

Batched and Weighted Threshold Encryption with Silent Setup

Amit Agarwal¹, Rutchathon Chairattana-Apirom², Sourav Das¹, and Babak Poorebrahim Gilkalaye¹

¹ Category Labs

² University of Washington

Abstract. Batched Threshold Encryption (BTE) enables a committee of parties to jointly decrypt any subset of ciphertexts from a large set, while all other ciphertexts remain private. BTE has applications in blockchains, particularly in designing encrypted mempools, where transactions are encrypted until included into a block to prevent maximal extractable value (MEV) attacks. Existing BTE constructions, however, encounter one of the following usability downsides: (a) reliance on expensive distributed key generation (DKG) protocols, (b) dependency on a batch label during ciphertext generation, (c) ciphertexts that are only decryptable if they are encoded with respect to a non-conflicting index in the batch, or (d) no support for weighted threshold policies.

In this work, we present a BTE scheme that simultaneously addresses all of the above issues. It has a silent setup (key generation does not require interaction between parties), is batch and index independent, and supports weighted threshold policies. Our construction is based on prime-order groups with bilinear pairings with security proof in the Generic Group Model. Our ciphertext size is also the shortest among existing silent batched threshold encryption schemes with only 2 group elements in addition to the message length.

1 Introduction

Threshold encryption [15,16] enables a committee of parties to jointly decrypt an arbitrary ciphertext. In many practical applications, however, parties need to decrypt a large batch of ciphertexts rather than a single ciphertext. Naively repeating the threshold decryption protocol for every ciphertext results in communication overhead being linear in the batch size. To address this limitation, the notion of Batched Threshold Encryption (BTE) was introduced. BTE enables a committee to decrypt an arbitrarily chosen subset, or batch, of ciphertexts from a large pool while ensuring that all remaining ciphertexts remain private [13,3,11,10].

A key property of BTE is scalability: the total party-to-party communication required for decryption grows sublinearly with the batch size and is independent of the overall pool size. This makes BTE a natural building block for encrypted mempools, where transactions remain encrypted until they are selected for inclusion in a block. Encrypted mempools are a promising approach for mitigating

maximal extractable value (MEV) [20,28,23]. Since practical encrypted mempools operate under tight latency constraints, reducing both communication and computation costs for batch decryption while supporting large batch sizes is an important problem.

Although existing BTE constructions address the scalability problem successfully by making the communication cost completely *independent* of the batch size, several qualitative properties are important for making BTE suitable for real-world deployments:

1. **No batch specific restrictions.**

- **Index independence:** An index *dependent* BTE scheme binds each ciphertext to an index i chosen from a polynomial-sized index space and two different ciphertexts formed w.r.t. same index are not allowed to be part of the same decryption batch. Ideally, a ciphertext should not be tied to such indices and any pair of arbitrary ciphertexts should be eligible for simultaneous inclusion in a decryption batch. Index independence is particularly important in blockchain systems, where index dependence can become a powerful censorship vector [7]. For example, after observing a target ciphertext associated with a particular index, an adversary could generate competing ciphertexts with the same index and incentivize its inclusion, preventing the target ciphertext from inclusion in the batch.
- **Batch independence:** A batch *dependent* BTE scheme binds each ciphertext tied to a *batch-specific* label. In other words, each distinct batch has a pre-assigned unique label ℓ (for example, this label can be the block number in a typical blockchain). The encryption algorithm takes ℓ as an input and produces a ciphertext ct_ℓ that can only be decrypted when batch ℓ is processed for decryption or becomes useless. Ideally, a ciphertext should not be tied to a particular batch label and should be a candidate for inclusion in any arbitrary batch. This property is particularly important in blockchain systems, where targeting a specific future block may be impractical or undesirable. Even if a block can be targeted, it may be full or the proposer may omit the transaction for other reasons. In such cases, the encryptor may need to target multiple candidate blocks, potentially creating a significant bandwidth bottleneck.

2. **Weighted threshold support:** The party-to-party communication complexity should not grow with, or depend on, the individual weights assigned to parties. This setting is relevant in Proof-of-Stake blockchains where each validator is assigned a weight proportional to its stake in the system.
3. **Silent setup:** The key-generation protocol should be non-interactive and should not require an additional coordination primitive such as an Asynchronous Common Subset (ACS) protocol. The schemes may still require a one-time setup to generate public parameters; however, these can be reused for subsequent key generations.

Existing approaches and their limitations. Recent work has made substantial progress towards practical BTE. Choudhuri et al. [13] introduced BTE and

achieved low decryption communication; however, their construction requires an MPC-based setup for every decryption batch. Subsequent schemes [14,3,29] eliminate this repeated setup cost. Nevertheless, these constructions require each ciphertext to be generated with respect to a batch label and therefore do not provide batch independence.

BEAT-MEV [11] provides an alternative construction in which ciphertexts are not tied to a particular decryption session. However, BEAT-MEV suffers from the aforementioned index-dependency limitation. BEAT++ [1] improves the computational efficiency of BEAT-MEV and supports weighted validators, while retaining the index-dependency limitation.

Three other works: BTX [2], Fernando et al. [18], and Boneh et al. [9] address the index-dependency and batch-dependency issues. Fernando et al. [18], however, requires a large common reference string (CRS) whose size grows proportionally with the number of decryption batches. Boneh et al. [9] and BTX [2] avoid this large-CRS requirement, but instead rely on non-standard and computationally expensive setup procedures. Notably, these constructions do not support weighted decryptors and require a DKG.

In an effort to avoid expensive distributed key generation protocols (DKG), Garg et al. [19] introduced *silent* threshold encryption. In a silent threshold encryption, decryptors (parties) obtain their threshold secret keys using a “TLS-style” setup. In particular, each party locally generates its public and secret key pair and announces the public key. Thereafter, any committee of N parties can come together to derive the public threshold encryption key *without any interaction*. BEAST-MEV [10] constructs a silent BTE scheme by combining the silent (unbatched) encryption scheme from [19] and batched (non-silent) encryption scheme from [11] in a black-box fashion. Subsequently, Gong et al. [21] proposed a Silent Batched IBE construction, which immediately yields a silent BTE scheme. However, these approaches either do not achieve index-independence or batch-independence.

We summarize the recent developments in Table 1. Observe that no prior construction satisfies all of the desirable qualitative properties. Thus, we ask the following open question:

Can we construct a batched threshold encryption scheme that is index and batch independent, supports weighted decryptors, and has a silent setup?

Our contribution. In this work, we affirmatively answer the above question and present the first batched threshold encryption that simultaneously satisfies all four aforementioned properties while maintaining comparable computational efficiency as prior work with weighted support or silent setup.³ Moreover, our ciphertext size is the smallest among existing silent threshold schemes and consists of only 2 group elements and the message length, and our partial decryption size (which captures communication cost) is only one group element. We prove indistinguishability under chosen-plaintext attacks (IND-CPA)

³ We give efficiency metric comparison in Table 1.

Table 1. Comparison of batched threshold encryption schemes based on a pairing-friendly cryptographic group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of prime order p . Here, N denotes the number of parties performing threshold decryption, W is the total weight across all parties, w_ℓ is the individual weight of party $\ell \in [N]$, and B is the batch size (number of ciphertexts to be decrypted). We compare the schemes in terms of i) qualitative properties: silent setup (**Sil**), index independence (**Idx Ind**), batch independence (**Bat Ind**), and weighted threshold (**Wt**), ii) quantitative costs: size (in terms of group elements) of decryption key, ciphertext, per-party communication and computation (in terms of group operations) for threshold decryption, iii) cryptographic hardness assumption. Except the ciphertext size, all cost expressions are in terms of asymptotics. **Red texts** describe additional sizes/assumptions needed for malicious security (including CCA security and partial decryption verification). The cost for schemes which do not natively support weighted thresholds have been adapted to weighted setting as per the naive “virtualization” approach (see Section 2.3) for a fair comparison. To get a sense of cost comparison accross schemes in the *unweighted* setting, one can set $W = N$ (as unweighted is simply a special case of weighted setting).

Scheme	Sil	Idx Ind	Bat Ind	Wt	Decryption key size (no. of group elements)	Ciphertext size (excluding message length)	Communication cost (download per party)	Decryption computation cost	Hardness assumption
[14,3,29]	✓				$B + W$	$3\mathbb{G}_2 + 2\mathbb{Z}_p$	$W\mathbb{G}_1$	$\frac{W}{B \log^2 B}$	GGM + ROM
TrX [†] [18]	✓	✓			$KB + W$	$1\mathbb{G}_1 + 1\mathbb{G}_2 + 2\mathbb{Z}_p$	$W\mathbb{G}_1$	$\frac{W}{B \log^2 B}$	GGM + ROM
PFE [9]	✓		✓		$B + WB$	$1\mathbb{G}_1 + 2\mathbb{Z}_p$	$W\mathbb{G}_1$	$\frac{W}{B \log B}$	q -type + SENIZK
BTX [2,25]	✓		✓		$B + WB$	$1\mathbb{G}_1 + 2\mathbb{Z}_p$	$W\mathbb{G}_1$	$\frac{W}{B \log B}$	q -type + SENIZK
Weighted BEAT ⁺⁺ [1]		✓		✓	$B + WB$	$2\mathbb{G}_1 + 2\mathbb{Z}_p$	$N\mathbb{G}_1$	$\frac{BW}{B \log B}$	GGM + ROM
GW26 [21]	✓	✓			NWB	$2\mathbb{G}_1 + 2\mathbb{G}_2 + 2\mathbb{Z}_p$	$2W\mathbb{G}_2 + W\mathbb{Z}_p$	$\frac{WB}{B \log^2 B}$	q -type
BEAST-MEV [‡] [10]	✓			✓	$B + CWN$	$(C+4)\mathbb{G}_1 + 5\mathbb{G}_2 + 6C\mathbb{G}_1 + (2C+6)\mathbb{Z}_p$	$W\mathbb{G}_1$	$\frac{B \log B + CW + C}{DL(Bp^{1/C})}$	GGM + ROM
This work	✓	✓	✓	✓	$W^2 + BW$	$2\mathbb{G}_1 + 2\mathbb{Z}_p$	$N\mathbb{G}_1$	$\frac{BW}{B \log B}$	GGM + ROM

†: TrX [18] requires an additional parameter K , denoting an a priori upper bound on the total number of decryption batches. The size of public parameter grows linearly with K .

‡: For BEAST-MEV, C denotes the number of decomposition of $\mathbb{Z}_p$ element into small integers, $DL(n)$ denotes time complexity to compute discrete logarithm on a domain size n .

security in the Generic Group Model (GGM) [27,24] and rely on simulation-extractable non-interactive zero-knowledge to achieve indistinguishability under chosen-ciphertext attacks (IND-CCA) security as in prior works [2,9].

Our construction is inspired by the technical ideas in Ferveo [6], Waters–Wu’26 [30], and BTX [2], which we combine in a non-trivial and novel manner. We give a detailed technical overview of our construction design in the next section.

Additional Related Work. Batched Encryption has also been studied in a different context of optimizing garbled circuits [5,4,17]. Instead of group-based constructions, recent works have also explored post-quantum alternatives. Boneh Laufer Tas [8] and Xiong, Zhang, and Chen [31] constructed batched threshold encryption from lattices. Hall-Andersen, Simkin, and Wagner [22] gave a generic construction of silent threshold encryption from PKEs and NIKEs with lattice-based instantiations.

Paper outline. Section 2 gives an overview of our construction design and techniques. Section 3 gives the necessary preliminaries for our construction. Section 4 formalizes the syntax and security of batched and weighted threshold encryption with silent setup. Our construction, its extensions, and security guarantees can be found in Section 5. In Appendix A, we discuss our usage of generative AI tools.

2 Technical Overview

We will start with the well-known threshold ElGamal encryption scheme (which doesn’t support any of these properties) and describe how we add the properties we seek in a step-by-step fashion. Figure 1 depicts the overall flow of ideas.

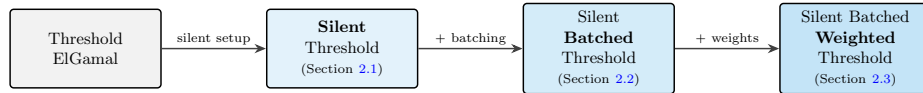


Fig. 1. Roadmap of the technical overview. Starting from plain threshold ElGamal, we first replace the interactive DKG by a silent setup, then add batched decryption, and finally add weighted thresholds on top, arriving at a threshold encryption scheme that is simultaneously silent, batched and weighted.

Background. The ElGamal encryption scheme is a public key encryption scheme that is based on the hardness of the discrete logarithm problem defined w.r.t. a cyclic group $\mathbb{G}$ of large prime order p . We will use the bracket notation $\llbracket x \rrbracket$ to denote the group element obtained by applying the group operation $+$ to the generator for x times. Such a group element can be multiplied by a scalar $c \in \mathbb{Z}_p$

to obtain the group element $c \cdot \llbracket x \rrbracket = \llbracket c \cdot x \rrbracket$, and two group elements $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$ can be added to obtain the group element $\llbracket x + y \rrbracket$. The discrete logarithm problem is the problem of finding x given a group element $\llbracket x \rrbracket$ for a uniformly random $x \in \mathbb{Z}_p$.

The ElGamal encryption scheme works as follows. The public key is a group element $\llbracket \alpha \rrbracket$ for a secret key $\alpha \in \mathbb{Z}_p$, and the encryption of a message $m \in \mathbb{G}$ is a pair of group elements $(\text{ct}^*, \text{ct}^\bullet) := (\llbracket r \rrbracket, m + r \cdot \llbracket \alpha \rrbracket)$ for a uniformly random $r \in \mathbb{Z}_p$. To decrypt a ciphertext, the secret key α is used to compute the partial decryption element $\text{pd} = \alpha \cdot \text{ct}^* = \llbracket \alpha \cdot r \rrbracket$, and the message is recovered as $m = \text{ct}^\bullet - \text{pd}$. In the threshold ElGamal scheme, the secret key α is shared among N parties using a T -out-of- N threshold secret sharing scheme, and each party i has a share α_i . To decrypt a ciphertext, each party computes and broadcasts their *share* of partial decryption $\text{pd}_i = \alpha_i \cdot \llbracket r \rrbracket$. Thereafter, any T parties can combine their partial decryption shares pd_i to reconstruct pd and then recover the message as before. The main limitation of the aforementioned threshold version of ElGamal scheme is that the parties need to run a Distributed Key Generation (DKG), to securely generate the public key $\llbracket \alpha \rrbracket$ and the corresponding secret shares α_i . Below we describe an approach to make the threshold ElGamal encryption scheme silent.

2.1 Making the Threshold ElGamal Silent

Warmup - A scheme with large ciphertexts. A folklore approach [26,22] for obtaining a silent threshold encryption scheme from an underlying (non-thresholdized) public-key encryption scheme is by running N independent instances of the underlying encryption scheme, one for each party, and encrypting a T -out-of- N secret share of the message under each instance.⁴ Here, we will outline a slightly different version of this folklore scheme specialized for the ElGamal encryption scheme, which will be more natural for the progression of ideas in this work. Instead of individually encrypting shares of the actual message m under each party's public key, we will encrypt secret shares of a one-time pad group element that masks the message m .

More concretely, in a one-time setup phase (that's independent of the parties involved and can be reused), we sample a random group element $\llbracket t \rrbracket \in \mathbb{G}$, secret-share it using a T -out-of- N threshold secret sharing scheme to derive

⁴ Concretely, the scheme works in the following way. Each party $i \in [N]$ generates their own key pair $(\text{pk}_i, \text{sk}_i)$ and announces the public key pk_i . The aggregated public key pk_{agg} is then computed as the collection of all individual public keys, i.e., $\text{pk}_{\text{agg}} = \{\text{pk}_i\}_{i \in [N]}$. In order to encrypt a message m , we first generate threshold secret-shares of the message using a T -out-of- N threshold secret sharing scheme to derive $m_1, \dots, m_N$. Thereafter, each share m_i is encrypted under the public key pk_i to derive $\text{ct}_i := \text{Enc}_{\text{pk}_i}(m_i)$ and the final ciphertext ct is the collection of all individual ciphertexts $\text{ct} = \{\text{ct}_i\}_{i \in [N]}$. In order to decrypt, the i^{th} party uses their secret key sk_i to decrypt ct_i and recover m_i , and any T parties can combine their recovered shares m_i to reconstruct the original message m .

$\llbracket t_1 \rrbracket, \dots, \llbracket t_N \rrbracket$ and then make the following elements publicly available as part of a public parameters ($\mathbf{pp}$).

$$\mathbf{pp} = \begin{cases} \llbracket t_1 \rrbracket, \dots, \llbracket t_N \rrbracket \\ \llbracket t \rrbracket \end{cases}$$

During the key generation phase, each party i samples its secret key $\mathbf{sk}_i := \alpha_i \leftarrow \mathbb{Z}_p$ and announces the corresponding public key

$$\mathbf{pk}_i := \llbracket \alpha_i \rrbracket$$

Thereafter, the aggregated public key is computed by adding the i^{th} public key $\mathbf{pk}_i := \llbracket \alpha_i \rrbracket$ with the $\mathbf{pp}$ element $\llbracket t_i \rrbracket$ and collecting all such sums for $i \in [N]$ in the following way:

$$\mathbf{pk}_{\text{agg}} := \llbracket t_1 + \alpha_1 \rrbracket, \dots, \llbracket t_N + \alpha_N \rrbracket$$

In order to encrypt a message m , we sample $r \leftarrow \mathbb{Z}_p$ as usual and output the ciphertext $\mathbf{ct}$ as follows:

$$\mathbf{ct} = \underbrace{\llbracket r \rrbracket}_{\mathbf{ct}^*}, \underbrace{\llbracket r \cdot (t_1 + \alpha_1) \rrbracket}_{\mathbf{ct}_1}, \dots, \underbrace{\llbracket r \cdot (t_N + \alpha_N) \rrbracket}_{\mathbf{ct}_N}, \underbrace{\llbracket r \cdot t \rrbracket}_{\mathbf{ct}^\bullet} + m$$

$\xrightarrow{r \cdot \mathbf{pk}_{\text{agg}}} \quad \xrightarrow{\text{one-time pad}}$

To decrypt the ciphertext $\mathbf{ct}$, each party i computes their partial decryption share as

$$\begin{aligned} \mathbf{pd}_i &:= \alpha_i \cdot \mathbf{ct}^* \\ &= \llbracket \alpha_i \cdot r \rrbracket \end{aligned}$$

and broadcasts it, which can be used to derive the i^{th} share of the one-time pad element.

$$\begin{aligned} & i^{\text{th}} \text{ share of the one-time pad} \\ & \underbrace{\llbracket r \cdot t_i \rrbracket}_{\text{one-time pad share}} := \mathbf{ct}_i - \mathbf{pd}_i \end{aligned} \tag{1}$$

Thereafter, any T such one-time pad shares $\llbracket r \cdot t_i \rrbracket$ can be combined to reconstruct the one-time pad $\llbracket r \cdot t \rrbracket$ and then recover the message as $m = \mathbf{ct}^\bullet - \llbracket r \cdot t \rrbracket$.

At a high-level, the security of the scheme follows from the fact that, as per Equation (1), the i^{th} share of the one-time pad $\llbracket r \cdot t_i \rrbracket$ is hidden unless the i^{th} party contributes its partial decryption share $\mathbf{pd}_i$. This implies that if less than T parties contribute their partial decryption shares, the one-time pad $\llbracket r \cdot t \rrbracket$ remains hidden, ensuring the semantic security of the message m .

Reducing ciphertext size to constant. The main drawback of the above approach is that the ciphertext size grows linearly with the number of parties N . Now we will describe how to reduce the ciphertext size to a constant, independent

of N . First note that the underlying reason for the $O(N)$ sized ciphertext is the $O(N)$ sized aggregate public key pk_{agg} which is simply the collection of individual $O(N)$ many public keys. If we can somehow compress the aggregate public key to a constant size, then the ciphertext can also be reduced to a constant size. It turns out that this can be achieved by moving to a pairing-friendly group and introducing a powers correlation⁵ into the warmup scheme.

A pairing-friendly cryptographic group is defined by a triple of groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$, each of large prime order p , along with an efficiently computable bilinear map $\circ : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ known as the “pairing” operation. For simplicity, we will describe our ideas w.r.t. a symmetric pairing setting where $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$, although the ideas can be extended to the asymmetric setting as well. We will stick to the previously introduced bracket notation to denote group elements and use $\llbracket \cdot \rrbracket_T$ to distinguish a target group element from a source group element $\llbracket \cdot \rrbracket$. As per this notation, we have the following standard pairing property: $\llbracket x \rrbracket \circ \llbracket y \rrbracket = \llbracket x \cdot y \rrbracket_T$.

