mathsf{m}_b)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{BDec}}}(\mathsf{ct}^*)$
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 $\mathcal{C}$.
$\{\mathsf{m}_i\}_{i\in[B]} \leftarrow \Pi\text{-}\mathsf{BDec}(\mathsf{dk},\{\mathsf{ct}_i\}_{i\in[B]},\langle\mathsf{sk}_j\rangle_{j\in[N]})$
$\mathcal{Q}_{\mathsf{BDec}} \leftarrow \mathcal{Q}_{\mathsf{BDec}} \cup \{\mathsf{ct}_i\}_{i\in[B]}$
return $\{\mathsf{m}_i\}_{i\in[B]}$

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

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathcal{C} \leftarrow \mathcal{A}(\mathsf{pp})$
if $|\mathcal{C}| > t$: **return** 0
$(\mathsf{ek},\mathsf{dk},\{\mathsf{sk}_j\}_{j\in[N]}) \leftarrow \mathsf{KeyGen}(1^{B_{\mathsf{max}}},N,t)$
$(B,S,\{\mathsf{m}_i\}_{i\in[B]\setminus S}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{BDec}}}(\mathsf{ek},\mathsf{dk},\{\mathsf{sk}_j\}_{j\in[N]})$
if $B > B_{\mathsf{max}} \vee S \not\subset [B]$: **return** 0
for all $i \in [B]\setminus S$:
    $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{ek},\mathsf{m}_i)$
$\{\mathsf{ct}_i\}_{i\in S} \leftarrow \mathcal{A}(\{\mathsf{ct}_i\}_{i\in[B]\setminus S})$
$\{\widetilde{\mathsf{m}}_i\}_{i\in[B]} \leftarrow \mathcal{O}_{\mathsf{BDec}}(\{\mathsf{ct}_i\}_{i\in[B]})$
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 $\mathsf{Expt}^{\mathsf{BTE\text{-}CCA}}_{\mathcal{A},b}$ and $\mathsf{Expt}^{\mathsf{BTE\text{-}rob}}_{\mathcal{A}}$, respectively, for a BTE scheme.

We require the BTE scheme to satisfy the *correctness*, IND-CCA *security*, and *robustness* properties. We formalize these properties in Theorems 4.7 to 4.9 and describe them next.

Correctness. The *correctness* property guarantees that correctly encrypted ciphertexts successfully decrypt to the original encrypted messages.

Definition 4.7 (Correctness of BTE)**.** A batched threshold encryption scheme BTE is correct if for all $N \in \mathbb{N}$, $t \le N$, $\lambda \in \mathbb{N}$, $B_{\mathsf{max}} \in \mathbb{N}$, $B \le B_{\mathsf{max}}$, and all messages $(\mathsf{m}_1,\mathsf{m}_2,\ldots,\mathsf{m}_B) \in \mathcal{M}^B$, the following holds with probability 1:

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

IND-CCA **Security.** The IND-CCA security for BTE guarantees that an adversary cannot distinguish between encryptions of two equal-length messages, even when it can invoke the batch decryption protocol (Π-BDec) on arbitrary batches of its choice not containing the challenge ciphertext. Security holds even when the adversary statically corrupts servers up to t.

Definition 4.8 (IND-CCA Security of BTE)**.** Let $\mathsf{Expt}^{\mathsf{BTE\text{-}CCA}}_{\mathcal{A},b}$ be the game as defined in Fig. 1. A batched threshold encryption scheme BTE is IND-CCA secure if for all $N \in \mathbb{N}, t < N, B_{\mathsf{max}} \in \mathbb{N}$, and all stateful PPT adversaries $\mathcal{A}$, we have

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

Robustness. The *robustness* property prevents an adversary $\mathcal{A}$ that can corrupt parties up to t from invoking the batch decryption protocol (Π-BDec) on malformed ciphertexts to disrupt decryption of correctly generated ciphertexts.

Definition 4.9 (Robustness of BTE)**.** Let $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{BTE\text{-}rob}}$ be the game as defined in Fig. 1. A batched threshold encryption scheme BTE is robust if for all $N \in \mathbb{N}, t < \lfloor N/2 \rfloor, B_{\mathsf{max}} \in \mathbb{N}$, we have:

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

Remark (Efficiency of BTE). We are interested in BTE schemes where the total communication complexity of Π-BDec, across all N servers, is $o(B)$, meaning that it scales sublinearly per server in the batch size B.

Definition 4.10 (Batched Encryption (BE))**.** The notion of Batched Threshold Encryption (Theorem 4.6) is also meaningful in the single server setting (special case where $N = 1, t = 0$). In this setting, we refer to it as a Batched Encryption (BE) scheme.

5 Construction of BTX

We present the BTX construction in two steps. We first describe the single-server setting (Fig. 2), which captures the core algebraic mechanism. We then extend it to the threshold setting (Fig. 3) by Shamir-sharing the secret key. In both figures, the parts needed for CCA security and robustness are highlighted in blue.

5.1 Single-Server Construction (BE)

We first describe BTX as a BE scheme (Theorem 4.10), where a single server holds the full secret key $\mathsf{sk} = \tau$.

Setup and key generation. The setup algorithm $\mathsf{Setup}(1^\lambda)$ generates a bilinear group $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, [\![1]\!]_1, [\![1]\!]_2, [\![1]\!]_T, p, \circ)$. The key generation algorithm $\mathsf{KeyGen}(1^{B_{\mathsf{max}}})$ samples a secret scalar $\tau \leftarrow\!\!\$\ \mathbb{Z}_p$ and produces three components:

- An *encryption key* $\mathsf{ek} = [\![\tau^{B_{\mathsf{max}}+1}]\!]_T$.
- A *decryption key* $\mathsf{dk} = \{h_i = [\![\tau^i]\!]_2\}_{i \in [2B_{\mathsf{max}}] \setminus \{B_{\mathsf{max}}+1\}}$, i.e., all powers of τ in $\mathbb{G}_2$ from 1 to $2B_{\mathsf{max}}$ except the middle power $B_{\mathsf{max}} + 1$.
- A *secret key* $\mathsf{sk} = \tau$.

The encryption key ek is a single $\mathbb{G}_T$ element. The decryption key dk consists of $2B_{\mathsf{max}}$ elements in $\mathbb{G}_2$ and is public. The omitted middle power $[\![\tau^{B_{\mathsf{max}}+1}]\!]_2$ is the key structural feature of the CRS. It prevents anyone from computing the encryption pad from the ciphertext alone, while the server can recover it using sk.