Now we will describe how we leverage a powers correlation to get a silent scheme with constant-sized ciphertext. As a first step, we will modify the public parameters by correlating the existing secret t and its shares t_i with powers of a random value $c \leftarrow \mathbb{Z}_p$ in the following way:

$$\text{pp} = \begin{cases} \llbracket c^1 \rrbracket, \dots, \llbracket c^{2N} \rrbracket \\ \llbracket c^1 \cdot t_1 \rrbracket, \dots, \cancel{\llbracket c^{N+1} \cdot t_1 \rrbracket}, \dots, \llbracket c^{2N} \cdot t_1 \rrbracket \\ \dots \\ \llbracket c^1 \cdot t_N \rrbracket, \dots, \cancel{\llbracket c^{N+1} \cdot t_N \rrbracket}, \dots, \llbracket c^{2N} \cdot t_N \rrbracket \\ \llbracket c^{N+1} \cdot t \rrbracket_T \end{cases} \quad (2)$$

Crucially, we have the product of c^{N+1} and shares t_i missing in pp —this is necessary for the CPA security of the silent scheme that we will design.

During the key generation phase, each party i samples its secret key $\text{sk}_i := \alpha_i$ and announces the following modified public key

$$\text{pk}_i := \llbracket c^1 \cdot \alpha_i \rrbracket, \dots, \cancel{\llbracket c^{N+1} \cdot \alpha_i \rrbracket}, \dots, \llbracket c^{2N} \cdot \alpha_i \rrbracket$$

Thereafter, the aggregated public key is computed in the following way:

$$\begin{aligned} \text{pk}_{\text{agg}} &:= \sum_{i=1}^N \llbracket c^i \cdot t_i \rrbracket + \llbracket c^i \cdot \alpha_i \rrbracket \\ &= \sum_{i=1}^N \llbracket c^i \cdot (t_i + \alpha_i) \rrbracket \end{aligned} \quad (3)$$

Note that the aggregated public key pk_{agg} is now a single group element rather than a vector of N elements as in the warmup scheme.⁶

⁵ Similar techniques have been leveraged in recent works [1,2,30,21] although the exact usage and the resulting scheme is different from what we obtain in this work.

⁶ We remark that this pk_{agg} structure is similar to key aggregation in the silent scheme of [30].

In order to encrypt a message $m \in \mathbb{G}_T$, we sample $r \leftarrow \mathbb{Z}_p$ as usual and output the following *constant* sized ciphertext:

$$\text{ct} = \underbrace{\llbracket r \rrbracket}_{\text{ct}^*}, \underbrace{\sum_{i=1}^N \llbracket r \cdot c^i \cdot (t_i + \alpha_i) \rrbracket}_{\text{ct}^\dagger = r \cdot \text{pk}_{\text{agg}}}, \underbrace{\llbracket r \cdot c^{N+1} \cdot t \rrbracket_T + m}_{\text{one-time pad}, \text{ct}^\bullet} \quad (4)$$

To decrypt the ciphertext ct , each party i computes their partial decryption share (as before without any changes) and broadcasts it.

$$\begin{aligned} \text{pd}_i &:= \alpha_i \cdot \text{ct}^* \\ &= \llbracket \alpha_i \cdot r \rrbracket \end{aligned}$$

Similar to the warmup scheme, the next step is to derive the shares of one-time pad $\llbracket r \cdot c^{N+1} \cdot t_i \rrbracket_T$. First, we note that pairing a ciphertext element $\text{ct}^\dagger$ with a pp element $\llbracket c^{N+1-i} \rrbracket$ allows us to isolate the desired term from the undesired terms as follows:

$$\begin{aligned} \text{ct}^\dagger \circ \underbrace{\llbracket c^{N+1-i} \rrbracket}_{\text{element of pp}} &= \llbracket r \cdot c^{N+1} \cdot (t_i + \alpha_i) \rrbracket_T + \sum_{j \in [N], j \neq i} \llbracket r \cdot c^{N+1+j-i} \cdot (t_j + \alpha_j) \rrbracket_T \\ &= \underbrace{\llbracket r \cdot c^{N+1} \cdot t_i \rrbracket_T}_{\substack{\text{desired term:} \\ i^{\text{th}} \text{ share of the one-time pad}}} + \underbrace{\llbracket r \cdot c^{N+1} \cdot \alpha_i \rrbracket_T}_{\text{first undesired term}} \\ &\quad + \underbrace{\sum_{j \in [N], j \neq i} \llbracket r \cdot c^{N+1+j-i} \cdot (t_j + \alpha_j) \rrbracket_T}_{\text{second undesired term}} \end{aligned} \quad (5)$$

Now, we will try to cancel off the undesired terms to isolate the desired term $\llbracket r \cdot c^{N+1} \cdot t_i \rrbracket_T$. To do so, we make the following observations:

- The first undesired term can be separately computed by pairing the partial decryption share of i^{th} party with a pp element in the following way.

$$\text{pd}_i \circ \underbrace{\llbracket c^{N+1} \rrbracket}_{\text{element of pp}} = \underbrace{\llbracket r \cdot c^{N+1} \cdot \alpha_i \rrbracket_T}_{\text{first undesired term}} \quad (6)$$

- The second undesired term can be separately computed by pairing the ciphertext element ct^* with the appropriate pp elements and public keys as

follows.

$$\begin{aligned}
& \text{ct}^* \circ \sum_{j \in [N], j \neq i} \left(\underbrace{\llbracket c^{N+1+j-i} \cdot t_j \rrbracket}_{\text{element of pp}} + \underbrace{\llbracket c^{N+1+j-i} \cdot \alpha_j \rrbracket}_{\text{element of pk}_j} \right) \\
&= \llbracket r \rrbracket \circ \sum_{j \in [N], j \neq i} \left(\llbracket c^{N+1+j-i} \cdot t_j + c^{N+1+j-i} \cdot \alpha_j \rrbracket \right) \\
&= \underbrace{\sum_{j \in [N], j \neq i} \llbracket r \cdot c^{N+1+j-i} \cdot (t_j + \alpha_j) \rrbracket_{\mathbb{T}}}_{\text{second undesired term}} \tag{7}
\end{aligned}$$

Hence, we can cancel off the first and second undesired terms computed via Equation (6) and Equation (7) respectively, from Equation (5) to obtain the desired one-time pad share $\llbracket r \cdot c^{N+1} \cdot t_i \rrbracket_{\mathbb{T}}$ in the following way:

i^{th} share of the one-time pad

$$\begin{aligned}
& \underbrace{\llbracket r \cdot c^{N+1} \cdot t_i \rrbracket_{\mathbb{T}}}_{\text{desired share}} := \text{ct}^\dagger \circ \underbrace{\llbracket c^{N+1-i} \rrbracket}_{\text{element of pp}} \\
& \quad - \text{pd}_i \circ \underbrace{\llbracket c^{N+1} \rrbracket}_{\text{element of pp}} \\
& \quad - \text{ct}^* \circ \sum_{j \in [N], j \neq i} \left(\underbrace{\llbracket c^{N+1+j-i} \cdot t_j \rrbracket}_{\text{element of pp}} + \underbrace{\llbracket c^{N+1+j-i} \cdot \alpha_j \rrbracket}_{\text{element of pk}_j} \right) \tag{8}
\end{aligned}$$

Now, we can combine T such distinct one-time pad shares to reconstruct the one-time pad $\llbracket r \cdot c^{N+1} \cdot t \rrbracket_{\mathbb{T}}$ and then finally recover the message as $m = \text{ct}^\bullet - \llbracket r \cdot c^{N+1} \cdot t \rrbracket_{\mathbb{T}}$. This completes the description of our silent scheme.

At a high-level, the security of the scheme follows from the fact that, as per Equation (6), computing the first undesired term requires the i^{th} party to contribute its partial decryption share pd_i . In the absence of such a contribution, the first undesired term is essentially hidden and cannot be canceled from Equation (5), ensuring that the corresponding one-time pad share $\llbracket r \cdot c^{N+1} \cdot t_i \rrbracket_{\mathbb{T}}$ also remains hidden. This implies that if less than T parties contribute their partial decryption shares, the one-time pad $\llbracket r \cdot c^{N+1} \cdot t \rrbracket_{\mathbb{T}}$ remains hidden, ensuring the semantic security of the message m .

2.2 Supporting Batching

In the aforementioned silent scheme, the size of the partial decryption share pd_i (which captures the communication cost for distributed decryption process) is a single group element. If we run the same scheme for decrypting a batch of B ciphertexts, the communication cost would naively scale linearly with B . Now we

will describe how to modify the scheme to support batching efficiently, keeping the communication cost constant regardless of the batch size. At a high-level, we will achieve this by adding yet another powers correlation in the public keys of the parties, which will allow each party to compute a single aggregated partial decryption share for the entire batch.

Concretely, we will have the same setup phase with exactly the same $\mathbf{pp}$ as in unbatched setting given by Equation (2). However, during the key-generation phase, each party i samples a slightly different secret key $\mathbf{sk}_i := (\alpha_i, \mathbf{s}_i)$ and announces the following modified public key:

$$\mathbf{pk}_i := \begin{cases} \llbracket c^1 \cdot \alpha_i \rrbracket, \dots, \llbracket c^{N+1} \cdot \alpha_i \rrbracket, \dots, \llbracket c^{2N} \cdot \alpha_i \rrbracket \\ \llbracket c^{N+1} \cdot \alpha_i \cdot s_i^{-B} \rrbracket, \dots, \llbracket c^{N+1} \cdot \alpha_i \cdot s_i^0 \rrbracket, \dots, \llbracket c^{N+1} \cdot \alpha_i \cdot s_i^B \rrbracket \end{cases} \quad (9)$$

The aggregate key $\mathbf{pk}_{\text{agg}}$ is computed exactly as in the unbatched setting given by Equation (3) and the ciphertext structure also remains the same as in the unbatched setting given by Equation (4). To decrypt a batch of ciphertexts $(\mathbf{ct}_1, \dots, \mathbf{ct}_B)$, where each $\mathbf{ct}_j = (\mathbf{ct}_j^*, \mathbf{ct}_j^\dagger, \mathbf{ct}_j^\bullet)$, each party i computes the following modified partial decryption share⁷ and broadcasts it.

$$\begin{aligned} \mathbf{pd}_i &:= \sum_{j \in [B]} \mathbf{ct}_j^* \cdot \mathbf{s}_i^j \\ &= \llbracket r_1 \cdot \mathbf{s}_i^1 + \dots + r_B \cdot \mathbf{s}_i^B \rrbracket \end{aligned} \quad (10)$$

Similar to the silent unbatched scheme, the next step is to derive the shares of one-time pad $\llbracket r_j \cdot c^{N+1} \cdot t_i \rrbracket_{\mathbb{T}}$ for each $j \in [B]$. To do this, we note that the modified structure of the partial decryption share $\mathbf{pd}_i$ *only* affects the computation of the first undesired term in Equation (5), namely $\llbracket r_j \cdot c^{N+1} \cdot \alpha_i \rrbracket_{\mathbb{T}}$, as shown in Equation (6). Everything else remains unchanged (because $\mathbf{pd}_i$ is not used anywhere else and ciphertext is identical), and the rest of the derivation follows exactly as in the unbatched setting. Having said that, we will now focus on the modified computation (meaning a modified version of Equation (6)) needed for computing the $\llbracket r_j \cdot c^{N+1} \cdot \alpha_i \rrbracket_{\mathbb{T}}$ term. In a batched setting, such terms need to be computed for all $j \in [B]$.

To that end, we first note that by pairing the modified partial decryption share $\mathbf{pd}_i$ with the appropriate element of the individual public key $\mathbf{pk}_i$, we can get the $\llbracket r_j \cdot c^{N+1} \cdot \alpha_i \rrbracket_{\mathbb{T}}$ term along with some extra terms as shown below.

$$\begin{aligned} \mathbf{pd}_i \circ \underbrace{\llbracket c^{N+1} \cdot \alpha_i \cdot \mathbf{s}_i^{-j} \rrbracket}_{\text{element of } \mathbf{pk}_i} &= \llbracket r_1 \cdot \mathbf{s}_i^1 + \dots + r_B \cdot \mathbf{s}_i^B \rrbracket \circ \llbracket c^{N+1} \cdot \alpha_i \cdot \mathbf{s}_i^{-j} \rrbracket \\ &= \llbracket r_j \cdot c^{N+1} \cdot \alpha_i \rrbracket_{\mathbb{T}} + \underbrace{\sum_{k \in [B], k \neq j} \llbracket r_k \cdot c^{N+1} \cdot \alpha_i \cdot \mathbf{s}_i^{(k-j)} \rrbracket_{\mathbb{T}}}_{\text{extra terms}} \end{aligned} \quad (11)$$

⁷ This structure resembles the batch decryption of BTX [2].

Now, the extra terms in the R.H.S need to be cancelled off to isolate the $\llbracket r_j \cdot c^{N+1} \cdot \alpha_i \rrbracket_{\mathsf{T}}$ term. To do this, we observe that the extra terms can be computed separately by pairing the ciphertext element ct_j^* with the appropriate elements in the individual public keys in the following way.

$$\begin{aligned} \forall k \in [B], k \neq j : \quad \text{ct}_k^* \circ \underbrace{\llbracket c^{N+1} \cdot \alpha_i \cdot s_i^{(k-j)} \rrbracket}_{\text{element of } \text{pk}_i} &= \llbracket r_k \rrbracket \circ \llbracket c^{N+1} \cdot \alpha_i \cdot s_i^{(k-j)} \rrbracket \quad (12) \\ &= \underbrace{\llbracket r_k \cdot c^{N+1} \cdot \alpha_i \cdot s_i^{(k-j)} \rrbracket}_{\text{extra term}} \end{aligned}$$

By combining Equation (11) with Equation (12), we get the following expression for computing the $\llbracket r_j \cdot c^{N+1} \cdot \alpha_i \rrbracket_{\mathsf{T}}$ term.

$$\llbracket r_j \cdot c^{N+1} \cdot \alpha_i \rrbracket_{\mathsf{T}} = \text{pd}_i \circ \underbrace{\llbracket c^{N+1} \cdot \alpha_i \cdot s_i^{-j} \rrbracket}_{\text{element of } \text{pk}_i} - \sum_{k \in [B], k \neq j} \underbrace{\text{ct}_k^* \circ \llbracket c^{N+1} \cdot \alpha_i \cdot s_i^{(k-j)} \rrbracket}_{\text{element of } \text{pk}_i}$$

This completes the explanation for how we can compute the $\llbracket r_j \cdot c^{N+1} \cdot \alpha_i \rrbracket_{\mathsf{T}}$ term in the batched setting. For completeness, note that the R.H.S of the above equation will now be plugged into Equation (8) to obtain the following modified computation for one-time pad share $\llbracket r_j \cdot c^{N+1} \cdot t_i \rrbracket_{\mathsf{T}}$, corresponding to the i^{th} party and the j^{th} ciphertext in the batch, as follows.

$$\begin{aligned} \overset{i^{\text{th}} \text{ share of the one-time pad}}{\llbracket r_j \cdot c^{N+1} \cdot t_i \rrbracket_{\mathsf{T}}} &:= \text{ct}_j^\dagger \circ \underbrace{\llbracket c^{N+1-i} \rrbracket}_{\text{element of pp}} \quad (13) \\ &- \left(\underbrace{\text{pd}_i \circ \llbracket c^{N+1} \cdot \alpha_i \cdot s_i^{-j} \rrbracket}_{\text{element of } \text{pk}_i} - \sum_{k \in [B], k \neq j} \underbrace{\text{ct}_k^* \circ \llbracket c^{N+1} \cdot \alpha_i \cdot s_i^{(k-j)} \rrbracket}_{\text{element of } \text{pk}_i} \right) \\ &- \text{ct}_j^* \circ \sum_{k \in [N], k \neq i} \left(\underbrace{\llbracket c^{N+1+k-i} \cdot t_k \rrbracket}_{\text{element of pp}} + \underbrace{\llbracket c^{N+1+k-i} \cdot \alpha_k \rrbracket}_{\text{element of } \text{pk}_k} \right) \end{aligned}$$

As mentioned in the unbatched setting, we can combine T such distinct one-time pad shares to reconstruct the one-time pad $\llbracket r_j \cdot c^{N+1} \cdot t \rrbracket_{\mathsf{T}}$ and then finally recover the message as $m_j = \text{ct}_j^\bullet - \llbracket r_j \cdot c^{N+1} \cdot t \rrbracket_{\mathsf{T}}$ for all $j \in [B]$. This completes the description of our silent *batched* scheme. The security of the scheme follows a similar intuitive argument as in the unbatched setting.

2.3 Extending to Weighted Threshold

We now discuss how to extend our threshold encryption scheme to support weighted thresholds, where each party i has an associated public weight w_i , and

the decryption succeeds if the sum of the weights of the participating parties is at least the threshold T . Let's say that the sum of weights of all parties is equal to some public value W . If we naively apply the so-called “party virtualization” technique, we would replace each party i with w_i many virtual parties, where each virtual party has weight 1 and then run an unweighted threshold encryption scheme with W parties. However, this approach would increase the size of partial decryption share of (real) party i by a multiplicative factor of its weight w_i . This would increase the total communication cost of the distributed decryption protocol to be proportional to the total weight W , which is undesirable. Instead, we aim to design a more efficient solution that keeps the communication cost independent of the weights.

To that end, we make a simple and interesting observation in the silent batched *unweighted* scheme that we described earlier. Recall that the secret key of each party i was a pair of field elements $\mathbf{sk}_i := (\alpha_i, \mathbf{s}_i)$. If the party i has a weight w_i , the naive virtualization approach would result in party i using w_i many such independent secret keys, one for each virtual party, thereby resulting in w_i many distinct partial decryption shares. Our key observation is that, as per Equation (10), only one of the terms of the secret key, namely $\mathbf{s}_i$, is actually used for computing the partial decryption share $\mathbf{pd}_i$; α_i is not used at all in computing $\mathbf{pd}_i$ and its usage is only limited to generating the public key $\mathbf{pk}_i$ as per Equation (9). Hence, we can retain the same $\mathbf{s}$ term for all virtual parties corresponding to the a real party i , and have w_i many independent α terms. Concretely, we will have the following structure for the secret key of party i :

$$\mathbf{sk}_i := (\alpha_{i,1}, \dots, \alpha_{i,w_i}, \mathbf{s}_i)$$

Now, party i will use a single $\mathbf{s}_i$ term for computing its partial decryption share $\mathbf{pd}_i$ exactly as in the unweighted scheme (Equation (10)), while the multiple $\alpha_{i,j}$ terms will be used for generating the corresponding public keys for the virtual parties as per Equation (9) w.r.t. the same $\mathbf{s}_i$ term. During decryption, when we compute the one-time pad shares $\llbracket r_j \cdot c^{W_{\max}+1} \cdot t_i \rrbracket_{\mathbb{T}}$ ⁸ as per Equation (13), we will treat the virtual parties as independent entities, but use the same $\mathbf{pd}_i$ term for all virtual parties corresponding to the real party i . This completes the description of the weighted threshold extension of our silent batched scheme. We provide the formal protocol construction in Figure 4 along with security claims in Section 5.2.

FURTHER EXTENSIONS. The construction sketched so far fixes the threshold T when generating the public parameters $\mathbf{pp}$. This might not suffice for intended applications. However, our scheme in Section 5.1 can be extended to support flexible threshold, where the threshold can be chosen during encryption. This follows from similar ideas used in [30] by introducing dummy parties into $\mathbf{pp}$ to extend their silent threshold encryption to support flexible threshold. We discuss this briefly in Section 5.1 and more formally in Section G.1.

⁸ Here, number of parties N is replaced with the maximum weights $W_{\max}$

3 Preliminaries

NOTATION. Let $[n, m] = \{n, \dots, m\}$ for integers $n \leq m$, and $[n] = [1, n]$ for $n \geq 1$. We denote the security parameter by λ , vectors by arrows (e.g., $\vec{v}, \vec{H}$), and formal variables by sans-serif letters (e.g., X, Y). We denote $y \leftarrow^s \mathcal{A}(x)$ as running a (probabilistic) algorithm $\mathcal{A}$ on input x with fresh randomness, and $y \in \mathcal{A}(x)$ as y being one of the possible outputs of running $\mathcal{A}$ on x .

BILINEAR GROUPS. A *bilinear group parameter generator* is a probabilistic algorithm GGen with input 1^λ and output $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \circ)$. Here, $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are groups of λ -bit prime order p , and $\circ : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a bilinear map satisfying (1) *bilinearity*, i.e., for any $A \in \mathbb{G}_1, B \in \mathbb{G}_2$ and $x, y \in \mathbb{Z}_p$, $xA \circ yB = (xy) \cdot A \circ B$, and (2) *non-triviality*, i.e., for any $G_1 \in \mathbb{G}_1^*, G_2 \in \mathbb{G}_2^*, G_1 \circ G_2 \in \mathbb{G}_T^*$. We will also use brackets notations $[\![\cdot]\!]_1, [\![\cdot]\!]_2, [\![\cdot]\!]_T$ to denote group elements in the group $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, respectively. For instance, let $a \in \mathbb{Z}_p$, $[\![a]\!]* = a[\![1]\!]*$ where $[\![1]\!]*$ is the generator of $\mathbb{G}_*$. Also, for any $a, b \in \mathbb{Z}_p$, we write $[\![a]\!]_1 \circ [\![b]\!]_2 = [\![ab]\!]_T$ to denote pairing operations.

GENERIC GROUP MODEL. Our security proofs rely on the Generic Group Model (GGM) extended to bilinear pairing groups. We use Shoup's model [27] following the formalization in [32], also used in [3]. We defer the formalization to Appendix B.