Encryption. To encrypt a message $m \in \mathbb{G}_T$, the client samples $r \leftarrow\!\!\$\ \mathbb{Z}_p$ and outputs

$$\mathsf{ct} := (\mathsf{ct}_1, \mathsf{ct}_2) := \big([\![r]\!]_1, \ \ m + r \cdot \mathsf{ek}\big).$$

This is an additive ElGamal-like ciphertext in $\mathbb{G}_T$: the encryption pad is $r \cdot \mathsf{ek} = [\![r \cdot \tau^{B_{\mathsf{max}}+1}]\!]_T$. For CCA security, the client additionally computes a NIZK proof π^{client} of knowledge of r such that $\mathsf{ct}_1 = r \cdot [\![1]\!]_1$, and appends it to the ciphertext (see Fig. 2). We emphasize that encryption *does not* require a batch index.

Batch decryption. Given an ordered batch $(\mathsf{ct}_1, \ldots, \mathsf{ct}_B)$ with $B \le B_{\mathsf{max}}$, the server decrypts them in two steps:

Building-blocks:

- N is the number of servers and t is the corruption threshold.
- λ denotes the security parameter, B_{max} is the maximum batch size, $B \le B_{\mathsf{max}}$ is the batch size.
- A NIZK system $\mathsf{DL} = (\mathsf{Prove}, \mathsf{Verify})$ to prove knowledge of discrete logarithm w.r.t. an arbitrary group.

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

1: $\mathcal{G} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, [\![1]\!]_1, [\![1]\!]_2, [\![1]\!]_T, p, \circ) \leftarrow \mathsf{GrpGen}(1^\lambda)$
2: **return** $\mathsf{pp} := \mathcal{G}$

$\underline{\mathsf{KeyGen}(1^{B_{\max}})}$:

1: $\tau \leftarrow\$ \mathbb{Z}_p$
2: $\mathsf{dk} := \{\underbrace{[\![\tau^i]\!]_2}_{h_i}\}_{i \in [2B_{\mathsf{max}}] \setminus \{B_{\mathsf{max}}+1\}}$
3: $\mathsf{ek} := [\![\tau^{B_{\mathsf{max}}+1}]\!]_T$
4: $\mathsf{sk} := \tau$
5: **return** $(\mathsf{ek}, \mathsf{dk}, \mathsf{sk})$

$\underline{\mathsf{Enc}(\mathsf{ek}, \mathsf{m} \in \mathbb{G}_T)}$:

1: $r \leftarrow\$ \mathbb{Z}_p$
// ElGamal encryption using randomness r
2: $\mathsf{ct}_1 := [\![r]\!]_1$
3: $\mathsf{ct}_2 := \mathsf{m} + \underbrace{r \cdot \mathsf{ek}}_{\mathsf{pad}}$
// Compute NIZK proof π^{client} for CCA security
4: $\chi^{\mathsf{client}} := (\underbrace{[\![1]\!]_1}_{\chi_1}, \underbrace{\mathsf{ct}_1}_{\chi_2}, \mathsf{ct}_2)$
5: $\mathsf{wit}^{\mathsf{client}} := r$
// Proving knowledge of discrete logarithm of χ_2 w.r.t. χ_1
6: $\pi^{\mathsf{client}} \leftarrow \mathsf{DL.Prove}(\chi^{\mathsf{client}}, \mathsf{wit}^{\mathsf{client}})$
7: **return** $\mathsf{ct} := (\mathsf{ct}_1, \mathsf{ct}_2, \pi^{\mathsf{client}})$

$\underline{\mathsf{BDec}_1(\mathsf{pp}, \{\mathsf{ct}_\ell\}_{\ell \in [B]}, \mathsf{sk})}$:

1: **for** all $\ell \in [B]$, parse ct_ℓ as $(\mathsf{ct}_{\ell,1}, \mathsf{ct}_{\ell,2}, \pi_\ell)$
// U contains the set of only valid ciphertexts
2: **for** $\ell \in [B]$, $\chi_\ell^{\mathsf{client}} := ([\![1]\!]_1, \mathsf{ct}_{\ell,1}, \mathsf{ct}_{\ell,2})$
3: $U := \{\ell \mid \ell \in [B] \wedge \mathsf{DL.Verify}(\chi_\ell^{\mathsf{client}}, \pi_\ell) = 1\}$
// Computing a succinct secret key σ for the batch of valid ciphertexts.
4: $\sigma := \Sigma_{\ell \in U} \mathsf{sk}^\ell \cdot \mathsf{ct}_{\ell,1}$
5: **return** σ

$\underline{\mathsf{BDec}_2(\sigma, \mathsf{dk})}$:

1: Parse dk as $(h_1, \ldots, h_{B_{\mathsf{max}}}, h_{B_{\mathsf{max}}+2}, \ldots, h_{2B_{\mathsf{max}}})$
2: **for** all $\ell \in U$, $\alpha_\ell := \sigma \circ h_{B_{\mathsf{max}}-\ell+1}$
// Compute all β_i in a single shot using FFT
3: **for** all $\ell \in U$, $\beta_\ell := \sum_{i \in U, i \neq \ell} \mathsf{ct}_{i,1} \circ h_{B_{\mathsf{max}}-\ell+i+1}$
// Decrypting all the valid ciphertexts
4: **for** all $\ell \in U$, $\widetilde{\mathsf{m}_\ell} := \mathsf{ct}_{\ell,2} - \underbrace{(\alpha_\ell - \beta_\ell)}_{\mathsf{pad}}$
// Default output for all the invalid ciphertexts
5: **for** all $\ell \in [B] \setminus U$, $\widetilde{\mathsf{m}_\ell} := \perp$
6: **return** $\{\widetilde{\mathsf{m}_\ell}\}_{\ell \in [B]}$

Figure 2: Construction of Batched Encryption scheme (**single server** setting) supporting a maximum batch size B_{max}. We highlight parts needed for robustness and CCA security (against any number of malicious clients) in blue.

Step 1. In the first step (BDec_1), the server verifies the client NIZK proofs and filters out invalid ciphertexts, retaining the set U of valid indices. It then computes the *batch secret key* σ as:

$$\sigma = \sum_{\ell \in U} \tau^\ell \cdot \mathsf{ct}_{\ell,1} = \sum_{\ell \in U} [\![r_\ell \cdot \tau^\ell]\!]_1.$$

This is a single multi-scalar multiplication of size $|U|$.

Step 2. In the second step (BDec_2), the server computes the decryption materials $(\alpha_\ell, \beta_\ell)$ for each slot $\ell \in U$, as follows:

$$\alpha_\ell = \sigma \circ h_{B_{\mathsf{max}}-\ell+1}, \qquad \beta_\ell = \sum_{\substack{i \in U \\ i \neq \ell}} \mathsf{ct}_{i,1} \circ h_{B_{\mathsf{max}}-\ell+i+1},$$

and recovers $m_\ell = \mathsf{ct}_{\ell,2} - (\alpha_\ell - \beta_\ell)$. For all $\ell \in [B] \setminus U$ (invalid ciphertexts), the output is $\perp$.

We prove in §7 that $\alpha_\ell - \beta_\ell = [\![r_\ell \cdot \tau^{B_{\mathsf{max}}+1}]\!]_T = r_\ell \cdot \mathsf{ek}$, which is exactly the encryption pad for message m_ℓ. Intuitively, the exponent $B_{\mathsf{max}} - \ell + i + 1$ above equals $B_{\mathsf{max}} + 1$ only when $i = \ell$, which the sum excludes.

Cost. Computing all α_ℓ terms requires $|U|$ pairings. Each β_ℓ term requires $|U| - 1$ pairings, for $O(B^2)$ pairings total. In §6, we reduce the β computation to $O(B \log B)$ group exponentiations and $O(B)$ pairings.

5.2 Threshold Construction (BTE)

We now extend the single-server scheme to the threshold setting, where N servers collectively hold the secret key and any $t + 1$ of them can decrypt.

Key generation. The key generation algorithm $\mathsf{KeyGen}(1^{B_{\mathsf{max}}}, N, t)$ runs exactly as in the single-server case to produce ek and dk, but instead of outputting $\mathsf{sk} = \tau$, it Shamir-shares each power τ^i for $i \in [B_{\mathsf{max}}]$ among the N servers:

$$\{\langle \tau^i \rangle_j\}_{j \in [N]} \leftarrow \mathsf{Share}(\tau^i, N, t, \Omega) \qquad \text{for } i \in [B_{\mathsf{max}}],$$

where $\Omega = \{\Omega_1, \ldots, \Omega_N\} \subseteq \mathbb{Z}_p$ is the evaluation domain. Server j receives $\mathsf{sk}_j = (\langle \tau \rangle_j, \langle \tau^2 \rangle_j, \ldots, \langle \tau^{B_{\mathsf{max}}} \rangle_j)$.

For CCA security against malicious servers, the decryption key dk is augmented with commitments to each server's shares: $\mathsf{pk}_j^i = [\![\langle \tau^i \rangle_j]\!]_2$ for $i \in [B_{\mathsf{max}}]$ and $j \in [N]$. These enable any decryptors to verify the correctness of each server's contribution (see below).

Encryption. The encryption algorithm is identical to the single-server case.

Batch decryption. As in the single server protocol, the batch decryption protocol Π-BDec proceeds in two steps.

Step 1. In the first step (Π-BDec_1), each server j independently verifies the client NIZK proofs to determine the valid set U, then computes its share of the batch secret key:

$$\sigma_j = \sum_{\ell \in U} \langle \tau^\ell \rangle_j \cdot \mathsf{ct}_{\ell,1}.$$

Each server sends a single $\mathbb{G}_1$ element σ_j to the combiner. The communication per server is $O(1)$, independent of B.

Step 2. In the second step (Π-BDec_2), a decryptor (could be the servers) first verifies each server's contribution using a multi-pairing check:

$$\sum_{\ell \in U} (\mathsf{ct}_{\ell,1} \circ \mathsf{pk}_j^\ell) \stackrel{?}{=} \sigma_j \circ [\![1]\!]_2.$$

Let S be the subset of validators who sent valid decryption shares, and let $V \subseteq S$ with $|V| > t$. We then combine decryption shares from the servers in V using Lagrange interpolation in the exponent to compute the batch secret key σ as:

$$\sigma := \sum_{j \in V} L_j \cdot \sigma_j.$$

As the final step of the decryption process, the decryptor computes α_ℓ, β_ℓ as in the BE scheme, and recovers each message exactly as in the single-server case.

Building-blocks:

- N is the number of servers and t is the corruption threshold.
- λ denotes the security parameter, B_{max} is the maximum batch size, $B \leq B_{\mathsf{max}}$ is the batch size.
- A NIZK system $\mathsf{DL} = (\mathsf{Prove}, \mathsf{Verify})$ to prove knowledge of discrete logarithm w.r.t. an arbitrary group.
- Shamir secret sharing scheme having algorithms Share and $\mathsf{Lagrange}$

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

1: $\mathcal{G} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, [\![1]\!]_1, [\![1]\!]_2, [\![1]\!]_T, p, \circ) \leftarrow \mathsf{GrpGen}(1^\lambda)$
2: **return** $\mathsf{pp} := \mathcal{G}$

$\underline{\mathsf{KeyGen}(1^{B_{\max}}, N, t)}$:

1: $\tau \leftarrow\$ \mathbb{Z}_p$
// Secret sharing evaluation domain Ω
2: $\Omega = \{\Omega_1, \ldots, \Omega_N\} \subseteq \mathbb{Z}_p$ s.t. $|\Omega| = N$
3: $\forall i \in [B_{\mathsf{max}}]$, $\{\tau_j^i\}_{j\in[N]} \leftarrow \mathsf{Share}(\tau^i, N, t, \Omega)$
4: **for** $j \in [N]$, $\mathsf{sk}_j := \{\tau_j^i\}_{i\in[B_{\mathsf{max}}]}$
5: $\mathsf{dk} := \begin{cases} \{\underbrace{[\![\tau^i]\!]_2}_{h_i}\}_{i\in[2B_{\mathsf{max}}]\setminus\{B_{\mathsf{max}}+1\}} \\ \{\underbrace{[\![\tau_j^i]\!]_2}_{\mathsf{pk}_j^i}\}_{i\in[B_{\mathsf{max}}],j\in[N]} \end{cases}$
6: $\mathsf{ek} := [\![\tau^{B_{\mathsf{max}}+1}]\!]_T$
7: **return** $(\mathsf{ek}, \mathsf{dk}, \{\mathsf{sk}_j\}_{j\in[N]})$

$\underline{\mathsf{Enc}(\mathsf{ek}, \mathsf{m} \in \mathbb{G}_T)}$:

1: $r \leftarrow\$ \mathbb{Z}_p$
// ElGamal encryption using randomness r
2: $\mathsf{ct}_1 := [\![r]\!]_1$
3: $\mathsf{ct}_2 := \mathsf{m} + \underbrace{r \cdot \mathsf{ek}}_{\mathsf{pad}}$
// Compute NIZK proof π^{client} for CCA security
4: $\chi^{\mathsf{client}} := \underbrace{[\![1]\!]_1}_{\chi_1}, \underbrace{\mathsf{ct}_1}_{\chi_2}, \mathsf{ct}_2$
5: $\mathsf{wit}^{\mathsf{client}} := r$
// Proving knowledge of discrete logarithm of χ_2 w.r.t. χ_1
6: $\pi^{\mathsf{client}} \leftarrow \mathsf{DL.Prove}(\chi^{\mathsf{client}}, \mathsf{wit}^{\mathsf{client}})$
7: **return** $\mathsf{ct} := (\mathsf{ct}_1, \mathsf{ct}_2, \pi^{\mathsf{client}})$

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

1: **for** all $\ell \in [B]$, parse ct_ℓ as $(\mathsf{ct}_{\ell,1}, \mathsf{ct}_{\ell,2}, \pi_\ell)$
// U contains the set of only valid ciphertexts
2: **for** $\ell \in [B]$, $\chi_\ell^{\mathsf{client}} := ([\![1]\!]_1, \mathsf{ct}_{\ell,1}, \mathsf{ct}_{\ell,2})$
3: $U := \{\ell \mid \ell \in [B] \wedge \mathsf{DL.Verify}(\chi_\ell^{\mathsf{client}}, \pi_\ell) = 1\}$
// Succinct secret key share σ_j for the batch of valid ciphertexts.
4: Parse sk_j as $(\mathsf{sk}_j^1, \ldots, \mathsf{sk}_j^{B_{\mathsf{max}}})$
5: $\sigma_j := \Sigma_{\ell\in U} \mathsf{sk}_j^\ell \cdot \mathsf{ct}_{\ell,1}$
6: **return** σ_j

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

1: Parse dk as $\begin{cases} \{h_i\}_{i\in[2B_{\mathsf{max}}]\setminus\{B_{\mathsf{max}}+1\}} \\ \{\mathsf{pk}_j^i\}_{i\in[B_{\mathsf{max}}],j\in[N]} \end{cases}$
// S includes the indices of only those servers who send a valid message σ_j
2: $S := \{j | j \in [N] \wedge \sum_{\ell\in U}(\mathsf{ct}_{\ell,1} \circ \mathsf{pk}_j^\ell) = \sigma_j \circ [\![1]\!]_2\}$
3: Select $V \subseteq S$ s.t. $|V| > t$
// Defining the share interpolation domain Ω' and lagrange coefficients L_ω
4: $\Omega' := \{\Omega_j\}_{j\in V}$ and $\{L_j\}_{j\in V} := \mathsf{Lagrange}(\Omega', t)$
// Reconstruct the batch secret key σ
5: $\sigma := \sum_{j\in V} L_j \cdot \sigma_j$
6: **for** all $\ell \in U$, $\alpha_\ell := \sigma \circ h_{B_{\mathsf{max}}-\ell+1}$
// Compute all β_i in a single shot using FFT
7: **for** all $\ell \in U$, $\beta_\ell := \sum_{i\in U, i\neq\ell} \mathsf{ct}_{i,1} \circ h_{B_{\mathsf{max}}-\ell+i+1}$
// Decrypting all the valid ciphertexts
8: **for** all $\ell \in U$, $\widetilde{\mathsf{m}_\ell} := \mathsf{ct}_{\ell,2} - \underbrace{(\alpha_\ell - \beta_\ell)}_{\mathsf{pad}}$
// 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 3: Construction of Batched Threshold Encryption scheme for N servers supporting a maximum batch size B_{max}. We highlight parts needed for robustness and CCA security (against any number of malicious clients and upto t malicious servers running $\Pi\text{-}\mathsf{BDec}$) in blue.

Remark. We note that, in the optimistic case, i.e., when all servers are honest, we can optimize the computation cost to verify the validity of the decryption by first interpolating $\{\sigma_j\}_{j\in S}$ to compute

σ and then verify it at once by checking:

$$\sum_{\ell \in U} (\mathsf{ct}_{\ell,1} \circ [\![\tau^{\ell}]\!]_2) = \sigma \circ [\![1]\!]_2\}$$

When all servers are honest, the verification above would require a multi-pairing of size B.

6 Computational Optimizations

The main computational bottleneck in batch decryption is the cross-term vector $(\beta_1, \ldots, \beta_B)$. Recall from §5 that for each slot $\ell \in [B]$,

$$\beta_\ell := \sum_{\substack{i=1 \\ i \neq \ell}}^{B} \mathsf{ct}_{i,1} \circ h_{B_{\mathsf{max}}-\ell+i+1}.$$

A naive implementation computes each β_ℓ using $B-1$ pairings, for $B(B-1)$ pairings in total. In contrast, the α_ℓ terms require only B pairings (one per slot) and depend on the threshold output σ. In this section, we show how to reduce the cross-term computation from quadratic to quasi-linear.

6.1 Middle-Product View

We observe that the β_ℓ values form a contiguous coefficient window of a bilinear polynomial product. This is exactly the *middle-product* pattern studied by Hanrot et al. [20]. Define the $\mathbb{G}_1$-valued polynomial

$$A(X) := \sum_{j=0}^{B-1} A_j X^j, \qquad A_j := \mathsf{ct}_{j+1,1} \in \mathbb{G}_1,$$

and define the fixed $\mathbb{G}_2$-valued polynomial $Q(X)$ from the CRS powers in reverse order:

$$Q(X) := \sum_{u=0}^{2B_{\mathsf{max}}-1} q_u X^u,$$

where

$$q_u := \begin{cases} [\![\tau^{2B_{\mathsf{max}}-u}]\!]_2 & \text{if } 2B_{\mathsf{max}} - u \neq B_{\mathsf{max}} + 1, \\ 0 & \text{if } 2B_{\mathsf{max}} - u = B_{\mathsf{max}} + 1. \end{cases}$$

The zero coefficient at position $u = B_{\mathsf{max}} - 1$ corresponds to the omitted power $[\![\tau^{B_{\mathsf{max}}+1}]\!]_2$.