SHAMIR SECRET SHARING. For positive integers $T \leq N$, we use T -out-of- N Shamir secret sharing scheme over $\mathbb{Z}_p$. For any secret $x \in \mathbb{Z}_p$, we write $(x_1, \dots, x_N) \leftarrow^s \mathsf{Share}(N, T, x)$ for the following algorithm: Sample $r_1, \dots, r_{T-1} \leftarrow^s \mathbb{Z}_p$, for $i \in [N]$, compute $x_i = x + \sum_{j=1}^{T-1} i^j r_j$, and return $(x_1, \dots, x_N)$. For any subset S of $[N]$ such that $|S| \geq T$, there exists efficiently computable vector $\vec{\omega} \in \mathbb{Z}_p^N$ such that $\omega_i = 0$ for $i \notin S$ and $\sum_{i \in S} \omega_i x_i = x$.

NON-INTERACTIVE ZERO KNOWLEDGE. A non-interactive proof system Π for an NP-relation $\mathcal{R}$ is a tuple $\Pi = (\Pi.\mathsf{Setup}, \Pi.\mathsf{Prove}, \Pi.\mathsf{Ver})$ with syntax:

$\mathsf{pp} \leftarrow^s \Pi.\mathsf{Setup}(1^\lambda)$: Generate the public parameters pp , which will be implicit inputs to the following two algorithms

$\pi \leftarrow^s \Pi.\mathsf{Prove}(\mathsf{x}, \mathsf{w})$: Given $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$, compute a proof π .

$b \leftarrow \Pi.\mathsf{Ver}(\mathsf{x}, \pi)$: Given a statement x and proof π , output a bit $b \in \{0, 1\}$ on whether the proof is valid.

We require perfect completeness and simulation extractability of Π , for the latter we adopt the notion taken from [12] which was used in the context of threshold encryption as well. We note that their definition combines both zero-knowledge aspect (indistinguishability of honest and simulated proofs) and extractability (extracted witness is valid for the statement) into one definition instead of defining them separately. We defer the formal definition to Appendix B.

4 Batched and Weighted Threshold Encryption with Silent Setup

In this section, we give syntax (Section 4.1) and definition (Sections 4.2 and 4.3) for batched weighted threshold encryption schemes with silent setup, taking inspiration from [10,30].

4.1 Syntax

A batched and weighted threshold encryption scheme with silent setup SBTE with message space $\mathcal{M}$ is a tuple of algorithms

$$\text{SBTE} = (\text{Setup}, \text{KeyGen}, \text{Prep}, \text{Enc}, \text{PartDec}, \text{Dec})$$

with the following syntax.

- $\text{Setup}(1^\lambda, W_{\max}, B_{\max}, \overline{T}) \Rightarrow \text{pp}$: On input the security parameter and the maximum weight $W_{\max}$, maximum batch size $B_{\max}$, and (optionally) the threshold T , output public parameters pp , which fix the message space $\mathcal{M}$. The dashed argument T is for schemes that fix the threshold T at setup time. We note that in our constructions, the public parameters pp *will be omitted and assumed to be implicit inputs to all algorithms*.
- $\text{KeyGen}(\hat{w}) \Rightarrow (\text{pk}, \text{sk})$: Take as input the *maximum weight $\hat{w}$ for the party*, and output a party's public key pk and a secret key sk ; pk is the party's public material.
- $\text{Prep}((\text{pk}_\ell, w_\ell)_{\ell \in [L]}) \Rightarrow ((\text{ek}, \text{dk}), I \subseteq [L])$: *Deterministically* aggregate a committee of $L \leq W_{\max}$ parties' public material into an encryption key ek and an aggregation key dk . The subset $I \subseteq [L]$ indicates the subset of parties whose keys are actually aggregated into ek and dk . We also assume that ek is included in dk for simplicity.
- $\text{Enc}(\text{ek}, \mu, \overline{T}) \Rightarrow \text{ct}$: Encrypt a message μ under ek . The shaded dashed T is the threshold determined at encryption time for a flexible-threshold scheme; this is omitted if T is already fixed during the setup.
- $\text{PartDec}(\text{ek}, \text{sk}_\ell, (\text{ct}_k)_{k \in [B]}) \Rightarrow \text{pd}_\ell$: Party ℓ 's partial decryption of a whole batch; outputs a single aggregated partial decryption pd_ℓ .
- $\text{Dec}(\text{dk}, (\text{pd}_\ell)_{\ell \in S}, (\text{ct}_k)_{k \in [B]}) \Rightarrow (\mu_k)_{k \in [B]}$: Recover the batch of messages from the aggregation key and a quorum S of partial decryptions.

EFFICIENCY. We require the following efficiency properties:

Aggregated encryption key size. $|\text{ek}| = o(W_{\max} + B_{\max}) \cdot \text{poly}(\lambda)$, i.e., it is sublinear in the maximum weight and batch size.

Ciphertext size. $|\text{ct}| = \text{poly}(\lambda)$, i.e., it is independent of the maximum weight and batch size.

Decryption key share size. For any ℓ , $|\text{pd}_\ell| = \text{poly}(\lambda)$, i.e., it is independent of the maximum weight and batch size.

Running time. All algorithms run in time $\text{poly}(\lambda, W_{\max}, B_{\max})$.

We note that for our constructions both ciphertext and decryption key share sizes are independent of $W_{\max}$ and $B_{\max}$.

Remark 4.1 (Coverage of existing variants). One important aspect of our syntax and definition is that they cover several settings of how the primitive can be used. In particular, the syntax also covers the following settings:

- Fixing $W_{\max} = N$, $B_{\max} = 1$, and $\hat{w}_\ell = 1$ for all parties ℓ allows us to achieve an *unweighted silent threshold encryption scheme*.
- Fixing $W_{\max} = N$ and $\hat{w}_\ell = 1$ for all parties ℓ allows us to achieve an *unweighted silent batched threshold encryption scheme*.
- Fixing $B_{\max} = 1$ allows us to achieve a *weighted silent threshold encryption scheme*.

Remark 4.2 (Distinctions from prior works). We note that our syntax differs from these works due to (1) we do not fix a slot/index in the batch at ciphertext generation time (e.g., [10]), (2) we do not have a batch label tied to a ciphertext (e.g., [21]), and (3) we consider batched encryption settings not batched identity-based encryption (BIBE). For the latter remark, in BIBE, ciphertexts are tied to a pre-determined identity id that is an input to Enc algorithm, and the partial decryption keys only depend on the batch of identities $\{\text{id}_1, \dots, \text{id}_B\}$ and can be used to decrypt any ciphertexts encrypted to an identity in the batch.

Moreover, we simplify the syntax by not explicitly including key verification, ciphertext verification, and partial decryption verification algorithms. These procedures will be present in our constructions but only invoked when necessary.

4.2 Correctness

We say that an SBTE scheme is correct if for any $\lambda \in \mathbb{N}$, positive integers $W_{\max} = W_{\max}(\lambda)$, $B_{\max} = B_{\max}(\lambda)$, $T = T(\lambda)$, $L = L(\lambda)$, $B = B(\lambda) \leq B_{\max}$, $\mathbf{pp}$ in the support of $\text{Setup}(1^\lambda, W_{\max}, B_{\max}, \lceil \frac{T}{B} \rceil)$, any positive integer weights $(w_\ell)_{\ell \in [L]}$ and $(\hat{w}_\ell)_{\ell \in [L]}$ such that $w_\ell \leq \hat{w}_\ell \leq W_{\max}$ for all $\ell \in [L]$ and $T \leq \sum_{\ell \in [L]} w_\ell \leq W_{\max}$ and any $S \subseteq [L]$ such that $\sum_{\ell \in S} w_\ell \geq T$, any messages $\mu_1, \dots, \mu_B \in \mathcal{M}$, the following probability is 1.

$$\Pr \left[\begin{array}{l} I = [L] \wedge \\ (\mu_k)_{k \in [B]} = (\mu'_k)_{k \in [B]} \end{array} \middle| \begin{array}{l} \text{for } \ell \in [L] : (\text{pk}_\ell, \text{sk}_\ell) \leftarrow \text{KeyGen}(\hat{w}_\ell) \\ ((\text{ek}, \text{dk}), I) \leftarrow \text{Prep}((\text{pk}_\ell, w_\ell)_{\ell \in [L]}) \\ \text{for } k \in [B] : \text{ct}_k \leftarrow \text{Enc}(\text{ek}, \mu_k, \lceil \frac{T}{B} \rceil) \\ \text{for } \ell \in S : \text{pd}_\ell \leftarrow \text{PartDec}(\text{ek}, \text{sk}_\ell, (\text{ct}_k)_{k \in [B]}) \\ (\mu'_k)_{k \in [B]} \leftarrow \text{Dec}(\text{dk}, (\text{pd}_\ell)_{\ell \in S}, (\text{ct}_k)_{k \in [B]}) \end{array} \right].$$

Here, the probability is over the randomness of key generation and encryption.

4.3 Security

INDISTINGUISHABILITY. We consider the following two indistinguishability guarantees for SBTE schemes with the corresponding games described in Figure 2:

- Indistinguishability under chosen-plaintext attack (see T-IND-CPA game): where (1) we allow adaptive registration of semi-honest keys (via $\mathcal{O}_{\text{CUser}}$), without adaptive corruption of honestly registered parties (generated and registered via $\mathcal{O}_{\text{HUser}}$), and (2) the adversary has access to a semi-honest decryption oracle ($\mathcal{O}_{\text{PDec}}$). In $\mathcal{O}_{\text{PDec}}$, we allow the adversary to query the decryption oracle with *semi-honest* ciphertexts, i.e., given a ciphertext, the adversary should be able to explain them with an underlying message and randomness.
- Indistinguishability under chosen-ciphertext attack (see T-IND-CCA game): where (1) we allow the adversary to *adaptively register malicious keys* for corrupted parties (this is via $\mathcal{O}_{\text{CUser}}$) but without adaptive corruption of honestly registered parties (generated and registered via $\mathcal{O}_{\text{HUser}}$), and (2) the adversary also has access to partial decryption oracle ($\mathcal{O}_{\text{PDec}}$) where they can query using maliciously generated batch of ciphertext.

For both games, we consider the setting inspired by the low-threshold definition of [7]. In this case, the adversary is allowed to make decryption oracle queries *on the challenge ciphertext* given that the *combined weight* of the corrupted parties $\mathcal{C}$ and the decryption shares involving batches containing the challenge ciphertext is strictly less than the threshold (see Line 6). Finally, the adversary's goal is to determine whether the challenge ciphertext encrypts to μ_0 or μ_1 .

We give the formal pseudocode description of the two games in Figure 2 with distinctions for fixed- and flexible- threshold schemes. The corresponding advantage is defined as follows, and we say that an SBTE scheme satisfies IND-CPA (IND-CCA, respectively) security if for any PPT adversary $\mathcal{A}$ the corresponding advantage is negligible in λ .

$$\text{Adv}_{\text{SBTE}}^{\text{t-ind-cpa/t-ind-cca}}(\mathcal{A}, \lambda) := \left| \Pr \left[(\text{T-IND-CPA/T-IND-CCA})_{\text{SBTE},0}^{\mathcal{A}}(1^\lambda) = 1 \right] - \Pr \left[(\text{T-IND-CPA/T-IND-CCA})_{\text{SBTE},1}^{\mathcal{A}}(1^\lambda) = 1 \right] \right| .$$

Remark 4.3 (Distinctions from prior works' security). Our security definition allows the adversary to make partial decryption queries on challenge ciphertext, given that the total weights (including the corrupted parties) do not exceed the threshold. This is stronger than in prior works [30,10,21] in silent threshold encryption schemes, which only consider adversaries that *do not* query the partial decryption shares on challenge ciphertext (or in the case of silent IBEs, the challenge identity). Moreover, they only consider static corruptions with less flexibility for the adversary to select malicious keys. For instance the definition in [10], the malicious keys are determined at the beginning of the game and cannot be decided after seeing some partial decryptions from honest users. This is in contrast to ours, where the malicious public keys can be determined later on even after seeing partial decryptions.

ROBUSTNESS. We define robustness according to the game SBTE-ROB given in Figure 3. In this game, the adversary has the following capabilities:

Game $\overline{\text{T-IND-CPA}}_{\text{SBTE},b}^A(1^\lambda)$	Game $\overline{\text{T-IND-CCA}}_{\text{SBTE},b}^A(1^\lambda)$
<pre> 1 : $(W_{\max}, B_{\max}, \lceil T \rceil, \text{st}_A) \leftarrow \mathcal{A}(1^\lambda)$ 2 : $\text{pp} \leftarrow \text{Setup}(1^\lambda, W_{\max}, B_{\max}, \lceil T \rceil)$ 3 : $\text{chal} \leftarrow 0 ; N \leftarrow 0 ; \mathcal{H}, \mathcal{C} \leftarrow \emptyset$ 4 : Initialize mapping $\mathcal{Q}$ 5 : $b' \leftarrow \mathcal{A}^{\mathcal{O}_{\text{HUser}}, \mathcal{O}_{\text{CUser}}, \mathcal{O}_{\text{Chal}}, \mathcal{O}_{\text{PDec}}}(\text{st}_A, \text{pp})$ 6 : if $\sum_{\{\ell \in I^* : \text{ct}^* \in \mathcal{Q}[\ell] \vee \ell \in \mathcal{C}\}} w_\ell^* < T^*$ then 7 : return b' 8 : return 0 </pre>	<pre> Oracle $\mathcal{O}_{\text{HUser}}(\hat{w})$: 1 : $N \leftarrow N + 1 ; \mathcal{H} \leftarrow \mathcal{H} \cup \{N\} ; \hat{w}_N \leftarrow \hat{w}$ 2 : $(\text{pk}_N, \text{sk}_N) \leftarrow \text{KeyGen}(\hat{w}_N)$ 3 : return pk_N </pre>
	<pre> Oracle $\mathcal{O}_{\text{CUser}}(\hat{w}, \rho)$: 1 : $N \leftarrow N + 1 ; \mathcal{C} \leftarrow \mathcal{C} \cup \{N\} ; \hat{w}_N \leftarrow \hat{w}$ 2 : $(\text{pk}_N, \text{sk}_N) \leftarrow \text{KeyGen}(\hat{w}; \rho)$ 3 : return </pre>
	<pre> Oracle $\mathcal{O}_{\text{CUser}}(\text{pk}, \hat{w})$: 1 : if $\exists \ell \in \mathcal{H} : (\text{pk}, \hat{w}) = (\text{pk}_\ell, \hat{w}_\ell)$ then 2 : return $\perp$ // No duplicate honest keys 3 : $N \leftarrow N + 1 ; \mathcal{C} \leftarrow \mathcal{C} \cup \{N\}$ 4 : $(\text{pk}_N, \hat{w}_N) \leftarrow (\text{pk}, \hat{w})$ 5 : return </pre>
<pre> Oracle $\mathcal{O}_{\text{Chal}}(J, \mu_0, \mu_1, (w_i)_{i \in J}, \lceil T \rceil)$: 1 : parse $J = (\ell_1, \dots, \ell_L)$ 2 : if $\text{chal} = 1 \vee (\exists j \in [L] : \ell_j \notin [N])$ 3 : $\vee (\{\ell_1, \dots, \ell_L\} < L)$ then 4 : return $\perp$ // only established keys 5 : $((\text{ek}^*, \text{dk}^*), I' \subseteq [L]) \leftarrow \text{Prep}((\text{pk}_{\ell_j}, w_{\ell_j})_{j \in [L]})$ 6 : $I \leftarrow (\ell_j)_{j \in I'}$ 7 : if $(\text{ek}^*, \text{dk}^*) = \perp \vee \sum_{\ell \in I \cap \mathcal{C}} w_\ell \geq T$ 8 : $\vee \sum_{\ell \in I} w_\ell < T \vee \sum_{\ell \in I} w_\ell > W_{\max}$ then 9 : return $\perp$ 10 : $\text{ct}^* \leftarrow \text{Enc}(\text{ek}^*, \mu_b, \lceil T \rceil)$ 11 : $\text{chal} \leftarrow 1 ; I^* \leftarrow I ; \lceil T \rceil \leftarrow T$ 12 : for $\ell \in I^* : w_\ell^* \leftarrow w_\ell$ 13 : return ct^* </pre>	<pre> Oracle $\mathcal{O}_{\text{PDec}}(\text{ek}, \ell, (\text{ct}_i, \lceil \mu_i, \rho_i, \lceil T \rceil \rceil)_{i \in [B]})$: 1 : if $\ell \notin [N] \vee \ell \in \mathcal{C} \vee B > B_{\max}$ then 2 : return $\perp$ </pre> <pre> $S \leftarrow \{i \in [B] : \text{ct}_i \neq \text{ct}^*\}$ if $\exists i \in S : \text{ct}_i \neq \text{Enc}(\text{ek}, \mu_i, \lceil T \rceil; \rho_i)$ then return $\perp$ </pre> <pre> 3 : 4 : $\mathcal{Q}[\ell] \leftarrow \mathcal{Q}[\ell] \cup \{\text{ct}_i\}_{i \in [B]}$ 5 : $\text{pd}_\ell \leftarrow \text{PartDec}(\text{ek}, \text{sk}_\ell, (\text{ct}_i)_{i \in [B]})$ 6 : return pd_ℓ </pre>

Fig. 2. The IND-CPA and IND-CCA games for SBTE. The solid and dotted boxes correspond to the IND-CPA and IND-CCA games, respectively. The *unshaded and dashed* boxes correspond to the defined games for *fixed*-threshold schemes, where T is chosen at setup. (See Line 1 and 2 of the game) The *shaded and dashed* boxes correspond to the defined games for *flexible*-threshold schemes, where T is chosen at encryption time. The game keeps track of honest parties set $\mathcal{H}$ and corrupted parties set $\mathcal{C}$.

- (1) Corrupt a subset of parties $\mathcal{C}$ and inject malicious public keys for these parties.
- (2) Choose $U^* \subseteq [B]$ and messages $(\mu_k)_{k \in [B] \setminus U^*}$ to be honestly encrypted.
- (3) Inject possibly malicious ciphertexts $(\text{ct}_k)_{k \in U^*}$ into the batch.
- (4) Inject decryption key shares of corrupted parties to be used for decryption.

Note that we make an assumption that the total weight of the honest parties in the decryption protocol is at least T so that decryption is possible, and adversary cannot force a failure attack. The goal of the adversary is to make the correctness guarantee fail, i.e., (1) to make the honest keys in $[L] \setminus \mathcal{C}$ not being aggregated into ek , or (2) to make decryption procedure fail to recover the

Game $\text{SBTE-ROB}_{\text{SBTE}}^A(\lambda)$:
1 : $(W_{\max}, L, C, \lceil T \rceil, B_{\max}, (\hat{w}_\ell)_{\ell \in [L] \setminus C}, \text{st}_A) \leftarrow \mathcal{A}(1^\lambda)$
2 : $\mathcal{H} \leftarrow [L] \setminus C$
3 : $\text{pp} \leftarrow \text{Setup}(1^\lambda, W_{\max}, B_{\max}, \lceil T \rceil)$
4 : for $\ell \in \mathcal{H}$: $(\text{pk}_\ell, \text{sk}_\ell) \leftarrow \text{KeyGen}(\hat{w}_\ell)$
5 : // Possibly malicious public keys
6 : $(B, \lceil T \rceil, (w_\ell)_{\ell \in [L]}, (\text{pk}_\ell, \hat{w}_\ell)_{\ell \in C}, U^*, (\mu_k)_{k \in [B] \setminus U^*}, \text{st}_A) \leftarrow \mathcal{A}(\text{st}_A, (\text{pk}_\ell)_{\ell \in [L] \setminus C})$
7 : $((\text{ek}, \text{dk}), I \subseteq [L]) \leftarrow \text{Prep}((\text{pk}_\ell, w_\ell)_{\ell \in [L]})$
8 : // Abort if aggregation fails
9 : if $(\text{ek}, \text{dk}) = \perp \vee \exists \ell \in [L] : w_\ell > \hat{w}_\ell$ then return 0
10 : // $\mathcal{A}$ wins if aggregation is successful but honest keys are not aggregated
11 : if $\mathcal{H} \not\subseteq I$ then return 1
12 : for $k \in [B] \setminus U^*$: $\text{ct}_k \leftarrow \text{Enc}(\text{ek}, \mu_k, \lceil T \rceil)$
13 : $(S \subseteq I, (\text{pd}_\ell)_{\ell \in C}, (\text{ct}_k)_{k \in U^*}) \leftarrow \mathcal{A}(\text{st}_A, (\text{ct}_k)_{k \in [B] \setminus U^*})$
14 : for $\ell \in S \setminus C$: $\text{pd}_\ell \leftarrow \text{PartDec}(\text{ek}, \text{sk}_\ell, (\text{ct}_k)_{k \in [B]})$
15 : $(\mu'_k)_{k \in [B]} \leftarrow \text{Dec}(\text{dk}, (\text{pd}_\ell)_{\ell \in S}, (\text{ct}_k)_{k \in [B]})$
16 : // $\mathcal{A}$ only wins if decryption outputs different messages from honestly encrypted ones
17 : return $(T > \sum_{\ell \in C \cap S} w_\ell) \wedge (T \leq \sum_{\ell \in S \setminus C} w_\ell) \wedge (\exists k \in [B] \setminus U^* : \mu_k \neq \mu'_k)$

Fig. 3. Robustness game for SBTE. We denote the differences between fixed and flexible threshold schemes via the dashed-unshaded and dashed-shaded boxes, respectively.

original messages $(\mu_k)_{k \in [B] \setminus U^*}$. We say that a silent batched weighted threshold encryption scheme SBTE is robust if for *any adversary* $\mathcal{A}$, the probability $\text{Adv}_{\text{SBTE}}^{\text{rob}}(\mathcal{A}, \lambda) := \Pr[\text{SBTE-ROB}_{\text{SBTE}}^A(\lambda) = 1]$ is negligible in λ .