Let $C(X) := A(X) \circ Q(X)$, where coefficient multiplication is given by the bilinear map:

$$\Big(\sum_j a_j X^j\Big) \circ \Big(\sum_u b_u X^u\Big) := \sum_m \Big(\sum_{j+u=m} a_j \circ b_u\Big) X^m.$$

Claim 1. *For each $\ell \in [B]$, the coefficient of $X^{B_{\mathsf{max}}+\ell-2}$ in $C(X)$ equals β_ℓ.*

Proof. The coefficient of $X^{B_{\mathsf{max}}+\ell-2}$ is $\sum_{j=0}^{B-1} A_j \circ q_{B_{\mathsf{max}}+\ell-2-j}$. Substituting $A_j = \mathsf{ct}_{j+1,1}$ and using the definition of q_u, we get $q_{B_{\mathsf{max}}+\ell-2-j} = [\![\tau^{B_{\mathsf{max}}-\ell+j+2}]\!]_2 = h_{B_{\mathsf{max}}-\ell+(j+1)+1}$, except when $j+1 = \ell$, in which case the exponent is $B_{\mathsf{max}}+1$ and $q_u = 0$. Setting $i = j+1$, the sum becomes $\sum_{i \neq \ell} \mathsf{ct}_{i,1} \circ h_{B_{\mathsf{max}}-\ell+i+1} = \beta_\ell$. □

The vector $(\beta_1, \ldots, \beta_B)$ is therefore a contiguous block of B coefficients of $C(X)$, starting at degree $B_{\mathsf{max}} - 1$. The problem of evaluating all β_ℓ reduces to a middle-product computation, where the second input $Q(X)$ is fixed across batches.

6.2 FFT-Based Middle Product

The standard way to compute a middle product is to embed the relevant coefficient window into a cyclic convolution of appropriate length [20]. Since $Q(X)$ is fixed, its FFT representation can be precomputed once and reused across batches. Let

$$m := 2^{\lceil \log_2(2B) \rceil}.$$

The $\mathsf{precompute}(B)$ step then performs:

1. One forward $\mathbb{G}_1$ FFT of size m to transform the ciphertext polynomial $A(X)$.
2. m pointwise pairings in the evaluation domain.
3. One truncated inverse $\mathbb{G}_T$ FFT of size m to recover the product coefficients.
4. Read off β_ℓ from the relevant coefficient window.

For power-of-two batches, this amounts to exactly $2B$ pairings and two FFTs of size $2B$. The FFTs contribute $O(B \log B)$ group exponentiations, so the total cost is $O(B \log B)$ exponentiations and $O(B)$ pairings.

Note that the inverse transform operates in the target group $\mathbb{G}_T$, where group operations are approximately 3–5× more expensive than in the source groups. This affects the concrete constants but not the asymptotic improvement.

6.3 Karatsuba-Style Middle Product

An alternative is to compute the middle product recursively using a Karatsuba-style decomposition [20]. The coefficient window is split into overlapping subproblems, and the Karatsuba reuse pattern reduces the number of bilinear multiplications below the naive quadratic count.

Since $Q(X)$ is fixed, all required combinations of its coefficients can be precomputed once. The recursive recombination uses only $\mathbb{G}_T$ additions and inversions, while the pairings occur only at the base level. In particular, the final exponentiation step of the pairing can be deferred through the recursion, yielding better constants for moderate batch sizes.

This gives a subquadratic algorithm with $O(B^{1.585})$ operations, without requiring a target-group FFT.

6.4 Centered Decryption Key

The middle-product structure becomes particularly clean in a centered coordinate system. For each nonzero offset $d \in \{-B_{\mathsf{max}}, \ldots, -1, 1, \ldots, B_{\mathsf{max}}\}$, define

$$h_d := [\![\tau^{B_{\mathsf{max}}+1+d}]\!]_2.$$

In this notation, the decryption equations become

$$\alpha_\ell = \sigma \circ h_{-\ell}, \qquad \beta_\ell = \sum_{\substack{i=1 \\ i \neq \ell}}^{B} \mathsf{ct}_{i,1} \circ h_{i-\ell}.$$

The cross-term depends only on the relative offset $i - \ell$, and the effective decryption key window for a batch of size B is

$$h_{-B}, h_{-(B-1)}, \ldots, h_{-1}, \quad 0, \quad h_1, \ldots, h_{B-1}.$$

This centered kernel of size $O(B)$ is the natural input to both FFT-based and Karatsuba-based middle-product algorithms.

6.5 Phase Decomposition

The batch decryption path separates naturally into four phases: $\mathsf{precompute}(B)$, $\mathsf{partialDecrypt}(B)$, $\mathsf{combine}(n)$, and $\mathsf{open}(B)$, where B is the batch size and $n := |V|$ is the number of valid threshold shares actually used in reconstruction. When robustness is enabled, two additional subroutines are layered on top of this path: $\mathsf{ctxtCheck}(B)$ inside $\mathsf{precompute}(B)$ and $\mathsf{serverCheck}(B, n)$ inside checked combine.

$\mathsf{precompute}(B)$ — **offline batch work.** The combiner computes all β_ℓ values using the FFT or Karatsuba method. Since this depends only on the ciphertext batch and the public CRS, it can begin as soon as the batch is fixed.

$\mathsf{ctxtCheck}(B)$ — **client proof filtering.** If robustness against malformed ciphertexts is enabled, the combiner also verifies the client proofs attached to the batch ciphertexts and filters to the valid subset U. This subroutine is linear in B and sits naturally inside $\mathsf{precompute}(B)$, since it depends only on the ciphertext batch and public verification material.

$\mathsf{partialDecrypt}(B)$ — **per-server threshold work.** Each server computes its aggregate share σ_j using one $\mathbb{G}_1$ MSM of size B.

$\mathsf{combine}(n)$ — **threshold reconstruction.** The combiner reconstructs σ from n shares using one $\mathbb{G}_1$ MSM of size n.

$\mathsf{serverCheck}(B, n)$ — **optimistic server-share check.** If robustness against malformed server shares is enabled, the combiner additionally verifies the reconstructed aggregate share against the public powers $\{[\![\tau^\ell]\!]_2\}_{\ell \in [B]}$. In the optimistic path, this is one aggregate pairing check over the batch and therefore linear in B and essentially independent of n.

$\mathsf{open}(B)$ — **final opening.** The combiner computes α_ℓ for all $\ell \in [B]$ (B pairings), subtracts the precomputed β_ℓ values, and unmasks the messages.

The $\mathsf{precompute}$ step runs concurrently with the threshold path ($\mathsf{partialDecrypt}$ and $\mathsf{combine}$). When enabled, $\mathsf{ctxtCheck}$ is absorbed into the same offline batch-processing window, while $\mathsf{serverCheck}$ is absorbed into checked combine. The final open step requires only B pairings plus linear-time subtraction and unmasking. In encrypted-mempool deployments, where the threshold path is often dominated by network delay, this decomposition is important: the batch-dependent FFT work overlaps with the threshold protocol, and the final opening step contributes as little additional latency as possible.

7 Analysis

7.1 Correctness

Theorem 7.1 (Correctness)**.** *For every ordered batch $(\mathsf{ct}_1, \ldots, \mathsf{ct}_B)$ of honestly generated ciphertexts with $B \le B_{\mathsf{max}}$, the decryption algorithm recovers all messages correctly.*

Proof. Let each $\mathsf{ct}_i = ([\![r_i]\!]_1, \mathsf{ct}_{i,2})$. Fix an index $i \in [B]$. By definition of σ,

$$\sigma = \sum_{k=1}^{B} [\![r_k \tau^k]\!]_1.$$

Hence

$$\alpha_i = \sigma \circ [\![\tau^{B_{\mathsf{max}}+1-i}]\!]_2 = \sum_{k=1}^{B} [\![r_k \tau^{k+B_{\mathsf{max}}+1-i}]\!]_T.$$

On the other hand,

$$\beta_i = \sum_{\substack{k=1 \\ k\neq i}}^{B} [\![r_k]\!]_1 \circ [\![\tau^{B_{\mathsf{max}}+1-i+k}]\!]_2 = \sum_{\substack{k=1 \\ k\neq i}}^{B} [\![r_k \tau^{k+B_{\mathsf{max}}+1-i}]\!]_T.$$

Subtracting gives

$$\gamma_i = \alpha_i - \beta_i = [\![r_i \tau^{B_{\mathsf{max}}+1}]\!]_T.$$

which is exactly the pad used in the second component of ct_i. Thus

$$\mathsf{ct}_{i,2} - \gamma_i = m_i.$$

Since this holds for all $i \in [B]$, all messages are recovered. □

7.2 CCA security (single server)

Theorem 7.2 (CCA security (single server))**.** *The single-server scheme BE in Fig. 2 is CCA secure under the* nDH *assumption (see Assumption 4.1) and the existence of a simulation-extractable NIZK (Definition 4.1).*

Proof. The proof is quite straightforward. It is easiest to present the proof using the "bit guessing" version of the CCA attack game (see, for example, Section 2.2.5 of [8]). Here, the challenger chooses $b \in \{0,1\}$ at random and the adversary's task is to guess b with probability significantly better than $1/2$. More precisely, the adversary outputs $b' \in \{0,1\}$ and the adversary's advantage is defined to be $|\Pr[b' = b] - 1/2|$. This bit-guessing advantage is $1/2$ the advantage as defined earlier in Definition 4.8, and our goal is to show that this is negligible.

We do this via a sequence of games.

Game 0. This is the bit-guessing version of the original attack game. Let X_0 be the event that $b' = b$ in this game, so that $\Pr[X_0] - 1/2$ is the adversary's bit-guessing advantage.

Game 1. This is the same as Game 0, except that we replace the NIZK used to create the challenge ciphertext ct^* by a simulated proof.

Let X_1 be the event that $b' = b$ in this game. By the zero-knowledge property of the NIZK, $|\Pr[X_0] - \Pr[X_1]|$ is negligible.

Game 2. This is the same as Game 1, except we use the simulation-extractability property of the NIZK to extract a witness from ciphertexts. More precisely, consider one invocation of the decryption oracle $\mathsf{BDec}_1(\mathsf{pp}, \{\mathsf{ct}_\ell\}_{\ell\in[B]}, \mathsf{sk})$. From each valid ciphertext $(\mathsf{ct}_{\ell,1}, \mathsf{ct}_{\ell,2}, \pi_\ell)$, we extract the witness r_ℓ, which satisfies $[\![r_\ell]\!]_1 = \mathsf{ct}_{\ell,1}$. If any of these extractions fail, we abort. Using all of these witnesses, and only public data, we can compute $\sigma = \sum_{\ell\in U} \mathsf{sk}^\ell \cdot \mathsf{ct}_{\ell,1}$, where U is the set of indices with valid NIZK proofs (as in Fig. 2), as follows:

$$\sigma \leftarrow \sum_{\ell\in U} r_\ell [\![\tau^\ell]\!]_1.$$

Let X_2 be the event that $b' = b$ in this game. By the simulation-extractability property of the NIZK, $|\Pr[X_1] - \Pr[X_2]|$ is negligible.

Game 3. Consider again the logic for generating the challenge ciphertext $\mathsf{ct}^* = (\mathsf{ct}_1^*, \mathsf{ct}_2^*, \pi^*)$, where $\mathsf{ct}_1^* = [\![r^*]\!]_1$ and $\mathsf{ct}_2^* = \mathsf{m}_b + r^* \cdot \mathsf{ek}$. In Game 1, we replaced π^* by a simulated proof, so r^* is not used in the generation of π^*. Moreover, after the modification made in Game 2, the secret key is not used anywhere. This justifies the following modification: compute $\mathsf{ct}_2^* \leftarrow \mathsf{m}_b + \rho$, where ρ is a random element of $\mathbb{G}_T$.

Let X_3 be the event that $b' = b$ in this game. A straightforward reduction to nDH shows that $|\Pr[X_2] - \Pr[X_3]|$ is negligible. It is also evident that $\Pr[X_3] = 1/2$. That completes the proof of the theorem. □

7.3 Implementing the NIZK

One can implement the NIZK by just applying the Fiat-Shamir transform to the Schnorr ID protocol. However, as noted by [25], this does not yield online extractability in the ROM. Prior works [10, 1, 7] get around this by appealing to the AGM (algebraic group model). We suggest another approach that allows us to prove that Fiat-Shamir applied to Schnorr gives the result we need in GGM+ROM.

The idea is the following. We extend our nDH assumption by making it interactive. Specifically, we consider the following game between a challenger and an adversary. In the first step, the challenger chooses x, y, z at random as in Assumption 4.1, and sends the public data (either $\mathcal{D}_0$ or $\mathcal{D}_1$) to the adversary. Next, the adversary interacts with the challenger by effectively opening many (possibly concurrent) Schnorr ID sessions, where the adversary plays the role of prover and the challenger plays the role of verifier. In each such session, the adversary first sends a pair of group elements $([\![r]\!]_1, [\![s]\!]_1)$ to the challenger, who responds with a random Schnorr challenge $c \in \mathbb{Z}_p$. At some later time, the adversary may resume the session by sending a response $w \in \mathbb{Z}_p$ to the challenger, who checks if $[\![w]\!]_1 = [\![s]\!]_1 + c[\![r]\!]_1$. If this check passes, the challenger responds with $\{[\![rx^i]\!]_1\}_{i\in[n]}$ (and otherwise does not respond at all).