We note that our robustness definition captures both (a) Rogue-Key Attacks: the adversary injecting malformed public keys into the aggregated encryption key along with malformed decryption key shares during decryption procedure, and (b) Rogue-Ciphertext Attacks: the adversary injecting malformed ciphertexts into the to-be-decrypted batch of ciphertexts.

5 Fixed-Threshold Construction from Pairings

In this section, we give constructions of silent batched weighted-threshold encryption schemes where the threshold is fixed. We first start with the description of the scheme in Section 5.1 along with some extensions, and then discuss correctness and security in Section 5.2.

5.1 Construction of fSWBTX

We give the silent weighted threshold encryption construction with the threshold fixed at setup time, denoted $\text{fSWBTX}_{\text{cpa}}$ and fSWBTX (for IND-CPA and IND-CCA schemes, respectively), in Figures 4 and 5. In the figure, we highlight

Alg. Enc(ek, $\mu \in \mathbb{G}_T$):	Alg. PartDec(ek, sk = s, (ct _k) _{k ∈ [B]}):
1 : $r \leftarrow \mathbb{Z}_p$	1 : for $k \in [B]$: parse $\text{ct}_k = (\text{ct}_{k,1}, \text{ct}_{k,2}, \text{ct}_{k,3}, \pi_{\text{ct},k})$
2 : $\text{ct}' \leftarrow (\llbracket r \rrbracket_1, r \cdot \text{pk}_{\text{agg}},$	2 : for $k \in [B]$: $\mathbb{x}_k \leftarrow (\llbracket 1 \rrbracket_1, \text{ek}, \text{ct}_{k,1}, \text{ct}_{k,2}, \text{ct}_{k,3})$
3 : $\llbracket r c^{W_{\max}+1} t \rrbracket_T + \mu)$	3 : $U \leftarrow [B]$; $U \leftarrow \{k \in [B] : \Pi_{\text{ct}}.\text{Ver}(\mathbb{x}_k, \pi_{\text{ct},k}) = 1\}$
4 : $\mathbb{x}_{\text{ct}} \leftarrow (\llbracket 1 \rrbracket_1, \text{ek}, \text{ct}')$	4 : $\text{pd} \leftarrow \sum_{k \in U} s^k \cdot \text{ct}_{k,1} \quad // = \llbracket \sum_{k \in U} r_k s^k \rrbracket_1$
5 : $\text{w}_{\text{ct}} = r$	5 : return pd
6 : $\pi_{\text{ct}} \leftarrow \Pi_{\text{ct}}.\text{Prove}(\mathbb{x}_{\text{ct}}, \text{w}_{\text{ct}})$	Alg. PartVerify(ek, pk _ℓ , pd _ℓ , (ct _k) _{k ∈ [B]}): $// B \leq B_{\max}$
7 : return $\text{ct} = (\text{ct}', \pi_{\text{ct}})$	1 : for $k \in [B]$: $\mathbb{x}_k \leftarrow (\llbracket 1 \rrbracket_1, \text{ek}, \text{ct}_{k,1}, \text{ct}_{k,2}, \text{ct}_{k,3})$
	2 : $U \leftarrow [B]$; $U \leftarrow \{k \in [B] : \Pi_{\text{ct}}.\text{Ver}(\mathbb{x}_k, \pi_{\text{ct},k}) = 1\}$
	3 : return $\text{pd}_\ell \circ \llbracket 1 \rrbracket_2 = \sum_{k \in U} \text{ct}_{k,1} \circ \xi_{\ell,k,2}$
Alg. Dec(dk, {pd _ℓ } _{ℓ ∈ S} , (ct _k) _{k ∈ [B]}):	
1 : for $k \in [B]$: $\mathbb{x}_k \leftarrow (\llbracket 1 \rrbracket_1, \text{ek}, \text{ct}_{k,1}, \text{ct}_{k,2}, \text{ct}_{k,3})$	
2 : $U \leftarrow [B]$; $U \leftarrow \{k \in [B] : \Pi_{\text{ct}}.\text{Ver}(\mathbb{x}_k, \pi_{\text{ct},k}) = 1\}$	
3 : $S' \leftarrow \{\ell \in S : \text{PartVerify}(\text{ek}, \text{pk}_\ell, \text{pd}_\ell, (\text{ct}_k)_{k \in [B]}) = 1\}$	
4 : $W_{S'} \leftarrow \sum_{\ell \in S'} \Omega_\ell $	
5 : if $S' \not\subseteq I \vee W_{S'} < T$ then return $\perp$	
6 : $// \text{ Step 1: Recover } \llbracket r_k c^{W_{\max}+1} \alpha_{\ell[i],j[i]} \rrbracket_T$	
7 : for $k \in U, \ell \in S', i \in \Omega_\ell$:	
8 : $E_{k,i} \leftarrow \text{pd}_\ell \circ \delta_{i,-k} - \sum_{k' \in U \setminus \{k\}} \text{ct}_{k',1} \circ \delta_{i,k'-k} \quad // = \llbracket r_k c^{W_{\max}+1} \alpha_{\ell[i],j[i]} \rrbracket_T$	
9 : $// \text{ Step 2: Recover } \llbracket r_k c^{W_{\max}+1} (t_i + \alpha_{\ell[i],j[i]}) \rrbracket_T$	
10 : for $k \in U, \ell \in S', i \in \Omega_\ell$:	
11 : $A_{k,i} \leftarrow \text{ct}_{k,2} \circ \llbracket c^{W_{\max}+1-i} \rrbracket_2$	
12 : $B_{k,i} \leftarrow \text{ct}_{k,1} \circ \sum_{i' \in [W] \setminus \{i\}} \gamma_{W_{\max}+1-i+i',i'}$	
13 : $// A_{k,i} - B_{k,i} = \llbracket r_k c^{W_{\max}+1} (t_i + \alpha_{\ell[i],j[i]}) \rrbracket_T$	
14 : $// \text{ Step 3: Reconstruction over virtual shares}$	
15 : Let $\tilde{\omega}$ be reconstruction coeff. for $\bigcup_{\ell \in S'} \Omega_\ell \subseteq [W_{\max}]$	
16 : for $k \in U$: $\mu_k \leftarrow \text{ct}_{k,3} - \sum_{\ell \in S', i \in \Omega_\ell} \omega_i (A_{k,i} - B_{k,i} - E_{k,i})$	
17 : for $k \in [B] \setminus U$: $\mu_k \leftarrow \perp$	
18 : return $(\mu_k)_{k \in [B]}$	

Fig. 5. Encryption, partial decryption, and decryption algorithms for fSWBTX extended from Figure 4.

CONSTRUCTION OVERVIEW. We briefly discuss the design of our silent weighted batched threshold encryption with fixed threshold below.

Policy enforcement in the public parameters. The setup algorithm Setup samples $t, c \leftarrow \mathbb{Z}_p$, shares t via a T -out-of- $W_{\max}$ Shamir secret sharing $(t_j)_{j \in [W_{\max}]}$, and posts powers of c 's $\llbracket c^i \rrbracket_2$ for $i \in [2W_{\max}]$ along with embeddings of $c^i t_j$ but puncturing the $i = W_{\max} + 1$ term. Looking ahead, the t_j 's will be embedded into the aggregated encryption key such that per-user secrets $\alpha_{\ell,j}$'s

individually mask each of the t_j , and the decryption procedure will remove these terms and reconstruct t .

Per-party keys. For a party with maximum virtualized weight $\hat{w}$, its secret key consists of two important components: (1) the masks for virtualized shares $\alpha_1, \dots, \alpha_{\hat{w}} \in \mathbb{Z}_p$ (which can be dropped) and (2) secret key $s \in \mathbb{Z}_p^*$ for batch decryption. The public key contains powers of s , $\llbracket s^k \rrbracket_1, \llbracket s^k \rrbracket_2$ for $k \in [-B_{\max}, B_{\max}]$, and group elements that embed (1) $c^i \alpha_j$ for $i \neq W_{\max} + 1$ and (2) $\alpha_j s^k c^{W_{\max}+1}$ for $k \neq 0$. The elements with $c^i \alpha_j$ will be used to mask the t_j in the aggregated encryption key, while elements with $\alpha_j s^k c^{W_{\max}+1}$ will be used in batch decryption for each of these parties.

Aggregated key. Assuming that all the keys $(\text{pk}_\ell)_{\ell \in [L]}$ (each with weight $w_\ell \leq \hat{w}_\ell$ with total weight $W \leq W_{\max}$) to be aggregated has the right structure (i.e., passes VerKey check), the aggregator computes a group element $\text{pk}_{\text{agg}} = \llbracket \sum_{i \in [W]} c^i (t_i + \alpha_{\ell[i], j[i]}) \rrbracket_1$ where $\ell[i], j[i]$ are corresponding assignments of the virtualization defined in Figure 4. This aggregated key structure resembles the silent threshold mechanisms in [30] (see also Equation (3)). The aggregated decryption key dk will then consist of decryption hints which along with enough decryption key shares allows the decryptor to decrypt a corresponding batch of ciphertext.

Encryption. A ciphertext of a message $\mu \in \mathbb{G}_T$ with randomness $r \leftarrow \mathbb{S}$ is

$$\text{ct} = (\llbracket r \rrbracket_1, r \cdot \text{pk}_{\text{agg}}, \llbracket r c^{W_{\max}+1} t \rrbracket_T + \mu).$$

Online partial decryption. The partial decryption for party ℓ on batch $(\text{ct}_k = (\text{ct}_{k,1}, \text{ct}_{k,2}, \text{ct}_{k,3})_{k \in [B]})$ is $\text{pd}_\ell = \sum_{k \in [B]} s_\ell^k \cdot \text{ct}_{k,1} = \llbracket \sum_{k \in [B]} r_k s_\ell^k \rrbracket_1$, (as in Equation (10)) which can be computed independently of w_ℓ .

Batch decryption. Given the decryption key shares pd_ℓ , we can separate our threshold batch decryption procedure into three phases. The goal of the decryption is to compute the one-time pad $\llbracket r_k c^{W_{\max}+1} t \rrbracket_T$ for each $k \in [B]$.

- **Applying batched decryption à la BTX.** For each party ℓ in the quorum S , given $\text{pd}_\ell = \llbracket \sum_{k \in [B]} r_k s_\ell^k \rrbracket_1$ and using $\llbracket c^{W_{\max}+1} s_\ell^{k'} \alpha_{\ell,j} \rrbracket_2$ in dk , we can recover $\llbracket r_k c^{W_{\max}+1} \alpha_{\ell[i], j[i]} \rrbracket_T$ for $k \in [B], i \in \bigcup_{\ell \in S} \Omega_\ell$.
- **Applying silent threshold decryption.** For each ciphertext $\text{ct}_k = (\text{ct}_{k,1}, \text{ct}_{k,2}, \text{ct}_{k,3})$, given how pk_{agg} is structured and the given elements in dk , we apply the cross-term cancellations from prior work [30] to compute group elements $\llbracket r_k c^{W_{\max}+1} (t_i + \alpha_{\ell[i], j[i]}) \rrbracket_T$. From the first step, we can cancel out the masks $\llbracket r_k c^{W_{\max}+1} \alpha_{\ell[i], j[i]} \rrbracket_T$ to obtain $\llbracket r_k c^{W_{\max}+1} t_i \rrbracket_T$.
- **Threshold reconstruct.** Given at least T of the values $\llbracket r_k c^{W_{\max}+1} t_i \rrbracket_T$, we can use linear reconstruction of Shamir secret sharing to compute $\llbracket r_k c^{W_{\max}+1} t \rrbracket_T$ and decrypt the message for each k .

ADDING CCA SECURITY. To boost our fSWBTX scheme from IND-CPA secure with semi-honest ciphertext and keys to be IND-CCA secure, we need to ensure that both the *maliciously generated keys and ciphertexts are well-formed*. In this case, we rely on simulation-extractable NIZK proof systems to show knowledge of the secret key and underlying randomness for ciphertext generation, denoted

Π_{pk} and Π_{ct} respectively. In particular, these proof systems prove membership for languages induced by the following relations $\mathcal{R}_{\text{pk}}, \mathcal{R}_{\text{ct}}$:

$$\mathcal{R}_{\text{pk}} := \left\{ (\mathbb{x} = (\llbracket 1 \rrbracket_1, \text{pk}, \hat{w}), \mathbb{w} = (s, (\alpha_j)_{j \in [\hat{w}]}) \mid \begin{array}{l} \xi_{1,1} = \llbracket s \rrbracket_1 \wedge \\ \forall j \in [\hat{w}] : \varphi_{j,1} = \llbracket \alpha_j \rrbracket_1 \end{array} \right\}, \quad (14)$$

$$\mathcal{R}_{\text{ct}} := \left\{ (\mathbb{x} = (\llbracket 1 \rrbracket_1, \text{ek}, \text{ct}' = (\text{ct}_1, \text{ct}_2, \text{ct}_3)), \mathbb{w} = r) \mid \begin{array}{l} \text{ct}_1 = \llbracket r \rrbracket_1 \wedge \\ \text{ct}_2 = r \text{pk}_{\text{agg}} \end{array} \right\}. \quad (15)$$

In the Equation (14), pk is parsed as in **KeyGen** dropping the subscript ℓ . The NIZK Π_{pk} is used in the key generation (**KeyGen**) and preprocessing **Prep** algorithms; it is a simple proof of knowledge for the underlying secret keys $(s, \alpha_1, \dots, \alpha_{\hat{w}})$ of pk . The NIZK Π_{ct} is used in the encryption (**Enc**) algorithm and the decryption protocol (**PartDec**, **Dec**) to prove knowledge of the randomness r used in the ciphertext generation. These NIZKs are implemented via Schnorr proofs, and for completeness, we give the constructions in Appendix H.

EFFICIENCY. The following theorem summarizes the efficiencies of fSWBTX.

Theorem 5.1. *The silent batched weighted threshold encryption fSWBTX has the following efficiency metrics: Let $W_{\max}$ be the maximum weights, $B_{\max}$ be the maximum batch size, and $\hat{w}_\ell$ be the maximum weight for party ℓ .*

Public parameters. $1\mathbb{G}_{\mathsf{T}}$ element, $2W_{\max}^2 + W_{\max}$ $\mathbb{G}_1$ elements, and $2W_{\max}^2 + W_{\max}$ $\mathbb{G}_2$ elements.

Public key pk_ℓ . $(2B_{\max} + 2\hat{w}_\ell W_{\max})\mathbb{G}_1$ elements, $2B_{\max}(\hat{w}_\ell + 1) + 2\hat{w}_\ell W_{\max}\mathbb{G}_2$ elements, and $(2 + \hat{w}_\ell)\mathbb{Z}_p$ elements.

Aggregated encryption key ek . $1\mathbb{G}_1$ and $1\mathbb{G}_{\mathsf{T}}$ elements.

Ciphertext. $2\mathbb{G}_1$, $1\mathbb{G}_{\mathsf{T}}$, and $2\mathbb{Z}_p$ elements.

Partial decryption. $1\mathbb{G}_1$ element

Decryption runtime. Let S' be the set of parties whose decryption shares are included, $N' = |S'|$, W be total weight of the committee, w_ℓ be the weight for party $\ell \in S'$, and B be the batch size. Ignoring the verification of pd_ℓ , this requires the following operations:

- 1 FFT in $\mathbb{G}_1$ of size $2B$ and $W_{\max}$ multi-scalar multiplication (MSM) of size W can be preprocessed.
- $2B + 2$ MSMs of size W for each $\ell \in S'$, B MSMs each of size w_ℓ .
- B multi-pairing evaluation of size N'
- $4B$ pairing
- 1 FFT in $\mathbb{G}_2$, and 1 inverse FFT in $\mathbb{G}_{\mathsf{T}}$ each of size $2B$.

Proof. We now show the above theorem, considering the public parameters and key sizes.

Sizes. The sizes follow from inspection. The instantiation of the NIZKs do not have public parameters and $|\pi_{\text{pk}}|$ and $|\pi_{\text{ct}}|$ are $(2 + \hat{w}_\ell)\mathbb{Z}_p$ and $2\mathbb{Z}_p$ elements, respectively.

Decryption complexity. We ignore the complexity of verifying pd_ℓ , which costs N' multi-pairing evaluation of size B . As written in Figure 5, the computational

complexity of naively computing the decryption is $O(WB(W+B))$ group operations (scalar multiplication and pairing evaluation). However, we can make the following observation to reduce the computational complexity. First, consider for each $k \in [B]$, how $\mu_k = \text{ct}_{k,3} - \sum_{\ell \in S', i \in \Omega_\ell} \omega_i (A_{k,i} - B_{k,i} - E_{k,i})$ is computed. We can expand the sum as follows.

$$\begin{aligned}
& \sum_{\ell \in S', i \in \Omega_\ell} \omega_i (A_{k,i} - B_{k,i} - E_{k,i}) \\
&= \underbrace{\text{ct}_{k,2} \circ \sum_{\ell \in S', i \in \Omega_\ell} \omega_i \cdot \llbracket c^{W_{\max}+1-i} \rrbracket_2}_{\text{same for all } k} - \underbrace{\text{ct}_{k,1} \circ \sum_{\ell \in S', i \in \Omega_\ell} \omega_i \sum_{i' \in [W] \setminus \{i\}} \gamma_{W_{\max}+1-i+i', i'}}_{\text{same for all } k} \\
&\quad - \sum_{\ell \in S', i \in \Omega_\ell} \omega_i E_{k,i}.
\end{aligned}$$

Notice that the two terms denoted are the same for all $k \in [B]$, meaning we can only compute them once and use it in the computation of all μ_k . Note that these terms can be computed via two multi-scalar multiplication of sizes $W_{S'}$. This is because the elements $\sum_{i' \in [W] \setminus \{i\}} \gamma_{W_{\max}+1-i+i', i'}$ are independent of the reconstruction coefficients $\vec{\omega}$ and can be computed beforehand (e.g., already aggregated in the dk).

Finally, we consider the last summation in the equation above:

$$\begin{aligned}
\sum_{\ell \in S', i \in \Omega_\ell} \omega_i E_{k,i} &= \sum_{\ell \in S', i \in \Omega_\ell} \omega_i \text{pd}_\ell \circ \delta_{i,-k} - \sum_{\ell \in S', i \in \Omega_\ell} \omega_i \sum_{k' \in [B] \setminus \{k\}} \text{ct}_{k',1} \circ \delta_{i,k'-k} \\
&= \sum_{\ell \in S'} \sum_{i \in \Omega_\ell} \omega_i \text{pd}_\ell \circ \delta_{i,-k} - \sum_{k' \in [B] \setminus \{k\}} \sum_{\ell \in S', i \in \Omega_\ell} \omega_i \text{ct}_{k',1} \circ \delta_{i,k'-k} \\
&= \left(\sum_{\ell \in S'} \text{pd}_\ell \circ \sum_{i \in \Omega_\ell} \omega_i \delta_{i,-k} \right) - \sum_{k' \in [B] \setminus \{k\}} \text{ct}_{k',1} \circ \underbrace{\sum_{\ell \in S', i \in \Omega_\ell} \omega_i \delta_{i,k'-k}}_{=: H_{k'-k}}
\end{aligned}$$

We then observe that the above computation can be separated into two parts for the first and second terms. For the first sum, we need one multi-pairing evaluation of size $|S'|$ and $|S'|$ multi-scalar multiplication each of size $\hat{w}$ (over $k \in [B]$, this is $O(WB)$ computation), and (2) Fast-Fourier Transforms (in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$) of size B to compute the convolution $\sum_{k' \in [B] \setminus \{k\}} \text{ct}_{k',1} \circ H_{k'-k}$ for all $k \in [B]$ ($O(B \log B)$ computation). In particular, to see how FFT can help with (2), we can view this as trying to find the coefficients for the multiplication of the following two polynomials (with coefficients being group elements):

$$f(X) := \sum_{k' \in [B]} \text{ct}_{k',1} X^{k'}, \quad g(X) := X^{B-1} \sum_{d \in [-B+1, B-1]} H_d X^{-d}.$$

Here, $H_0 = \llbracket 0 \rrbracket_2$ and $k' - k$ only ranges from $-B+1$ to $B-1$. One can see that the coefficient of X^{B+k-1} for $k \in [B]$ in $f \circ g$ is

$$\sum_{k' \in [B]} \text{ct}_{k',1} \circ H_{k'-k},$$

which is exactly the terms we want to compute. Note that this is the same middle-product optimization considered in BTX [2]. Hence, we can compute this by one forward FFTs of size $2B$ in each of $\mathbb{G}_1$ and $\mathbb{G}_2$ (the former can be precomputed after knowing the batch), $2B$ pairings, and one inverse FFT of size $2B$ in $\mathbb{G}_T$. Importantly, each value H_d can be computed with 1 multi-scalar multiplication of size at most W .