The goal of the adversary is still the same: to distinguish between $\mathcal{D}_0$ and $\mathcal{D}_1$. The *extended* nDH assumption is that this remains hard even given access to these Schnorr ID sessions.

We claim two things, which are fairly straightforward to prove.

1. In the ROM (modeling the hash function used for Fiat-Shamir), the CCA security of the BE scheme reduces to the extended nDH assumption.

2. The extended nDH assumption holds in the GGM.

As known in the literature, it is also possible to get straightline simulation-extractability NIZK only in ROM (without GGM or AGM) by using the Fischlin transformation [16] at the cost of a much larger proof size.

7.4 CCA security (multi server)

Theorem 7.3 (CCA security (multi server))**.** *The multi-server scheme BTE in Fig. 3 is CCA secure if the single-server scheme BE is CCA secure.*

Proof. This is a standard reduction and we omit the proof. □

Note that we are crucially assuming *static* corruptions in the multi-server setting. This is what makes the reduction straightforward.

Phase	BTX	PFE [7]	BEAT++ [1]
precompute(B)	$2B$ pairings; 2 FFTs of size $2B$	B pairings; 4 FFTs of size $2B$	$2B$ pairings; 2 FFTs of size $2B$
partialDecrypt(B)	1 $\mathbb{G}_1$ MSM of size B	1 $\mathbb{G}_1$ MSM of size B	1 $\mathbb{G}_1$ MSM of size B
combine(n)	1 $\mathbb{G}_1$ MSM of size n	1 $\mathbb{G}_1$ MSM of size n	1 $\mathbb{G}_1$ MSM of size n
open(B)	B pairings; B $\mathbb{G}_T$ subtractions; B KDF/unmask operations	$4B$ pairings; linear $\mathbb{G}_T$ work; B KDF operations	B pairings; linear $\mathbb{G}_T$ work; B KDF operations

Table 2: Phase-by-phase cost comparison assuming B is a power of two. Here B is the batch size and n is the number of threshold shares combined.

	combine(n) (ms/share)			serverCheck(B, n) (ms/item)			
n	PFE	BTX	BEAT++	B	$n = 8$	$n = 62$	$n = 512$
2	0.042	0.042	0.042	32	0.179	0.179	0.179
4	0.043	0.043	0.042	64	0.167	0.167	0.165
8	0.039	0.039	0.043	128	0.161	0.161	0.160
16	0.031	0.031	0.044	256	0.158	0.158	0.157
				512	0.156	0.156	0.156

Table 3: Threshold/combine measurements over 3 repetitions. Left: mean running time of combine(n) in ms/share. Right: mean running time of serverCheck(B, n), reported as the per-item difference (ms/item) between checked combine and raw combine.

7.5 Complexity

Assume throughout this subsection that B is a power of two, so the transform size used in the FFT implementation is $m = 2B$. We write the batch opening path as the four phases precompute(B), partialDecrypt(B), combine(n), and open(B), where B is the batch size and $n := |V|$ is the number of valid threshold shares actually used in reconstruction.

When robustness is enabled, two additional subroutines are layered on top of this core decomposition for BTX. First, ctxtCheck(B) verifies client proofs inside precompute(B) and therefore costs linear work in B. Second, serverCheck(B, n) verifies the reconstructed aggregate share during checked combine. In the optimistic path used in our implementation, this adds one aggregate pairing check over the batch, i.e., $B + 1$ pairing terms in one multipairing, so the cost is $O(B)$ and essentially independent of n. This replaces the direct per-share server-check path in Fig. 3, which cost $\Theta(Bn)$.

The table makes the comparison transparent for the core four phases. BTX and BEAT++ match on all four phases at the level of operation counts. The main difference relative to PFE is the split between the batch-dependent and final opening work: PFE saves pairings in precompute(B) at the cost of extra FFTs, but its open(B) phase requires $4B$ pairings rather than B.

8 Implementation and Evaluation

We implement BTX over BLS12-381 using the `blst` library, compiled with Clang 21.1.8 in release mode with AVX-512, ADX, and native vectorization enabled. All experiments run on an AWS instance with an Intel Xeon Platinum 8488C processor (48 cores, 96 threads, 3.8 GHz max frequency) and approximately 105 MiB shared L3 cache. Our implementation uses AVX-512 vectorization,

	partialDecrypt(B) (ms/item)			precompute(B) (ms/item)			ctxtCheck(B) (ms/item)	open(B) (ms/item)		
B	PFE	BTX	BEAT++	PFE	BTX	BEAT++	BTX	PFE	BTX	BEAT++
32	0.035	0.035	0.039	0.963	0.644	0.642	0.097	0.721	0.171	0.171
64	0.029	0.029	0.039	1.120	0.722	0.722	0.097	0.721	0.171	0.172
128	0.025	0.025	0.039	1.278	0.801	0.800	0.098	0.721	0.171	0.171
256	0.022	0.022	0.039	1.436	0.880	0.880	0.098	0.722	0.171	0.171
512	0.019	0.019	0.039	1.596	0.959	0.960	0.100	0.723	0.171	0.171

Table 4: Mean running times of the batch-size-dependent phases, together with the isolated ciphertext-proof verification cost ctxtCheck(B).

in line with the broader recent push toward vectorized number-theoretic cryptography [22]. We compare BTX against two baselines: PFE [7] and BEAT++ [1]. To avoid sandbagging the baselines, we reimplement all three systems in the same aggressively optimized codebase rather than benchmarking against previously reported prototype numbers. This shared implementation stack includes AVX-512 field/group arithmetic, optimized MSM and multipairing routines, FFT-based backends wherever the algebra permits them, and a number of additional micro-optimizations across the pipeline. In particular, the absolute runtimes we obtain for the baselines are materially better than earlier reported implementations, especially for BEAT++. For fairness, we also implement an FFT-optimized Toeplitz multiplication backend for the PFE baseline.

We report the average time organized around the four-phase decomposition from §6: precompute(B), partialDecrypt(B), combine(n), and open(B), where B is the batch size. Here $n := |V|$ denotes the number of valid server shares actually used in reconstruction, so $t + 1 \le n \le N$, where N is the total number of servers. To make the robustness overhead equally explicit, we also write ctxtCheck(B) for the isolated client ciphertext-proof verification subroutine inside precompute(B), and serverCheck(B, n) for the optimistic server-share correctness check inside checked combine. The batch-size-dependent phases are grouped in Table 4, while the threshold/combine measurements are grouped in Table 3.