Thus, the total computation of the scheme sums up to the theorem statement, assuming that $\sum_{i' \in [W] \setminus \{i\}} \gamma_{W_{\max}+1-i+i', i'}$ is precomputed (as it does not depend on the quorum S). Hence, the running time for decryption is $O(WB + B \log B)$ group operations. $\square$

EXTENSIONS AND MINOR OPTIMIZATIONS. We now discuss some extensions of the scheme. We also note that the (un)weighted and (un)batched variants can be obtained simply from substituting $W_{\max}, \hat{w}_\ell, B_{\max}$ as discussed in Remark 4.1 into Theorem 5.1. Their security follows via the theorems in Section 5.2.

Remark 5.2 (Flexible threshold scheme). We can achieve a flexible threshold variant of the fSWBTX via the ideas from [30]. At a high-level, we use the fixed threshold scheme with maximum weight $2W_{\max} - 1$ and fixed threshold $W_{\max}$. Then, public keys for dummy parties are included into the public parameters and are aggregated at encryption time according to the threshold set during encryption. We refer the reader to Appendix G for more technical detail, recapping the techniques in [30], and applying it to our fixed threshold scheme. In the same section, we also discuss further optimizations to compress the public parameters to $O(W_{\max} \log W_{\max})$ and the public keys to $O(B_{\max} \hat{w}_\ell + W_{\max} \log \hat{w}_\ell)$ per party.

Remark 5.3 (Bit-vector messages). We can extend the scheme to support general bit-vector messages $\mu \in \{0, 1\}^*$ by setting ct_3 as $\mu \oplus H(\llbracket rc^{W_{\max}+1} t \rrbracket_T)$. This is standard and done in prior works, e.g., [18, 1]. The security proofs in the next sections follow similarly assuming that H is a random oracle.

5.2 Correctness and Security

CORRECTNESS. The following theorem establishes that the scheme is correct. This follows from inspection, and we refer to the proof in Appendix C.

Theorem 5.4 (Correctness). *The scheme fSWBTX_{cpa} is perfectly correct. Also, if the proof systems Π_{pk} and Π_{ct} are perfectly complete, the scheme fSWBTX is perfectly correct.*

ROBUSTNESS. The following theorem establishes perfect robustness of fSWBTX scheme with the proof given in Appendix D. The proof follows similarly to correctness. In particular, these are due to the well-formed checks via VerKey in Prep (only well-formed pk_ℓ are aggregated) and PartVerify in Dec checks (only well-formed pd_ℓ are used during decryption).

Theorem 5.5 (Robustness). *If $\Pi_{\text{pk}}, \Pi_{\text{ct}}$ are perfectly complete, then the scheme fSWBTX is perfectly robust, i.e., for any adversary $\mathcal{A}$, $\text{Adv}_{\text{fSWBTX}}^{\text{rob}}(\mathcal{A}, \lambda) = 0$.*

INDISTINGUISHABILITY. The following theorems (Theorems 5.6 and 5.7) establish IND-CPA and IND-CCA security of the scheme. We note that Theorem 5.6 only considers the fSWBTX scheme *without* the additional NIZK proofs Π_{pk} and Π_{ct} .

Theorem 5.6 (IND-CPA Security). *Let GGen be a bilinear group parameter generator outputting groups of prime order $p = p(\lambda)$, fSWBTX_{cpa} be the fSWBTX scheme without the NIZKs. Let $\mathcal{A}$ be a generic adversary in T-IND-CPA game of fSWBTX_{cpa}, making at most $Q_{\mathbb{G}} = Q_{\mathbb{G}}(\lambda)$ queries to its generic group oracles and $Q_{\text{PDec}} = Q_{\text{PDec}}(\lambda)$ queries to the partial decryption oracle. Also, let $W_{\max} = W_{\max}(\lambda)$, $B_{\max} = B_{\max}(\lambda)$, $N = N(\lambda)$ be the maximum weight, batch size, and number of parties. Then,*

$$\text{Adv}_{\text{fSWBTX}_{\text{cpa}}}^{\text{t-ind-cpa}}(\mathcal{A}, \lambda) \leq \frac{Q^2(8W_{\max} + 4B_{\max} + 4NB_{\max})}{p - 1},$$

where $Q \leq Q_{\mathbb{G}} + B_{\max}N + 7 + 4W_{\max} + 4W_{\max}^2 + N(4B_{\max} + 4W_{\max}^2 + 4W_{\max}B_{\max})$.

Proof (Sketch). We now briefly give some intuition for our IND-CPA security proof and refer to Appendix E for a full formal proof. For simplicity, let us first ignore the partial decryption queries *on the challenge ciphertext* (we will reconcile on this later). The main goal is to show that the challenge the challenge ciphertext ct^* of the form

$$\text{ct}^* = ([r^*]_1, r^* \text{pk}_{\text{agg}}^*, [r^* c^{W_{\max}+1} t]_{\text{T}} + \mu_b),$$

for $r^* \leftarrow_{\$} \mathbb{Z}_p$, is indistinguishable from random. First, recall that the aggregated public key pk_{agg}^* derived from the adversary's choices in the query to $\mathcal{O}_{\text{Chal}}$ is of the form:

$$\text{pk}_{\text{agg}}^* := \sum_{i=1}^W [c^i \cdot (t_i + \alpha_{\ell[i], j[i]}^*)]_1.$$

In the above equation, the $\alpha_{\ell[i], j[i]}^*$ terms for $i \in [W]$ will consist of maliciously chosen values (coming from public keys registered via $\mathcal{O}_{\text{CUser}}$) and uniformly sampled values (coming from public keys registered via $\mathcal{O}_{\text{HUser}}$). Moreover, less than T of these are maliciously chosen, due to the limited corruption threshold.

Via standard techniques, our GGM security proof first transitions into the symbolic game where for the uniformly random scalars $r^*, c, t, \alpha_{\ell, j} \leftarrow_{\$} \mathbb{Z}_p$ (hidden from the adversary) are treated as formal variables $R, C, T, A_{\ell, j}$. Each group element will also be associated with an underlying polynomial in these variables. Then, the proof proceeds to argue that the monomial $RC^{W_{\max}+1}T$ cannot be reconstructed from group elements accessible to the adversary (excluding the $[r^* c^{W_{\max}+1} t]_{\text{T}} + \mu_b$ element). This is to argue that the element $[r^* c^{W_{\max}+1} t]_{\text{T}} + \mu_b$ looks random given all the other elements.

Consider the group element $r^* \text{pk}_{\text{agg}}^* = r^* \sum_{i=1}^W [c^i \cdot (t_i + \alpha_{\ell[i], j[i]}^*)]_1$ along with the given public parameters and public keys. Using these *publicly available group elements* as in Dec algorithm, one can recover

$$[r^* c^{W_{\max}+1} (t_i + \alpha_{\ell[i], j[i]}^*)]_{\text{T}}.$$

Hence, the proof remains to argue that the adversary cannot compute at least T of the elements $\llbracket r^* c^{W_{\max}+1} \alpha_{\ell[i],j[i]}^* \rrbracket_{\mathsf{T}}$ to recover $\llbracket r^* c^{W_{\max}+1} t \rrbracket_{\mathsf{T}}$. Here, unless the adversary knows $\alpha_{\ell[i],j[i]}$, observe that none of the elements available can be used to compute such term. This is due to none of the public group elements containing $\llbracket c^{W_{\max}+1} t_i \rrbracket_2$ or $\llbracket c^{W_{\max}+1} \alpha_{\ell,j} \rrbracket_2$.⁹ Therefore, since the adversary knows less than T of $\alpha_{\ell[i],j[i]}^*$, in the symbolic game, the monomial $\mathsf{RC}^{W_{\max}+1} \mathsf{T}$ is hard to compute.

Finally, including partial decryption queries on challenge ciphertext does not help the adversary much. In particular, recall that the partial decryption queries on honest user ℓ and a batch $\{\mathsf{ct}_k = (\mathsf{ct}_{k,1}, \mathsf{ct}_{k,2}, \mathsf{ct}_{k,3})\}_{k \in [B]}$ allows the adversary to obtain group element

$$\mathsf{pd}_\ell = \sum_{k \in [B]} s_\ell^k \mathsf{ct}_{k,1} .$$

These ciphertexts need to be honestly generated or correspond to the challenge ciphertext. If ct_k is the former, the term $s_\ell^k \mathsf{ct}_{k,1}$ can be computed by the adversary since it knows the randomness of ct_k and $\llbracket s_\ell^k \rrbracket_1$ is available. Hence, these partial decryption queries only reveal elements of the form $\llbracket r^* s_\ell^k \rrbracket_1$ for all $k \in [B_{\max}]$, which allows the adversary to obtain $\llbracket r^* c^{W_{\max}+1} \alpha_{\ell,j} \rrbracket_{\mathsf{T}}$ using the elements in pk_ℓ . Still, due to the winning condition in the game restricting sum of the corrupted party weights and the partial decryption to be less than T , these are still not enough for the adversary to recover the desired term. $\square$

Theorem 5.7 (IND-CCA Security). *Let GGen, p be as in Theorem 5.6, and $\Pi_{\mathsf{pk}}, \Pi_{\mathsf{ct}}$ be simulation-extractable NIZK proof systems for $\mathcal{R}_{\mathsf{pk}}, \mathcal{R}_{\mathsf{ct}}$, respectively. If $\mathsf{fSWBTX}_{\mathsf{cpa}}$ is T-IND-CPA secure, then fSWBTX is T-IND-CCA secure. In particular, for any adversary $\mathcal{A}$ running in time $T_{\mathcal{A}} = T_{\mathcal{A}}(\lambda)$, there exist adversaries $\mathcal{B}_{\mathsf{simext},1}, \mathcal{B}_{\mathsf{simext},2}, \mathcal{B}$ running in time $\approx T_{\mathcal{A}}$, such that*

$$\begin{aligned} \mathsf{Adv}_{\mathsf{fSWBTX}}^{\mathsf{t-ind-cca}}(\mathcal{A}, \lambda) &\leq 2 \cdot \mathsf{Adv}_{\Pi_{\mathsf{pk}}, \mathsf{Sim}_{\mathsf{pk}}}^{\mathsf{simext}}(\mathcal{B}_{\mathsf{simext},1}, \lambda) + 2 \cdot \mathsf{Adv}_{\Pi_{\mathsf{ct}}, \mathsf{Sim}_{\mathsf{ct}}}^{\mathsf{simext}}(\mathcal{B}_{\mathsf{simext},2}, \lambda) \\ &\quad + \mathsf{Adv}_{\mathsf{fSWBTX}_{\mathsf{cpa}}}^{\mathsf{t-ind-cpa}}(\mathcal{B}, \lambda) , \end{aligned}$$

where $\mathsf{Sim}_{\mathsf{pk}}$ and $\mathsf{Sim}_{\mathsf{ct}}$ are simulators for simulation-extractability properties of Π_{pk} and Π_{ct} , respectively.

Proof (Sketch). We refer to Appendix F for the full formal proof and only outline the intuition here. Simulation-extractability of Π_{pk} , allows us to simulate the proofs π_{pk} for honest parties' keys generated via $\mathcal{O}_{\mathsf{HUser}}$ and extract the underlying key generation randomness $(s, \alpha_1, \dots, \alpha_{\hat{w}})$ from $\mathcal{O}_{\mathsf{CUser}}$ queries. Simulation-extractability of Π_{ct} allows us to simulate the proof π_{ct} in the challenge ciphertext, and extract the encryption randomness from maliciously chosen ciphertexts in $\mathcal{O}_{\mathsf{PDec}}$ queries. Given these simulated proofs and extracted values, we can construct a straightforward reduction to the IND-CPA security of $\mathsf{fSWBTX}_{\mathsf{cpa}}$. $\square$

⁹ We note that the actual argument is much more involved, arguing that linear spans of several polynomials available cannot contain such term. However, it essentially boils down to arguing this fact.

References

1. Agarwal, A., Babel, K., Das, S., Gilakalaye, B.P., Mondal, A., Pinkas, B., Rindal, P., Yadav, A.: Weighted batched threshold encryption with applications to mempool privacy. In: 2026 IEEE Symposium on Security and Privacy (SP). pp. 2462–2481. IEEE (2026). <https://doi.org/10.1109/SP63933.2026.00175>
2. Agarwal, A., Das, S., Gilakalaye, B.P., Rindal, P., Shoup, V.: BTX: Simple and efficient batch threshold encryption. Cryptology ePrint Archive, Paper 2026/754 (2026), <https://eprint.iacr.org/2026/754>
3. Agarwal, A., Fernando, R., Pinkas, B.: Efficiently-thresholdizable batched identity based encryption, with applications. In: Kalai, Y.T., Kamara, S.F. (eds.) CRYPTO 2025, Part III. LNCS, vol. 16002, pp. 69–100. Springer, Cham (Aug 2025). https://doi.org/10.1007/978-3-032-01881-6_3
4. Applebaum, B., Beimel, A., Ishai, Y., Kushilevitz, E., Liu, T., Vaikuntanathan, V.: Succinct computational secret sharing. In: Saha, B., Servedio, R.A. (eds.) 55th ACM STOC. pp. 1553–1566. ACM Press (Jun 2023). <https://doi.org/10.1145/3564246.3585127>
5. Applebaum, B., Ishai, Y., Kushilevitz, E., Waters, B.: Encoding functions with constant online rate or how to compress garbled circuits keys. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 166–184. Springer, Berlin, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40084-1_10
6. Bebel, J., Ojha, D.: Ferveo: Threshold decryption for mempool privacy in BFT networks. Cryptology ePrint Archive, Paper 2022/898 (2022), <https://eprint.iacr.org/2022/898>
7. Boneh, D., Bünz, B., Nayak, K., Rotem, L., Shoup, V.: Context-dependent threshold decryption and its applications. In: Hanaoka, G., Yang, B.Y. (eds.) ASIACRYPT 2025, Part VI. LNCS, vol. 16250, pp. 506–538. Springer, Singapore (Dec 2025). https://doi.org/10.1007/978-981-95-5119-4_16
8. Boneh, D., Laufer, E., Tas, E.N.: Batch decryption without epochs and its application to encrypted mempools. Cryptology ePrint Archive, Report 2025/1254 (2025), <https://eprint.iacr.org/2025/1254>
9. Boneh, D., Nema, R., Roy, A., Tas, E.N.: Efficient batch threshold encryption using partial fraction techniques. In: Heninger, N., Rosulek, M. (eds.) Advances in Cryptology – CRYPTO 2026. pp. 36–67. Springer Nature Switzerland, Cham (2026). https://doi.org/10.1007/978-3-032-35374-0_2
10. Bormet, J., Choudhuri, A.R., Faust, S., Garg, S., Othman, H., Policharla, G.V., Qu, Z., Wang, M.: BEAST-MEV: Batched threshold encryption with silent setup for MEV prevention. Cryptology ePrint Archive, Report 2025/1419 (2025), <https://eprint.iacr.org/2025/1419>
11. Bormet, J., Faust, S., Othman, H., Qu, Z.: BEAT-MEV: Epochless approach to batched threshold encryption for MEV prevention. In: Bauer, L., Pellegrino, G. (eds.) USENIX Security 2025. pp. 3457–3476. USENIX Association (Aug 2025), <https://www.usenix.org/conference/usenixsecurity25/presentation/bormet>
12. Brzuska, C., Kloß, M., Woo, I.K.Y.: Threshold public-key encryption: Definitions, relations, and CPA-to-CCA transforms. In: Bai, S., Persichetti, E. (eds.) PKC 2026, Part IV. LNCS, vol. 16554, pp. 162–197. Springer, Cham (May 2026). https://doi.org/10.1007/978-3-032-26740-5_6

13. Choudhuri, A.R., Garg, S., Piet, J., Policharla, G.V.: Mempool privacy via batched threshold encryption: Attacks and defenses. In: Balzarotti, D., Xu, W. (eds.) *USENIX Security 2024*. USENIX Association (Aug 2024), <https://www.usenix.org/conference/usenixsecurity24/presentation/choudhuri>
14. Choudhuri, A.R., Garg, S., Policharla, G.V., Wang, M.: Practical mempool privacy via one-time setup batched threshold encryption. In: Bauer, L., Pellegrino, G. (eds.) *USENIX Security 2025*. pp. 3477–3495. USENIX Association (Aug 2025), <https://www.usenix.org/conference/usenixsecurity25/presentation/choudhuri>
15. Desmedt, Y.: Society and group oriented cryptography: A new concept. In: Pomerance, C. (ed.) *CRYPTO’87*. LNCS, vol. 293, pp. 120–127. Springer, Berlin, Heidelberg (Aug 1988). https://doi.org/10.1007/3-540-48184-2_8
16. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) *CRYPTO’89*. LNCS, vol. 435, pp. 307–315. Springer, New York (Aug 1990). https://doi.org/10.1007/0-387-34805-0_28
17. Dietz, M., Li, H., Lin, H.: TinyLabels: How to compress garbled circuit input labels, efficiently. In: Fehr, S., Fouque, P.A. (eds.) *EUROCRYPT 2025*, Part VI. LNCS, vol. 15606, pp. 245–274. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91095-1_9
18. Fernando, R., Policharla, G.V., Tonkikh, A., Xiang, Z.: TrX: Encrypted mempools in high performance BFT protocols. *Cryptology ePrint Archive*, Report 2025/2032 (2025), <https://eprint.iacr.org/2025/2032>
19. Garg, S., Kolonelos, D., Policharla, G.V., Wang, M.: Threshold encryption with silent setup. In: Reyzin, L., Stebila, D. (eds.) *CRYPTO 2024*, Part VII. LNCS, vol. 14926, pp. 352–386. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68394-7_12
20. Garimidi, P., Bonneau, J., Heimbach, L.: On the limits of encrypted mempools (Jul 2025), <https://a16zcrypto.com/posts/article/limits-encrypted-mempools/>, a16z Crypto
21. Gong, J., Waters, B., Wee, H., Wu, D.J.: Threshold batched identity-based encryption from pairings in the plain model. In: Daemen, J., Thomé, E. (eds.) *EUROCRYPT 2026*, Part V. LNCS, vol. 16545, pp. 548–578. Springer, Cham (May 2026). https://doi.org/10.1007/978-3-032-25330-9_19
22. Hall-Andersen, M., Simkin, M., Wagner, B.: Silent threshold encryption with one-shot adaptive security. *Cryptology ePrint Archive*, Report 2025/1384 (2025), <https://eprint.iacr.org/2025/1384>
23. Lührs, F., Bezzenberger, L., Mosterts, F., Faust, S., Erwig, A.: The road towards an encrypted mempool on ethereum (2025), https://docs.shutter.network/docs/shutter/research/the_road_towards_an_encrypted_mempool_on_ethereum, shutter Docs
24. Maurer, U.M.: Abstract models of computation in cryptography (invited paper). In: Smart, N.P. (ed.) *10th IMA International Conference on Cryptography and Coding*. LNCS, vol. 3796, pp. 1–12. Springer, Berlin, Heidelberg (Dec 2005). https://doi.org/10.1007/11586821_1
25. Policharla, G.V.: A simple batched threshold encryption scheme. *Cryptology ePrint Archive* (2026)
26. Reyzin, L., Smith, A., Yakubov, S.: Turning hate into love: Compact homomorphic ad hoc threshold encryption for scalable mpc. In: Dolev, S., Margalit, O., Pinkas, B., Schwarzmann, A. (eds.) *Cyber Security Cryptography and Machine Learning*. pp. 361–378. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-78086-9_27

27. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 256–266. Springer, Berlin, Heidelberg (May 1997). https://doi.org/10.1007/3-540-69053-0_18
28. Shutter Network, Tatu: The completion of our op grant: Defining an encrypted mempool for the op stack (Nov 2023), <https://blog.shutter.network/defining-an-encrypted-mempool-for-the-op-stack/>, shutter Blog
29. Suegami, S., Ashizawa, S., Shibano, K.: Constant-cost batched partial decryption in threshold encryption. Cryptology ePrint Archive, Paper 2024/762 (2024), <https://eprint.iacr.org/2024/762>
30. Waters, B., Wu, D.J.: Silent threshold cryptography from pairings: Expressive policies in the plain model. In: Daemen, J., Thomé, E. (eds.) EUROCRYPT 2026, Part V. LNCS, vol. 16545, pp. 517–547. Springer, Cham (May 2026). https://doi.org/10.1007/978-3-032-25330-9_18
31. Xiong, S., Zhang, Y., Chen, J.: Efficient batched IBE from lattices in the standard model. Cryptology ePrint Archive, Report 2025/2158 (2025), <https://eprint.iacr.org/2025/2158>
32. Zhandry, M.: To label, or not to label (in generic groups). In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part III. LNCS, vol. 13509, pp. 66–96. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15982-4_3

Supplementary Materials

A AI Policy Statement

All results, proofs, and constructions were developed by human authors. Claude Fable-5 and Opus-4.8 were used to help with typesetting and to verify the correctness of the constructions and proof ideas. GPT-6 Astra was used for manuscript-reviewing and for verifying prior works' efficiency claims made in Table 1.