Robustness overhead. We also isolate the two robustness subroutines. The column labeled ctxtCheck(B) in Table 4 reports the per-item cost of client ciphertext-proof verification inside precompute(B). The right half of Table 3 reports serverCheck(B, n), namely the added per-item cost of the optimistic server-side correctness check obtained by taking the difference between the raw combine routine and the checked combine routine that first reconstructs σ and then verifies it against the public powers $\{[\![\tau^\ell]\!]_2\}_{\ell\in[B]}$. In that table, n again denotes the number of server messages passed to checked combine. The main takeaway is that serverCheck(B, n) is essentially independent of n and scales linearly with B, as expected from a single aggregate pairing check over the batch. This replaces the older per-share server-check path, whose cost scaled as $\Theta(Bn)$.

Takeaway. BTX and BEAT++ remain close on the batch-dependent phases precompute(B) and open(B), where both share the same internal algebraic structure and differ mainly in the surrounding protocol structure (collision-free indexing vs. user-chosen indices). On the MSM-only threshold phases, however, the current implementation constants favor BTX over BEAT++.

Relative to PFE, the gains appear exactly where the complexity analysis predicts. In precompute(B), BTX is approximately 1.5–1.7× faster than PFE across all tested batch sizes. In open(B), the gap is larger: for $B = 512$, BTX requires only 0.171 ms per ciphertext compared with 0.723 ms per ciphertext for PFE, a speedup of approximately 4.2×. This matches the expected ratio, since PFE requires four pairings per slot in the opening step while BTX requires only one.

On the threshold-MSM phases partialDecrypt(B) and combine(n), BTX and PFE are essentially

identical in the current implementation. For example, at $B = 512$, both schemes require about 0.019 ms per ciphertext in partialDecrypt, and at $n = 16$ both require about 0.03 ms per share in combine.

References

[1] Amit Agarwal, Kushal Babel, Sourav Das, Babak Poorebrahim Gilkalaye, Arup Mondal, Benny Pinkas, Peter Rindal, and Aayush Yadav. Weighted batched threshold encryption with applications to mempool privacy. Cryptology ePrint Archive, Paper 2025/2115, 2025.

[2] Amit Agarwal, Rex Fernando, and Benny Pinkas. Efficiently-thresholdizable batched identity based encryption, with applications. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part III*, volume 16002 of *Lecture Notes in Computer Science*, pages 69–100, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland.

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

[4] Rishiraj Bhattacharyya, Jan Bormet, Sebastian Faust, Pratyay Mukherjee, and Hussien Othman. CCA-secure traceable threshold (ID-based) encryption and application. Cryptology ePrint Archive, Report 2025/341, 2025.

[5] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238. Springer, 2004.

[6] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229, Santa Barbara, CA, USA, August 19–23, 2001. Springer Berlin Heidelberg, Germany.

[7] Dan Boneh, Rohit Nema, Arnab Roy, and Ertem Nusret Tas. Efficient batch threshold encryption using partial fraction techniques. *Cryptology ePrint Archive*, 2026.

[8] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. *Draft 0.6*, 2023.

[9] 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, Paper 2025/1419, 2025.

[10] 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.

[11] 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: 33rd USENIX Security Symposium*, Philadelphia, PA, USA, August 14–16, 2024. USENIX Association.

[12] Arka Rai Choudhuri, Sanjam Garg, Guru-Vamsi Policharla, and Mingyuan Wang. Practical mempool privacy via one-time setup batched threshold encryption. Cryptology ePrint Archive, Report 2024/1516, 2024.

[13] Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wohnig. McFly: Verifiable encryption to the future made practical. In Foteini Baldimtsi and Christian Cachin, editors, *FC 2023: 27th International Conference on Financial Cryptography and Data Security, Part I*, volume 13950 of *Lecture Notes in Computer Science*, pages 252–269, Bol, Brač, Croatia, May 1–5, 2023. Springer, Cham, Switzerland.

[14] Stefan Dziembowski, Sebastian Faust, and Jannik Luhn. Shutter network: Private transactions from threshold cryptography. Cryptology ePrint Archive, Report 2024/1981, 2024.

[15] Rex Fernando, Guru-Vamsi Policharla, Andrei Tonkikh, and Zhuolun Xiang. TrX: Encrypted mempools in high performance BFT protocols. Cryptology ePrint Archive, Report 2025/2032, 2025.

[16] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 152–168, Santa Barbara, CA, USA, August 14–18, 2005. Springer Berlin Heidelberg, Germany.

[17] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold encryption with silent setup. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part VII*, volume 14926 of *Lecture Notes in Computer Science*, pages 352–386, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

[18] Pranav Garimidi, Joseph Bonneau, and Lioba Heimbach. On the limits of encrypted mempools, July 2025. a16z Crypto.

[19] Junqing Gong, Brent Waters, Hoeteck Wee, and David J Wu. Threshold batched identity-based encryption from pairings in the plain model. *Cryptology ePrint Archive*, 2025.

[20] Guillaume Hanrot, Michel Quercia, and Paul Zimmermann. The middle product algorithm i: Speeding up the division and square root of power series. *Applicable Algebra in Engineering, Communication and Computing*, 14(6):415–438, 2004.

[21] Alireza Kavousi, Duc V. Le, Philipp Jovanovic, and George Danezis. BlindPerm: Efficient MEV mitigation with an encrypted mempool and permutation. Cryptology ePrint Archive, Report 2023/1061, 2023.

[22] Simon Langowski, Kaiwen He, and Srinivas Devadas. VROOM: Accelerating (almost all) number-theoretic cryptography using vectorization and the residue number system. Cryptology ePrint Archive, Paper 2026/393, 2026.

[23] Frederik Lührs, Luis Bezzenberger, Francesco Mosterts, Sebastian Faust, and Andreas Erwig. The road towards an encrypted mempool on ethereum. Shutter Docs.

[24] Peyman Momeni, Sergey Gorbunov, and Bohan Zhang. FairBlock: Preventing blockchain front-running with minimal overheads. Cryptology ePrint Archive, Report 2022/1066, 2022.

[25] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *J. Cryptol.*, 15(2):75–96, 2002. Also at *Eurocrypt 1998*.

[26] Shutter Network and Tatu. The completion of our op grant: Defining an encrypted mempool for the op stack, November 2023. Shutter Blog.

[27] Sora Suegami, Shinsaku Ashizawa, and Kyohei Shibano. Constant-cost batched partial decryption in threshold encryption. Cryptology ePrint Archive, Paper 2024/762, 2024.