B Deferred Preliminaries

GENERIC GROUP MODEL. We now recall the formalization of the generic group model. The model based on bilinear groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ with prime order p relies on random injections $\mathcal{L}_i : \mathbb{Z}_p \rightarrow S_i$ for $i \in \{1, 2, T\}$. Here, $S_i \subseteq \{0, 1\}^*$ are of cardinality at least p . An adversary $\mathcal{A}$ in this model has access to the following oracles:

- **Labelling oracle:** On input $(i \in \{1, 2, T\}, x \in \mathbb{Z}_p)$, return $\mathcal{L}_i(x)$.
- **Group operation oracle:** On inputs $(i \in \{1, 2, T\}, l_1 \in S_i, l_2 \in S_i, a_1 \in \mathbb{Z}_p, a_2 \in \mathbb{Z}_p)$, if there exists x_1, x_2 such that $\mathcal{L}_i(x_1) = l_1$ and $\mathcal{L}_i(x_2) = l_2$, return $\mathcal{L}_i(a_1 x_1 + a_2 x_2)$. Otherwise, return $\perp$.
- **Pairing oracle:** On input $(l_1 \in S_1, l_2 \in S_2)$, the oracle checks whether there exists $x_1, x_2 \in \mathbb{Z}_p$ such that $\mathcal{L}_1(x_1) = l_1$ and $\mathcal{L}_2(x_2) = l_2$. If not, it returns $\perp$. Otherwise, returns $\mathcal{L}_T(x_1 x_2)$.

NON-INTERACTIVE ZERO KNOWLEDGE. We recall the security definitions required for NIZK proof system Π below.

Definition B.1 ((Perfect) Completeness). *For any $\text{pp} \in \Pi.\text{Setup}(1^\lambda)$ and $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$,*

$$\Pr[\Pi.\text{Ver}(\mathbb{x}, \pi) = 1 \mid \pi \leftarrow \Pi.\text{Prove}(\mathbb{x}, \mathbb{w})] = 1.$$

Definition B.2 (Simulation Extractability). *There exists a simulator $\text{Sim} = (\text{Sim}.\text{Setup}, \text{Sim}.\text{Sim}, \text{Sim}.\text{Ext})$ such that for any PPT adversary $\mathcal{A}$, the following advantage is negligible*

$$\text{Adv}_{\Pi, \text{Sim}}^{\text{simext}}(\mathcal{A}, \lambda) := |\Pr[\mathcal{A}^{\mathcal{O}_{\text{Prove}, 0}, \mathcal{O}_{\text{Ver}, 0}}(\text{pp}_0) = 1] - \Pr[\mathcal{A}^{\mathcal{O}_{\text{Prove}, 1}, \mathcal{O}_{\text{Ver}, 1}}(\text{pp}_1) = 1]|$$

Here, $\text{pp}_0 \leftarrow \Pi.\text{Setup}(1^\lambda)$ and $(\text{pp}_1, \text{td}) \leftarrow \text{Sim}.\text{Setup}(1^\lambda)$. Also, the oracles keep track of a set $\mathcal{Q} \leftarrow \emptyset$. The oracles $\mathcal{O}_{\text{Prove}, b}, \mathcal{O}_{\text{Ver}, b}$ are defined as follows:

- $\mathcal{O}_{\text{Prove}, b}(\mathbb{x}, \mathbb{w})$: Check if $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$, if not return $\perp$. Otherwise,
- If $b = 0$, return $\pi \leftarrow \Pi.\text{Prove}(\mathbb{x}, \mathbb{w})$.
 - If $b = 1$, compute $\pi \leftarrow \text{Sim}.\text{Sim}(\text{td}, \mathbb{x})$, $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathbb{x}, \pi)\}$, and return π .
- $\mathcal{O}_{\text{Ver}, b}(\mathbb{x}, \pi)$: Do the following.
- If $b = 0$, return $\Pi.\text{Ver}(\mathbb{x}, \pi)$.
 - If $b = 1$, if $\Pi.\text{Ver}(\mathbb{x}, \pi) = 0$, return 0. Otherwise, if $(\mathbb{x}, \pi) \in \mathcal{Q}$, return 1. Otherwise, run $\mathbb{w} \leftarrow \text{Sim}.\text{Ext}(\text{td}, \mathbb{x}, \pi)$ and return whether $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$.

C Proof of Theorem 5.4

We will first show correctness of $\text{fSWBTX}_{\text{cpa}}$ and note that for fSWBTX , the correctness follows simply from completeness of the NIZK. To show correctness, we have to show the following: (1) honestly generated keys will always be aggregated, (2) honest partial decryption of a batch of honestly generated ciphertext will always pass the PartVerify algorithm, and (3) valid partial decryption will always result in correct decryption.

Let $W_{\max}, B_{\max}, T$ be the maximum weight, maximum batch size, and threshold, $(\hat{w}_\ell)_{\ell \in [L]}$ be the maximum weight per party for each party, $(w_\ell)_{\ell \in [L]}$ be the individual weights to be input to the key aggregation algorithm, and $W = \sum_{\ell \in [L]} w_\ell$. Also, let

$$\begin{aligned} \text{pp} = & (W_{\max}, B_{\max}, \llbracket c^{W_{\max}+1} t \rrbracket_T, (\llbracket c^i \rrbracket_1, \llbracket c^i \rrbracket_2)_{i \in [2W_{\max}]}, \\ & (\llbracket c^i t_j \rrbracket_1, \llbracket c^i t_j \rrbracket_2)_{i \in [2W_{\max}] \setminus \{W_{\max}+1\}, j \in [W_{\max}]}, \text{pp}_{\text{pk}}, \text{pp}_{\text{ct}}) \end{aligned}$$

be the public parameters, and $(\text{pk}_\ell, \text{sk}_\ell) \leftarrow \text{KeyGen}(\hat{w}_\ell)$ for $\ell \in [L]$.

The following lemma shows that for the IND-CPA secure version of the scheme, the honestly generated keys pass the VerKey algorithm, and vice versa, the keys that pass the VerKey algorithm has the same format as honestly generated ones.

Lemma C.1. *For any $\text{pp} \in \text{Setup}(1^\lambda, W_{\max}, B_{\max}, T)$, $\text{fSWBTX}_{\text{cpa}}$ public key pk ¹⁰ with maximum weight $\hat{w}$ is of the form*

$$\begin{aligned} \text{pk} = & \left((\llbracket s^k \rrbracket_1, \llbracket s^k \rrbracket_2)_{k \in [-B_{\max}, B_{\max}]} \right. \\ & (\llbracket \alpha_j \rrbracket_1, \llbracket \alpha_j \rrbracket_2, (\llbracket c^i \alpha_j \rrbracket_1, \llbracket c^i \alpha_j \rrbracket_2)_{i \in [2W_{\max}] \setminus \{W_{\max}+1\}}, \\ & \left. (\llbracket c^{W_{\max}+1} \alpha_j s^k \rrbracket_2)_{k \in [-B_{\max}, B_{\max}] \setminus \{0\}} \right)_{j \in [\hat{w}]} \end{aligned}$$

for some $s \in \mathbb{Z}_p^*$, $\alpha_1, \dots, \alpha_{\hat{w}} \in \mathbb{Z}_p$ and c is the underlying scalar in pp if and only if $\text{VerKey}(\text{pk}) = 1$.

Proof. We first show that if pk is of the given format, then it satisfies VerKey . Via inspection, we can see that the checks on pk in VerKey pass due to how pk is computed as $s \neq 0$ (Line 1), the elements $\xi_{k,1}, \xi_{k,2}$ in both groups embed the same scalars (Line 3), and the elements are structured as powers-of- s (Line 5). Moreover, we can see that the elements $\varphi_{j,1}, \varphi_{j,2}$ in the two groups embed the same scalars (Line 6 and Line 8), the elements $\psi_{i,j,1}, \psi_{i,j,2}$ embed the scalar $c^i \alpha_j$ (Line 9), and the elements $\chi_{j,k}$ embed scalar $c^{W_{\max}+1} \alpha_j s^k$ (Line 11).

On the other hand, suppose that $\text{VerKey}(\text{pk}) = 1$. Then, let $s = \text{dlog}(\xi_{1,1})$ and $\alpha_j = \text{dlog}(\varphi_{j,1})$. Now, we argue the following:

¹⁰ We dropped the subscript for simplicity, and they will be reintroduced back when talking about specific parties.

- $s \neq 0$ due to the check in Line 1.
- $\xi_{k,1} = \llbracket s^k \rrbracket_1, \xi_{k,2} = \llbracket s^k \rrbracket_2$ due to the checks in Line 3 and Line 5.
- $\varphi_{j,1} = \llbracket \alpha_j \rrbracket_1, \varphi_{j,2} = \llbracket \alpha_j \rrbracket_2$ due to the check in Line 6.
- $\psi_{i,j,1} = \llbracket c^i \alpha_j \rrbracket_1, \psi_{i,j,2} = \llbracket c^i \alpha_j \rrbracket_2$ due to the checks in Line 8 and Line 9.
- $\chi_{j,k} = \llbracket c^{W_{\max}+1} \alpha_j s^k \rrbracket_2$ due to the check in Line 11.

This concludes the proof. $\square$

With the above lemma, given honestly generated keys $(\mathbf{pk}_\ell, \mathbf{sk}_\ell)$ with underlying randomness $(s_\ell \neq 0, (\alpha_{\ell,j})_{j \in [\hat{w}_\ell]})$ for $\ell \in [L]$, we have that **Prep** outputs $((\mathbf{ek}, \mathbf{dk}), I)$ such that $I = [L]$ (since no keys failed the **VerKey** checks), and one can write

$$\begin{aligned} \mathbf{ek} &= \left(\mathbf{pk}_{\text{agg}} = \llbracket \sum_{i \in [W]} c^i (t_i + \alpha_{\ell[i],j[i]}) \rrbracket_1, \llbracket c^{W_{\max}+1} t \rrbracket_{\mathbf{T}} \right), \\ \mathbf{dk} &= \left(\mathbf{ek}, I = [L], (\mathbf{pk}_\ell, \Omega_\ell)_{\ell \in [L]}, (\llbracket c^i \rrbracket_2)_{i \in [2W_{\max}]}, \right. \\ &\quad \underbrace{(\llbracket c^i (t_{i'} + \alpha_{\ell[i'],j[i']}) \rrbracket_2)_{i \in [2W_{\max}] \setminus \{W_{\max}+1\}, i' \in [W]}}_{\gamma_{i,i'}}, \\ &\quad \left. \underbrace{(\llbracket c^{W_{\max}+1} \alpha_{\ell[i],j[i]} s_\ell^k \rrbracket_2)_{i \in [W], k \in [-B_{\max}, B_{\max}] \setminus \{0\}}}_{\delta_{i,k}} \right). \end{aligned}$$

Here, Ω_ℓ are partitioning of $[W]$ into disjoint intervals, and $\ell[i], j[i]$ are mappings defined as in the **Prep** algorithm.

HONEST PARTIAL DECRYPTIONS PASS PartVerify. Consider any batch of honestly generated ciphertexts $(\mathbf{ct}_k)_{k \in [B]}$ where $\mathbf{ct}_k$ is an encryption of $\mu_k \in \mathbb{G}_{\mathbf{T}}$. Then, we have that

$$\mathbf{ct}_k = (\llbracket r_k \rrbracket_1, r_k \mathbf{pk}_{\text{agg}}, \llbracket r_k c^{W_{\max}+1} t \rrbracket_{\mathbf{T}} + \mu_k, \pi_{\mathbf{ct},k}).$$

Thus, $U = [B]$ in the **PartDec** algorithm. Hence, for any $\ell \in [L]$, we have that

$$\mathbf{pd}_\ell = \sum_{k \in U} s_\ell^k \mathbf{ct}_{k,1} = \llbracket \sum_{k \in [B]} s_\ell^k r_k \rrbracket_1. \quad (16)$$

Therefore, the check in **PartVerify** Line 3 passes for any $\ell \in [L]$.

CORRECT DECRYPTION. Let $S \subseteq [L]$ be such that $\sum_{\ell \in S} w_\ell \geq T$. First, recall that $U = [B]$ by the above. Then, for each $\ell \in S, i \in \Omega_\ell, k \in [B]$, consider the

values $E_{k,i}$ in Line 8 of fSWBTX.Dec:

$$\begin{aligned}
E_{k,i} &= \text{pd}_\ell \circ \delta_{i,-k} - \sum_{k' \in U \setminus \{k\}} \text{ct}_{k',1} \circ \delta_{i,k'-k} \\
&= \llbracket \sum_{k' \in [B]} s_\ell^{k'} r_{k'} \rrbracket_1 \circ \llbracket c^{W_{\max}+1} \alpha_{\ell,j[i]} s_\ell^{-k} \rrbracket_2 - \sum_{k' \in [B] \setminus \{k\}} \llbracket r_{k'} \rrbracket_1 \circ \llbracket c^{W_{\max}+1} \alpha_{\ell,j[i]} s_\ell^{k'-k} \rrbracket_2 \\
&= \sum_{k' \in [B]} \llbracket s_\ell^{k'} r_{k'} \rrbracket_1 \circ \llbracket c^{W_{\max}+1} \alpha_{\ell,j[i]} s_\ell^{-k} \rrbracket_2 - \sum_{k' \in [B] \setminus \{k\}} \llbracket r_{k'} \rrbracket_1 \circ \llbracket c^{W_{\max}+1} \alpha_{\ell,j[i]} s_\ell^{k'-k} \rrbracket_2 \\
&= \llbracket r_k c^{W_{\max}+1} \alpha_{\ell,j[i]} \rrbracket_\tau \\
&\quad + \sum_{k' \in [B] \setminus \{k\}} \left(\llbracket s_\ell^{k'} r_{k'} \rrbracket_1 \circ \llbracket c^{W_{\max}+1} \alpha_{\ell,j[i]} s_\ell^{-k} \rrbracket_2 - \llbracket r_{k'} \rrbracket_1 \circ \llbracket c^{W_{\max}+1} \alpha_{\ell,j[i]} s_\ell^{k'-k} \rrbracket_2 \right) \\
&= \llbracket r_k c^{W_{\max}+1} \alpha_{\ell,j[i]} \rrbracket_\tau . \tag{17}
\end{aligned}$$

Next, for $\ell \in S, i \in \Omega_\ell, k \in [B]$ we consider the values $A_{k,i}$ and $B_{k,i}$ defined in Lines (11 and 12) of fSWBTX.Dec.

$$\begin{aligned}
A_{k,i} &= \text{ct}_{k,2} \circ \llbracket c^{W_{\max}+1-i} \rrbracket_2 = (r_k \cdot \text{pk}_{\text{agg}}) \circ \llbracket c^{W_{\max}+1-i} \rrbracket_2 \\
&= \sum_{i' \in [W]} \llbracket r_k c^{i'} (t_{i'} + \alpha_{\ell[i'],j[i']}) \rrbracket_1 \circ \llbracket c^{W_{\max}+1-i} \rrbracket_2 \\
&= \llbracket r_k c^{W_{\max}+1} (t_i + \alpha_{\ell[i],j[i]}) \rrbracket_\tau + \sum_{i' \in [W] \setminus \{i\}} \llbracket r_k c^{W_{\max}+1+i'-i} (t_{i'} + \alpha_{\ell[i'],j[i']}) \rrbracket_\tau
\end{aligned} \tag{18}$$

$$\begin{aligned}
B_{k,i} &= \text{ct}_{k,1} \circ \sum_{i' \in [W] \setminus \{i\}} \gamma_{W_{\max}+1-i+i',i'} \\
&= \llbracket r_k \rrbracket_1 \circ \sum_{i' \in [W] \setminus \{i\}} \llbracket c^{W_{\max}+1-i+i'} (t_{i'} + \alpha_{\ell[i'],j[i']}) \rrbracket_2 \\
&= \sum_{i' \in [W] \setminus \{i\}} \llbracket r_k c^{W_{\max}+1-i+i'} (t_{i'} + \alpha_{\ell[i'],j[i']}) \rrbracket_\tau .
\end{aligned} \tag{19}$$

Therefore,

$$A_{k,i} - B_{k,i} - E_{k,i} = \llbracket r_k c^{W_{\max}+1} (t_{i'} + \alpha_{\ell[i'],j[i']}) \rrbracket_\tau .$$

Hence, with $U = [B]$, $S' = S$ where $\sum_{\ell \in S} w_\ell \geq T$, and correctness of Shamir secret sharing, we have that $\sum_{i \in [W]} \omega_i (A_{k,i} - B_{k,i} - E_{k,i}) = \llbracket r_k c^{W_{\max}+1} t \rrbracket_\tau$ for each $k \in [B]$. Therefore, the decryption algorithm always output $(\mu_k)_{k \in [B]}$. $\square$

D Proof of Theorem 5.5

Let $W_{\max}, L, \mathcal{C}, T, B_{\max}$ be the parameters that $\mathcal{A}$ outputs at the start of the game. Also, let $(\hat{w}_\ell)_{\ell \in [L]}$ be the upper bounds on the weights for each party $\ell \in [L]$. We will prove robustness via the following steps: (1) Show that the

aggregated keys $(\mathbf{ek}, \mathbf{dk})$ are of a certain format, (2) Show that $\mathbf{pd}_\ell$ that passes the **PartVerify** check is of a certain format, and (3) Show that decryption work correctly given enough honest parties.

First, we recall that the public parameters are of the following format with c, t, t_i being defined as in **Setup**.

$$\mathbf{pp} = (W_{\max}, B_{\max}, \llbracket c^{W_{\max}+1} t \rrbracket_{\mathsf{T}}, (\llbracket c^i \rrbracket_1, \llbracket c^i \rrbracket_2)_{i \in [2W_{\max}]}, \\ (\llbracket c^i t_j \rrbracket_1, \llbracket c^i t_j \rrbracket_2)_{i \in [2W_{\max}] \setminus \{W_{\max}+1\}, j \in [W_{\max}]}, \mathbf{pp}_{\mathsf{pk}}, \mathbf{pp}_{\mathsf{ct}})$$

Also, let $\mathbf{ct}_k = (\mathbf{ct}_{k,1}, \mathbf{ct}_{k,2}, \mathbf{ct}_{k,3}, \pi_{\mathbf{ct},k})$ for $k \in [B]$ be the ciphertexts defined in the game where for $k \in U^*$, the ciphertexts $\mathbf{ct}_k$ are output by $\mathcal{A}$.

Now, we will show that the output $((\mathbf{ek}, \mathbf{dk}), I \subseteq [L])$ of **Prep** is such that if $(\mathbf{ek}, \mathbf{dk}) \neq \perp$, then we have the following: Let Ω_ℓ for $\ell \in I$ be partitioning of $[W]$ where $|\Omega_\ell| = w_\ell$ and $W = \sum_{\ell \in I} w_\ell$, and $\ell[i], j[i]$ be the corresponding mapping as defined within **Prep** algorithm. Then, for $\ell \in I$, there exists some $s_\ell \in \mathbb{Z}_p^*, \alpha_{\ell,j} \in \mathbb{Z}_p$ for $j \in [\hat{w}_\ell]$, such that

$$\mathbf{ek} = (\mathbf{pk}_{\mathsf{agg}} = \sum_{i \in [W]} \llbracket c^i (t_i + \alpha_{\ell[i], j[i]}) \rrbracket_1, \llbracket c^{W_{\max}+1} t \rrbracket_{\mathsf{T}}), \quad (20)$$

$$\mathbf{dk} = (\mathbf{ek}, I, (\mathbf{pk}_\ell, \Omega_\ell)_{\ell \in I}, (\llbracket c^i \rrbracket_2)_{i \in [2W_{\max}]}, \\ (\llbracket c^i (t_{i'} + \alpha_{\ell[i'], j[i']}) \rrbracket_2)_{i \in [2W_{\max}] \setminus \{W_{\max}+1\}, i' \in [W]}, \\ (\llbracket c^{W_{\max}+1} s_{\ell[i]}^k \alpha_{\ell[i], j[i]} \rrbracket_2)_{i \in [W], k \in [-B_{\max}, B_{\max}] \setminus \{0\}}). \quad (21)$$

To show this, recall from how **Prep** is defined that I contains indices to public keys that passes **VerKey** check and $T \leq W \leq W_{\max}$ (since $(\mathbf{ek}, \mathbf{dk}) \neq \perp$). Moreover, via Lemma C.1, for $\mathbf{pk}_\ell$ with $\ell \in I$, we have that there exists $s_\ell \in \mathbb{Z}_p^*, \alpha_{\ell,j} \in \mathbb{Z}_p$ such that

$$\mathbf{pk}_\ell = \left(\llbracket s_\ell^k \rrbracket_1, \llbracket s_\ell^k \rrbracket_2 \right)_{k \in [-B_{\max}, B_{\max}]} \\ \left((\llbracket \alpha_{\ell,j} \rrbracket_1, \llbracket \alpha_{\ell,j} \rrbracket_2, (\llbracket c^i \alpha_{\ell,j} \rrbracket_1, \llbracket c^i \alpha_{\ell,j} \rrbracket_2)_{i \in [2W_{\max}] \setminus \{W_{\max}+1\}}, \right. \\ \left. (\llbracket c^{W_{\max}+1} \alpha_{\ell,j} s_\ell^k \rrbracket_2)_{k \in [-B_{\max}, B_{\max}] \setminus \{0\}})_{j \in [\hat{w}_\ell]}, \pi_{\mathbf{pk}} \right). \quad (22)$$

Then, with how $\mathbf{ek}$ and $\mathbf{dk}$ are defined in **Prep**, we see that Equations (20) and (21) are satisfied.

Next, let $U \subseteq [B]$ be the subset of indices $k \in [B]$ such that $\Pi_{\mathbf{ct}, \mathbf{Ver}}(\mathbb{x}_k = (\llbracket 1 \rrbracket_1, \mathbf{ek}, \mathbf{ct}_{k,1}, \mathbf{ct}_{k,2}, \mathbf{ct}_{k,3}), \pi_{\mathbf{ct},k}) = 1$. We will show that for $\ell \in S \subseteq I$ such that $\mathbf{pd}_\ell$ passes **PartVerify**, we have that

$$\mathbf{pd}_\ell = \sum_{k \in U} s_\ell^k \mathbf{ct}_{k,1}. \quad (23)$$

This is easy to see because the subset U corresponds to the one defined in **PartVerify**. Moreover, the check in Line 3 along with format of $\mathbf{pk}_\ell$ for $\ell \in S \subseteq I$, we have that $\mathbf{pd}_\ell$ satisfies Equation (23).

Finally, we show that for $k \in [B] \setminus U^*$, the decryption algorithm outputs μ_k . Since for $k \in [B] \setminus U^*$, we know that for some $r_k \in \mathbb{Z}_p$

$$\text{ct}_k = (\llbracket r_k \rrbracket_1, r_k \cdot \text{pk}_{\text{agg}}, \llbracket r_k c^{W_{\max}+1} t \rrbracket_{\mathbb{T}} + \mu_k) .$$

Moreover, since the proof $\pi_{\text{ct},k}$ is honestly generated, by completeness of Π_{ct} , the verification always passes and $k \in U$. Therefore, $[B] \setminus U^* \subseteq U$. Also, the winning condition of $\mathcal{A}$ infers that the weights of uncorrupted parties is at least T , meaning the weights of parties in S' is at least T and the decryption algorithm does not abort.

We now focus on how μ_k are computed in the decryption, mainly considering the values of $E_{k,i}, A_{k,i}, B_{k,i}$ as in the correctness proof. In particular, since we observed the format of dk from Equation (21), the derivation in Equations (17) to (19) follows immediately (barring the change in the aggregated set of indices from $[L]$ to I). Therefore, for $k \in [B] \setminus U^*$

$$\begin{aligned} \mu'_k &= \text{ct}_{k,3} - \sum_{\ell \in S', i \in \Omega_\ell} \omega_i (A_{k,i} - B_{k,i} - E_{k,i}) \\ &= \llbracket r_k c^{W_{\max}+1} t \rrbracket_{\mathbb{T}} + \mu_k - \sum_{\ell \in S', i \in \Omega_\ell} \omega_i \llbracket r_k c^{W_{\max}+1} t_i \rrbracket_{\mathbb{T}} = \mu_k . \end{aligned}$$

This concludes the proof. $\square$

E IND-CPA Proof of fSWBTX

E.1 Proof of Theorem 5.6

In this section, we show IND-CPA security of the weighted silent batched Threshold Encryption construction in Section 5 in the GGM. The proof structure consists of the following steps: (1) showing that the IND-CPA security of the scheme is implied by an ad-hoc security game for the scheme where the decryption shares given is different, and (2) showing this ad-hoc security is implied in the GGM.

MODIFIED IND-CPA GAME FOR THE WEIGHTED SCHEME. We consider a IND-CPA game for SBTE casted for the construction in Figure 4 with the oracle $\mathcal{O}_{\text{PDec}}$ modified, which we will denote as $\text{T-IND-CPA}'_{\text{GGen}}^{\mathcal{A}}(\lambda)$ game. The oracle $\mathcal{O}_{\text{PDec}}$ is modified as follows.

- The oracle takes as input an index $\ell \in [N]$ indicating the decrypting party.
- If $\ell \in \mathcal{C}$ or $\text{ct}^* = \perp$ (i.e., challenge ciphertext undefined), return $\perp$.
- Otherwise, set $\mathcal{HS} \leftarrow \mathcal{HS} \cup \{\ell\}$ and return $(\llbracket r s_\ell^k \rrbracket_1)_{k \in [B_{\max}]}$ where r is the randomness used to compute the challenge ciphertext.

At the end of the game, if $\sum_{\ell \in (\mathcal{HS} \cup \mathcal{C}) \cap I^*} w_\ell^* \geq T$, the game returns 0. Note here that $\mathcal{HS}$ functions similarly as the mapping $\mathcal{Q}$ in that it is used to check how many honest parties have issued decryption share on the challenge ciphertext to the adversary.

We define the advantage of any adversary $\mathcal{A}$ as

$$\text{Adv}_{\text{GGen}}^{\text{t-ind-cpa}'}(\mathcal{A}, \lambda) := |\Pr[\text{T-IND-CPA}'_{\text{GGen},0}^{\mathcal{A}(\lambda)} = 1] - \Pr[\text{T-IND-CPA}'_{\text{GGen},1}^{\mathcal{A}(\lambda)} = 1]| ,$$

and say that the scheme is T-IND-CPA' secure if the advantage is negligible in λ .

We then conclude the proof with the following lemmata: one showing that T-IND-CPA security of $\text{fSWBTX}_{\text{cpa}}$ is implied by T-IND-CPA' (proved in Section E.2), and another showing that T-IND-CPA' security is satisfied in the GGM (proved in Section E.3).

Lemma E.1. *Let $\text{GGen}, p, W_{\max}, B_{\max}, N, \text{fSWBTX}_{\text{cpa}}$ be as in Theorem 5.6. For any adversary $\mathcal{A}$ in the T-IND-CPA $_{\text{fSWBTX}_{\text{cpa}}}$ game making $Q_{\text{PDec}} = Q_{\text{PDec}}(\lambda)$ queries to its partial decryption oracle and running in time $T_{\mathcal{A}} = T_{\mathcal{A}}(\lambda)$, there exists an adversary $\mathcal{B}$ making at most Q_{PDec} queries to its decryption oracle and running in time $T_{\mathcal{B}} \approx T_{\mathcal{A}}$, such that*

$$\text{Adv}_{\text{fSWBTX}_{\text{cpa}}}^{\text{t-ind-cpa}}(\mathcal{A}, \lambda) \leq \text{Adv}_{\text{GGen}}^{\text{t-ind-cpa}'}(\mathcal{B}, \lambda) .$$

Lemma E.2. *Let $\text{GGen}, p, W_{\max}, B_{\max}, N, \text{fSWBTX}_{\text{cpa}}$ be as in Theorem 5.6. For any generic adversary $\mathcal{A}$ making at most $Q_{\text{G}} = Q_{\text{G}}(\lambda)$ queries to its group operation,*

$$\text{Adv}_{\text{GGen}}^{\text{t-ind-cpa}'}(\mathcal{A}, \lambda) \leq \frac{Q^2(8W_{\max} + 4B_{\max} + 4NB_{\max})}{p-1} ,$$

where $Q = Q_{\text{G}} + B_{\max}N + 7 + 4W_{\max} + 4W_{\max}^2 + N(4B_{\max} + 4W_{\max}^2 + 4W_{\max}B_{\max})$.

E.2 Proof of Lemma E.1

Consider an adversary $\mathcal{A}$ in the T-IND-CPA $_{\text{fSWBTX}_{\text{cpa}}}$ game. We construct the following adversary $\mathcal{B}$:

Setup. The adversary $\mathcal{A}$ first outputs $(W_{\max}, B_{\max}, T)$, which $\mathcal{B}$ also forwards to its game. Then, $\mathcal{B}$ receiving pp forwards it to $\mathcal{A}$.

Key generation oracles. The reduction $\mathcal{B}$ answers these queries as follows: Note that it keeps track of the number of parties N and the honest and corrupted party sets $\mathcal{H}$ and $\mathcal{C}$. Also, let ℓ be the index of the current party being generated by the oracle call

- $\mathcal{O}_{\text{HUser}}(\hat{w}_{\ell})$: Make a query $\text{pk}_{\ell} \leftarrow_{\$} \mathcal{O}_{\text{HUser}}(\hat{w}_{\ell})$ and return the output to $\mathcal{A}$.
- $\mathcal{O}_{\text{CUser}}(\hat{w}_{\ell}, \rho_{\ell})$: Make a query $\mathcal{O}_{\text{CUser}}(\hat{w}_{\ell}, \rho_{\ell})$ and also register the key $(\text{pk}_{\ell}, \text{sk}_{\ell}) \leftarrow \text{KeyGen}(\hat{w}_{\ell}; \rho_{\ell})$.

Challenge oracle query. On a query $(J = (\ell_j)_{j \in [L]}, \mu_0, \mu_1, (w_{\ell})_{\ell \in J})$ to the challenge oracle, $\mathcal{B}$ calls its own $\text{ct}^* \leftarrow \mathcal{O}_{\text{Chal}}(J, \mu_0, \mu_1, (w_{\ell})_{\ell \in J})$ and returns ct^* to $\mathcal{A}$. Also, $\mathcal{B}$ keeps track of the encryption key ek^* (which is deterministic based on the given subset J and the public key materials).

Partial decryption queries. On a decryption query $(\text{ek}, \ell \in [N], (\text{ct}_k, \mu_k, \rho_k)_{k \in [B]})$, $\mathcal{B}$ does as follows: We note first that the randomness ρ_k is simply $r_k \in \mathbb{Z}_p$ by how the encryption algorithm is defined.

- If $\ell \in \mathcal{C}$, return $\perp$.
- Parse $\text{ct}_k = (\text{ct}_{k,1}, \text{ct}_{k,2}, \text{ct}_{k,3})$ for each $k \in [B]$.
- Compute the subset $S = \{k \in [B] : \text{ct}_k \neq \text{ct}^*\}$ and return $\perp$ if there exists $k \in S$ such that $\text{ct}_k \neq \text{Enc}(\text{ek}, \mu_k; \rho_k)$.
- If $S = [B]$, notice that $\text{ct}_{k,1} = \llbracket r_k \rrbracket_1$, so we can output the partial decryption key using the public key $\text{pk}_\ell = (\llbracket s_\ell^k \rrbracket_1, \llbracket s_\ell^k \rrbracket_2)_{k \in [-B_{\max}, B_{\max}]}$, with s_ℓ as a placeholder for the secret key of honest party ℓ , as

$$\text{pd}_\ell = \sum_{k=1}^B r_k \llbracket s_\ell^k \rrbracket_1.$$

- Otherwise, $\mathcal{B}$ makes a query to obtain $(\text{pd}_{\ell,k})_{k \in [B_{\max}]} \leftarrow \mathcal{O}_{\text{PDec}}(\ell)$ (where by definition $\text{pd}_{\ell,k} = s_\ell^k \cdot \text{ct}_1^*$) and outputs

$$\text{pd}_\ell = \sum_{k \in [B] \setminus S} \text{pd}_{\ell,k} + \sum_{k \in S} r_k \llbracket s_\ell^k \rrbracket_1.$$

Guess bit b' . The output b' of $\mathcal{A}$ is forwarded to $\mathcal{B}$'s game.

It is easy to see that the adversary $\mathcal{B}$ perfectly simulates the T-IND-CPA game for $\mathcal{A}$ in both cases $b = 0, 1$. Moreover, the total weight of corrupted parties and the partial decryption queries on the challenge ciphertext are identical between $\mathcal{A}$ and $\mathcal{B}$, since $\mathcal{B}$ only invokes its oracle when the challenge ciphertext is queried to $\mathcal{O}_{\text{PDec}}$. This also implies that $\mathcal{B}$ makes at most Q_{PDec} queries to its partial decryption oracles. Therefore, we have proved the lemma. $\square$

E.3 Proof of Lemma E.2

Let $\mathcal{A}$ be a generic adversary playing the T-IND-CPA'_{GGen,b} games for $b \in \{0, 1\}$. We assume that $\mathcal{A}$ makes at most $Q_{\mathbb{G}}$ oracle queries to the generic group oracles (both group operation and pairing operation) and Q_{PDec} to the $\mathcal{O}_{\text{PDec}}$ oracle. Let $W_{\max}, N, B_{\max}$ be the maximum weight, number of parties, and batch size chosen by $\mathcal{A}$.

To prove this lemma, we consider the following sequence of games: $\mathbf{G}_{0,b}, \mathbf{G}_{1,b}, \mathbf{G}_2$ for $b \in \{0, 1\}$. At a high-level, $\mathbf{G}_{0,b}$ is the T-IND-CPA' game in the GGM, $\mathbf{G}_{1,b}$ is the symbolic version of the prior game, and $\mathbf{G}_2$ is the game where the ct_3^* portion of the challenge ciphertext is now a random label independent of the challenge messages.

$\mathbf{G}_{0,b}$ (T-IND-CPA'_b game in the GGM). This is the T-IND-CPA'_{GGen,b} with the adversary $\mathcal{A}$ modeled as a generic adversary, i.e., its group operations are now modeled as oracle calls to the following oracles, and the group elements in the game are now labels corresponding to the underlying discrete logarithm values. We detail the notable changes as follows:

Generic Group Oracles: The game keeps three labelling maps $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_T$ mapping scalars in $\mathbb{Z}_p$ to the set of labels $S_1 = S_2 = S_T = \{0, 1\}^{p'}$, respectively, where $2^{p'} \geq p$. We denote $\mathcal{L}_i[\cdot]$ as a function mapping elements in $\mathbb{Z}_p$ to S_i for $i \in \{1, 2, T\}$. The mapping are maintained via the following *oracle*:

- *Labelling oracle:* On input $(i \in \{1, 2, T\}, x \in \mathbb{Z}_p)$, if $\mathcal{L}_i[x] \neq \perp$, return $\mathcal{L}_i[x]$. Otherwise, the oracle samples a label $l_i \leftarrow S_i$, sets $S_i \leftarrow S_i \setminus \{l_i\}$, $\mathcal{L}_i[x] = l_i$, and return l_i . We will write $\mathcal{L}_i[f]$ and $\mathcal{L}_i^{-1}[l]$ as a short-hand for calling the labelling oracle and inverting the label l to the representation of the corresponding element.

Moreover, at any point during the game, the adversary gets access to the following oracles:

- *Group operation oracle:* On input $(i \in \{1, 2, T\}, l_1 \in \{0, 1\}^{p'}, l_2 \in \{0, 1\}^{p'}, a_1 \in \mathbb{Z}_p, a_2 \in \mathbb{Z}_p)$, the oracle checks whether there exists $f_1, f_2 \in \mathbb{Z}_p$ such that $\mathcal{L}_i[f_1] = l_1$ and $\mathcal{L}_i[f_2] = l_2$. If not, it returns $\perp$. Otherwise, it computes $g = a_1 f_1 + a_2 f_2$, executes the internal labelling for $\mathcal{L}_i[g]$, and returns $\mathcal{L}_i[g]$.
- *Pairing oracle:* On input $(l_1 \in \{0, 1\}^{p'}, l_2 \in \{0, 1\}^{p'})$, the oracle checks whether there exists $f_1, f_2 \in \mathbb{Z}_p$ such that $\mathcal{L}_1(f_1) = l_1$ and $\mathcal{L}_2(f_2) = l_2$. If not, it returns $\perp$. Otherwise, it computes $g = f_1 f_2$, executes the internal labelling for $\mathcal{L}_T[g]$, and returns $\mathcal{L}_T[g]$.

Public parameters: For uniformly random $t, c \leftarrow \mathbb{Z}_p$ and T -out-of- $W_{\max}$ Shamir secret shares $(t_i)_{i \in [W_{\max}]}$ of t , the public parameters contain the labels for the following scalars:

- In both $\mathbb{G}_1$ and $\mathbb{G}_2$, $\mathbf{pp}$ contain labels for $1, (c^i)_{i \in [2W_{\max}]}, (c^i t_j)_{i \in [2W_{\max}], \{W_{\max}+1\}, j \in [W_{\max}]}$
- In $\mathbb{G}_T$, $\mathbf{pp}$ contain a label for $c^{W_{\max}+1}t$.

Key Generation queries: For both $\mathcal{O}_{\text{HUser}}, \mathcal{O}_{\text{CUser}}$ queries, the adversary would learn labels for the corresponding public keys, but note that for the corrupted users, the adversary can generate / compute these labels themselves. For each party $\ell \in [N]$, the public keys contain labels for the following scalars: Let s_ℓ and $\alpha_{\ell,j}$ be the randomness used for sampling the key (in $\mathcal{O}_{\text{HUser}}$) or the specified key generation randomness by the adversary (in $\mathcal{O}_{\text{CUser}}$).

- In $\mathbb{G}_1$ and $\mathbb{G}_2$, $\mathbf{pk}_\ell$ contains labels for scalars $(s_\ell^k)_{k \in [-B_{\max}, B_{\max}]}, \alpha_{\ell,j}$ and $c^i \alpha_{\ell,j}$ for $i \in [2W_{\max}] \setminus \{W_{\max}+1\}$ and $j \in [\hat{w}_\ell]$.
- In $\mathbb{G}_2$, $\mathbf{pk}_\ell$ additionally contains labels for scalars $c^{W_{\max}+1} \alpha_{\ell,j} s_\ell^k$ for $k \in [-B_{\max}, B_{\max}] \setminus \{0\}$ and $j \in [\hat{w}_\ell]$.

Challenge ciphertext: On the query $(J \subseteq [N], \mu_0, \mu_1, (w_\ell)_{\ell \in J})$, let I^* be the subset of valid public keys that are aggregated into the encryption key $\mathbf{ek}^*$ (as defined in the game) and W be the total weight of parties in I^* . Recall also that the algorithm Prep computes the mapping $\ell[i], j[i]$ for $i \in [W]$ mapping an index to corresponding slot $(\ell \in I^*, j \in [w_\ell])$ of the virtualized keys. The game returns the challenge ciphertext which contains labels embedding the following scalars: r, rz in $\mathbb{G}_1$ with $z = \sum_{i \in [W]} c^i (t_i + \alpha_{\ell[i], j[i]})$, i.e., the underlying discrete logarithm of $\mathbf{pk}_{\text{agg}}$, and $rc^{W_{\max}+1}t + \mathcal{L}_T^{-1}(\mu_b)$.

Decryption query: For each party $\ell \in [N]$ that the adversary issued a decryption query, the output of the oracle contains labels that embed the scalars $rs_\ell^1, \dots, rs_\ell^{B_{\max}}$ in $\mathbb{G}_1$.

G_{1,b} (Symbolic game, encrypting to μ_b). We will use capital letter in sans-serif fonts to denote formal variable in the symbolic game. The symbolic T-IND-CPA' game is defined as follows:

Variables: We first describe the formal variables used in this game, which are

$$(\mathbf{C}, (\mathbf{X}_j)_{j \in [T]}, (\mathbf{A}_{\ell,j})_{\ell \in [N] \setminus \mathcal{C}, j \in [\hat{w}_\ell]}, (\mathbf{S}_\ell)_{\ell \in [N] \setminus \mathcal{C}}, \mathbf{R}) .$$

- The generators $\llbracket 1 \rrbracket_1, \llbracket 1 \rrbracket_2, \llbracket 1 \rrbracket_\tau$ correspond to the constant 1 in the corresponding groups.
- Random scalar c corresponds to $\mathbf{C}$.
- Random scalar t corresponds to $\mathbf{X}_1$, while $\mathbf{X}_j$ for $j = 2, \dots, T$ corresponds to the randomness for generating the virtual shares t_j ; in particular, we write $\mathbf{T}_j(\vec{X}) = \sum_{i=1}^T j^{i-1} \mathbf{X}_i$ for the polynomial underlying the virtual share t_j , $j \in [W_{\max}]$.
- The linear mask $\alpha_{\ell,j}$ that party $\ell \in [N] \setminus \mathcal{C}$ places on its j -th virtual share, $j \in [\hat{w}_\ell]$, corresponds to the variable $\mathbf{A}_{\ell,j}$.
- The per-node invertible secret $s_\ell \in \mathbb{Z}_p^*$ corresponds to $\mathbf{S}_\ell$, for $\ell \in [N]$.
- Randomness r in the challenge ciphertext corresponds to $\mathbf{R}$.

From this point on, we denote $\mathbb{Z}_p[*]$ as the ring of Laurent polynomials in the variables denoted above.¹¹

Generic group oracles: The game keeps three labelling maps $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_\tau$ mapping polynomials in $\mathbb{Z}_p[*]$ the variables above to the set of labels $S_1 = S_2 = S_T = \{0, 1\}^{p'}$, respectively, where $2^{p'} \geq p$. The mapping are maintained via the following *oracle*:

- *Labelling oracle:* On input $(i \in \{1, 2, T\}, f \in \mathbb{Z}_p[*])$, note first that the adversary can only query with $f \in \mathbb{Z}_p$ and the extension to $\mathbb{Z}_p[*]$ is to accommodate the labelling on polynomials made by the game. if $\mathcal{L}_i[f] \neq \perp$, return $\mathcal{L}_i[f]$. Otherwise, the oracle samples a label $l_i \leftarrow S_i$, sets $S_i \leftarrow S_i \setminus \{l_i\}$, $\mathcal{L}_i[f] = l_i$, and return l_i . We will write $\mathcal{L}_i[f]$ and $\mathcal{L}_i^{-1}[l]$ as a short-hand for calling the labelling oracle and inverting the label l to the symbolic representation of the corresponding element.

At any point during the game, the adversary gets access to the following oracles:

- *Group operation oracle:* On input $(i \in \{1, 2, T\}, l_1 \in \{0, 1\}^{p'}, l_2 \in \{0, 1\}^{p'}, a_1 \in \mathbb{Z}_p, a_2 \in \mathbb{Z}_p)$, the oracle checks whether there exists $f_1, f_2 \in \mathbb{Z}_p[*]$ such that $\mathcal{L}_i[f_1] = l_1$ and $\mathcal{L}_i[f_2] = l_2$. If not, it returns $\perp$. Otherwise, it computes $g = a_1 f_1 + a_2 f_2$, executes the internal labelling for $\mathcal{L}_i[g]$ (i.e., calling the labelling oracle with $g \in \mathbb{Z}_p[*]$), and returns $\mathcal{L}_i[g]$.
- *Pairing oracle:* On input $(l_1 \in \{0, 1\}^{p'}, l_2 \in \{0, 1\}^{p'})$, the oracle checks whether there exists $f_1, f_2 \in \mathbb{Z}_p[*]$ such that $\mathcal{L}_1(f_1) = l_1$ and $\mathcal{L}_2(f_2) = l_2$. If not, it returns $\perp$. Otherwise, it computes $g = f_1 f_2$, executes the internal labelling for $\mathcal{L}_\tau[g]$ (i.e., calling the labelling oracle with $g \in \mathbb{Z}_p[*]$), and returns $\mathcal{L}_\tau[g]$.

¹¹ We consider Laurent polynomials because the group elements will embed some negative degree terms S^{-k} for some integer $k \geq 1$.

Public parameters: The public parameters include labels which embed the following monomials / polynomials:

- C^i for $i \in [2W_{\max}]$ in both $\mathbb{G}_1, \mathbb{G}_2$.
- $C^i T_j(\vec{X})$ for $i \in [2W_{\max}] \setminus \{W_{\max} + 1\}, j \in [W_{\max}]$ in both $\mathbb{G}_1, \mathbb{G}_2$.
- $C^{W_{\max}+1} X_1$ in $\mathbb{G}_T$.

Key generation queries: We will ignore the labels for the keys for the corrupted parties since they are linear combination of elements in public parameters. For each party $\ell \in [N] \setminus \mathcal{C}$, the public key $\mathbf{pk}_\ell$ contains group elements that embed the following monomials:

- S_ℓ^k for $k \in [-B_{\max}, B_{\max}]$ ($k = 0$ is the constant term 1) in both groups.
- $A_{\ell,j}$ for $j \in [\hat{w}_\ell]$ in both groups.
- $C^i A_{\ell,j}$ for $i \in [2W_{\max}] \setminus \{W_{\max} + 1\}$ and $j \in [\hat{w}_\ell]$ in both groups.
- $C^{W_{\max}+1} A_{\ell,j} S_\ell^k$ for $j \in [\hat{w}_\ell]$ and $k \in [-B_{\max}, B_{\max}] \setminus \{0\}$ in $\mathbb{G}_2$.

Challenge ciphertext: For the challenge oracle query on $J = (\ell_1, \dots, \ell_L) \in [N]^L$ with weights $(w_\ell)_{\ell \in J}$, this game does the same operations as in the previous game to compute the vector I^* of aggregated party indices, total weight $W = \sum_{\ell \in I^*} w_\ell$, and the partitioning Ω_ℓ for $\ell \in I^*$. Also, recall again that the mappings $\ell[i], j[i]$ are implicitly defined within the call to algorithm **Prep**. Then, the challenge ciphertext includes labels in $\mathbb{G}_1$ embedding polynomials R, RZ , and the element $RC^{W_{\max}+1} X_1 + \mathcal{L}_T^{-1}(\mu_b)$ in $\mathbb{G}_T$, where

$$Z = \sum_{\ell \in I^* \setminus \mathcal{C}, i \in \Omega_\ell} C^i(T_i(\vec{X}) + A_{\ell,j[i]}) + \sum_{\ell \in I^* \cap \mathcal{C}, i \in \Omega_\ell} C^i(T_i(\vec{X}) + \alpha_{\ell,j[i]})$$

Partial decryption shares: Let $\mathcal{HS}$ be the set of parties that the adversary requested partial decryption shares. Note that $\sum_{\ell \in I^* \cap (\mathcal{HS} \cup \mathcal{C})} w_\ell < T$. The partial decryption shares of party $\ell \in \mathcal{HS}$ are labels for group elements in $\mathbb{G}_1$ that embed the monomials RS_ℓ^k for $k \in [B_{\max}]$.

The following lemma, proved in Section E.4, upper bounds the distinguishing advantage of $\mathcal{A}$ between $\mathbf{G}_{0,b}$ and $\mathbf{G}_{1,b}$.

Lemma E.3. $|\Pr[\mathbf{G}_{0,b}^A(\lambda) = 1] - \Pr[\mathbf{G}_{1,b}^A(\lambda) = 1]| \leq \frac{Q^2 \cdot (4W_{\max} + 2B_{\max} + 2NB_{\max})}{p-1}$
with $Q = Q_G + B_{\max}N + 7 + 4W_{\max} + 4W_{\max}^2 + N(4B_{\max} + 4W_{\max}^2 + 4W_{\max}B_{\max})$.

$\mathbf{G}_2$ (Sending random challenge ciphertext). In this game, we change how the challenge ciphertext is computed. In particular, instead of the last component $\mathbf{ct}_3^*$ being computed as a labelling of $RC^{W_{\max}+1} X_1 + \mathcal{L}_T^{-1}(\mu_b)$, we output $\mathbf{ct}_3^*$ as a label of Y^* where Y^* is formal variable independent of the ones introduced in the prior game.

The following lemma, proved in Section E.5, shows that the view of $\mathcal{A}$ does not change between the $\mathbf{G}_2$ and $\mathbf{G}_{1,b}$.

Lemma E.4. $\Pr[\mathbf{G}_{1,b}^A(\lambda) = 1] = \Pr[\mathbf{G}_2^A(\lambda) = 1]$.

Combining all the lemmata and summing over the advantage changes for both $b \in \{0, 1\}$, we have shown Lemma E.2. $\square$

E.4 Proof of Lemma E.3

To show this, we consider the following sequence of game transition from $\mathbf{G}_{0,b}$ to $\mathbf{G}_{1,b}$:

$\mathbf{G}'_0$: This is exactly the game $\mathbf{G}_{0,b}$.

$\mathbf{G}'_1$: In this game, the game additionally keeps track of the underlying polynomial representation of each label along with the corresponding discrete logarithm, which are evaluation of these multivariate polynomials. The polynomial representation of group elements is defined as in $\mathbf{G}_{1,b}$, and if there are multiple polynomials with the same evaluation, we stick to the label of the first polynomial registered. In particular, we make the following changes to the generic group oracles: The game keeps the same labelling maps $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_T$ mapping **polynomials in $\mathbb{Z}_p[*]$** to the set of labels $S_1 = S_2 = S_T = \{0, 1\}^{p'}$, respectively, where $2^{p'} \geq p$. We denote $\mathcal{L}_i[\cdot]$ as a function mapping **polynomials in $\mathbb{Z}_p[*]$** to S_i for $i \in \{1, 2, T\}$. The oracles are now as follows:

- *Labelling oracle*: On input $(i \in \{1, 2, T\}, f \in \mathbb{Z}_p[*])$, **if there exists g such that $\mathcal{L}_i[g] \neq \perp$ and $g(\vec{v}) = f(\vec{v})$ where**¹²

$$\vec{v} = (c, (x_j)_{j \in [T]}, (\alpha_{\ell,j})_{\ell \in [N] \setminus \mathcal{C}, j \in [\hat{w}_\ell]}, (s_\ell)_{\ell \in [N] \setminus \mathcal{C}}, r)$$

return $\mathcal{L}_i[g]$. Otherwise, the oracle samples a label $l_i \leftarrow S_i$, sets $S_i \leftarrow S_i \setminus \{l_i\}$, $\mathcal{L}_i[f] = l_i$, and return l_i .

- *Group operation oracle*: On input $(i \in \{1, 2, T\}, l_1 \in \{0, 1\}^{p'}, l_2 \in \{0, 1\}^{p'}, a_1 \in \mathbb{Z}_p, a_2 \in \mathbb{Z}_p)$, the oracle checks whether there exists $f_1, f_2 \in \mathbb{Z}_p[*]$ such that $\mathcal{L}_i[f_1] = l_1$ and $\mathcal{L}_i[f_2] = l_2$ (where the labelling function is determined via the labelling oracle). If not, it returns $\perp$. Otherwise, it computes $g = a_1 f_1 + a_2 f_2$, executes the internal labelling for $\mathcal{L}_i[g]$ (i.e., calling the labelling oracle with $g \in \mathbb{Z}_p[*]$), and returns $\mathcal{L}_i[g]$.
- *Pairing oracle*: On input $(l_1 \in \{0, 1\}^{p'}, l_2 \in \{0, 1\}^{p'})$, the oracle checks whether there exists $f_1, f_2 \in \mathbb{Z}_p[*]$ such that $\mathcal{L}_1(f_1) = l_1$ and $\mathcal{L}_2(f_2) = l_2$. If not, it returns $\perp$. Otherwise, it computes $g = f_1 f_2$, executes the internal labelling for $\mathcal{L}_T[g]$, and returns $\mathcal{L}_T[g]$.

We note that this is simply a syntactical change and the labels are still assigned in the same manner as in $\mathbf{G}'_0$, so $\Pr[\mathbf{G}'_0^{\mathcal{A}}(\lambda) = 1] = \Pr[\mathbf{G}'_1^{\mathcal{A}}(\lambda) = 1]$.

$\mathbf{G}'_2$: In this game, we introduce the following event **Bad**: when a new polynomial representation is computed but there already exists another polynomial with the same evaluation. If **Bad** occurs, the game aborts. In particular, the bad event is introduced to the internal labelling oracle as follows: On input $(i \in \{1, 2, T\}, f \in \mathbb{Z}_p[*])$, **if there exists g such that $\mathcal{L}_i[g] \neq \perp$ and $g(\vec{v}) = f(\vec{v})$ where $\vec{v} = (c, (x_j)_{j \in [T]}, (\alpha_{\ell,j})_{\ell \in [N] \setminus \mathcal{C}, j \in [\hat{w}_\ell]}, (s_\ell)_{\ell \in [N] \setminus \mathcal{C}}, r)$, then **if additionally $g \neq f$, denote this as **Bad**, and the game aborts; if not, return $\mathcal{L}_i[g]$.** Otherwise, the oracle samples a label $l_i \leftarrow S_i$, sets $S_i \leftarrow S_i \setminus \{l_i\}$, $\mathcal{L}_i[f] = l_i$, and return l_i .**

¹² This is the vector of sampled values for the formal variable introduced. Note that for $\ell \in \mathcal{C}$, the key generation randomness is chosen by the adversary.

It is easy to see that $|\Pr[\mathbf{G}'_1^{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{G}'_2^{\mathcal{A}}(\lambda) = 1]| \leq \Pr[\text{Bad}]$. We will now upper bound $\Pr[\text{Bad}]$. We then show the following claim.

Claim. Let $Q = Q_{\mathbb{G}} + B_{\max}N + 7 + 4W_{\max} + 4W_{\max}^2 + N(4B_{\max} + 4W_{\max}^2 + 4W_{\max}B_{\max})$, $\Pr[\text{Bad}] \leq \frac{Q^2 \cdot (4W_{\max} + 2B_{\max} + 2NB_{\max})}{p-1}$.

Proof. Note that Q is the upper bound on the number of times the labelling oracle is called. We first let E_i for $i \in [Q]$ be the event that **Bad** is triggered in the i -th call to the labelling oracle.

To be more specific, E_i is the event that for the i -th call to the labelling oracle with input $(k_i \in \{1, 2, T\}, f_i \in \mathbb{Z}_p[*])$

$$E_i := (\exists g : \mathcal{L}_{k_i}[g] \neq \perp \wedge g \neq f_i \wedge g(\vec{v}) = f_i(\vec{v})) .$$

Thus, $\text{Bad} = \bigvee_{i=1}^Q E_i$, so $\Pr[\text{Bad}] \leq \sum_{i=1}^Q \Pr[E_i]$.

Then, via a union bound over prior polynomial presentations before the i -th query, we can bound each $\Pr[E_i]$ separately. Here, $\Pr[E_i]$ is the probability of sampling $\vec{v}$ such that for any distinct Laurent polynomials $g \neq f$ that can be constructed in the game, $g(\vec{v}) = f(\vec{v})$. In this case, observe that g and f can be written as linear combination of (a) products between one polynomial representation in $\mathbb{G}_1$ and one in $\mathbb{G}_2$ and (b) polynomial representation in $\mathbb{G}_T$. We now bound the maximum degree and number of variables in each monomial that appears in these terms. For simplicity, we normalize the negative degrees in S_ℓ in $\mathbb{G}_1, \mathbb{G}_2$ by multiplying each polynomial with $\prod_{\ell \in [N] \setminus C} S_\ell^{B_{\max}}$; as a result, this also means we normalize the Laurent polynomials in $\mathbb{G}_T$ with $\prod_{\ell \in [N] \setminus C} S_\ell^{2B_{\max}}$. Therefore, the maximum total degree of $(f - g) \prod_{\ell \in [N] \setminus C} S_\ell^{2B_{\max}} \neq 0$ is $2 \max\{2W_{\max} + 1, W_{\max} + B_{\max} + 2\} + 2NB_{\max}$. Therefore, via Schwartz-Zippel lemma, we have that $\Pr[E_i] \leq i \cdot \frac{2 \max\{2W_{\max} + 1, W_{\max} + B_{\max} + 2\} + 2NB_{\max}}{p-1} \leq i \cdot \frac{4W_{\max} + 2B_{\max} + 2NB_{\max}}{p}$. Here, $p-1$ is due to sampling s_ℓ from $\mathbb{Z}_p^*$. Thus, we can conclude that

$$\Pr[\text{Bad}] \leq \frac{Q^2 \cdot (4W_{\max} + 2B_{\max} + 2NB_{\max})}{p-1}$$

□

G'_3: This is exactly the game $\mathbf{G}_{1,b}$. Note that due to the abort introduced in $\mathbf{G}'_2$, all underlying polynomials of each label are distinct. This corresponds directly to the symbolic game $\mathbf{G}_{1,b}$. Therefore, $\Pr[\mathbf{G}'_2^{\mathcal{A}}(\lambda) = 1] = \Pr[\mathbf{G}'_3^{\mathcal{A}}(\lambda) = 1]$.

Thus, we conclude the proof. □

E.5 Proof of Lemma E.4

To show that the view of the adversary $\mathcal{A}$ is identical in both games $\mathbf{G}_1$ and $\mathbf{G}_2$, we have to show that the monomial $M^* = RC^{W_{\max}+1}X_1$ is not in the span

$\text{Span}(L_T)$, where L_1, L_2, L_T are sets (we use different fonts from the labelling maps $\mathcal{L}_i$) of embedded polynomials presented to the adversary in each respective group, defined as:

$$\begin{aligned} L_1 &:= \left\{ 1, (\mathbf{C}^i)_{i \in [2W_{\max}]}, (\mathbf{C}^i \mathbf{T}_j(\vec{\mathbf{X}}))_{i \in [2W_{\max}] \setminus \{W_{\max}+1\}, j \in [W_{\max}]}, \mathbf{R}, \mathbf{RZ}, (\mathbf{RS}_\ell^k)_{\ell \in \mathcal{HS}, k \in [B_{\max}]} \right. \\ &\quad \left. ((\mathbf{S}_\ell^k)_{k \in [-B_{\max}, B_{\max}]}, (\mathbf{A}_{\ell,j})_{j \in [\hat{w}_\ell]}, (\mathbf{C}^i \mathbf{A}_{\ell,j})_{i \in [2W_{\max}] \setminus \{W_{\max}+1\}, j \in [\hat{w}_\ell]})_{\ell \in [N] \setminus \mathcal{C}} \right\}, \\ L_2 &:= \left\{ 1, (\mathbf{C}^i)_{i \in [2W_{\max}]}, (\mathbf{C}^i \mathbf{T}_j(\vec{\mathbf{X}}))_{i \in [2W_{\max}] \setminus \{W_{\max}+1\}, j \in [W_{\max}]}, \right. \\ &\quad \left. ((\mathbf{S}_\ell^k)_{k \in [-B_{\max}, B_{\max}]}, (\mathbf{C}^i \mathbf{A}_{\ell,j})_{i \in [2W_{\max}] \setminus \{W_{\max}+1\}, j \in [\hat{w}_\ell]}, (\mathbf{A}_{\ell,j})_{j \in [\hat{w}_\ell]}} \right. \\ &\quad \left. 4\text{in } (\mathbf{C}^{W_{\max}+1} \mathbf{A}_{\ell,j} \mathbf{S}_\ell^k)_{j \in [\hat{w}_\ell], k \in [-B_{\max}, B_{\max}] \setminus \{0\}})_{\ell \in [N] \setminus \mathcal{C}} \right\}, \\ L_T &:= (L_1 \otimes L_2) \cup \{\mathbf{C}^{W_{\max}+1} \mathbf{X}_1\} \end{aligned}$$

We then proceed in the following steps to rule out terms in L_T that cannot contribute to the linear combination to recover $\mathbf{M}^*$.

Ruling out non-R-factor terms. Our first step is to rule out terms that does not have $\mathbf{R}$ as a factor, since $\mathbf{M}^*$ contains $\mathbf{R}$ as a factor. Via inspection, we can see that

$$\{\mathbf{R}, \mathbf{RZ}, (\mathbf{RS}_\ell^k)_{\ell \in \mathcal{HS}, k \in [B_{\max}]}\} \otimes L_2$$

are the only subset of L_T containing terms of $\mathbf{R}$ as a factor, and other terms does not contain monomials with $\mathbf{R}$ as a factor.

Simplifying out corrupted parties in $\mathbf{RZ}$. Recall that for corrupted parties $\ell \in \mathcal{C}$, the adversary knows the underlying scalar randomness used to sample the keys. Then, with $\mathbf{RC}^i$ available in L_T , we can simplify $\mathbf{RZ}$ to

$$\mathbf{R} \left(\sum_{\ell \in I^* \cap \mathcal{H}, i \in \Omega_\ell} \mathbf{C}^i (\mathbf{T}_i(\vec{\mathbf{X}}) + \mathbf{A}_{\ell,j[i]}) + \sum_{\ell \in I^* \cap \mathcal{C}, i \in \Omega_\ell} \mathbf{C}^i \mathbf{T}_i(\vec{\mathbf{X}}) \right).$$

Moreover, the set L_2 is simplified to $L'_2 = \{1\} \cup S_1 \cup S_2 \cup S_3 \cup S_4 \cup S_5$ where each set is defined as follows.

$$\begin{aligned} S_1 &:= \{\mathbf{C}^i\}_{i \in [2W_{\max}]}, \\ S_2 &:= \{\mathbf{C}^i \mathbf{T}_j(\vec{\mathbf{X}})\}_{i \in [2W_{\max}] \setminus \{W_{\max}+1\}, j \in [W_{\max}]}, \\ S_3 &:= \{\mathbf{S}_\ell^k\}_{\ell \in [N] \setminus \mathcal{C}, k \in [-B_{\max}, B_{\max}] \setminus \{0\}}, \\ S_4 &:= \{\mathbf{A}_{\ell,j}\}_{\ell \in [N] \setminus \mathcal{C}, j \in [\hat{w}_\ell]} \cup \{\mathbf{C}^i \mathbf{A}_{\ell,j}\}_{\ell \in [N] \setminus \mathcal{C}, j \in [\hat{w}_\ell], i \in [2W_{\max}] \setminus \{W_{\max}+1\}}, \\ S_5 &:= \{\mathbf{C}^{W_{\max}+1} \mathbf{A}_{\ell,j} \mathbf{S}_\ell^k\}_{\ell \in [N] \setminus \mathcal{C}, j \in [\hat{w}_\ell], k \in [-B_{\max}, B_{\max}] \setminus \{0\}}. \end{aligned}$$

Removing terms with $\mathbf{S}_\ell^k$ for $k \neq 0$. Observe that $\mathbf{M}^*$ does not contain monomials with $\mathbf{S}_\ell^k$ for any $\ell \in [N]$ and $k \neq 0$. Moreover, only the following terms contain monomials with $\mathbf{S}_\ell^k$ and only consists of such monomials:

- $\{\mathbf{R}, \mathbf{RZ}\} \otimes (S_3 \cup S_5)$: Since $\mathbf{R}, \mathbf{RZ}$ does not contain any monomial with $\mathbf{S}_\ell$ and S_3, S_5 only consists of such monomials, all the terms in this set consist of only monomials with $\mathbf{S}_\ell^k$ for $k \neq 0$.

- For each $\ell \in \mathcal{HS}$, $k \in [B_{\max}]$, $\{\mathbf{RS}_\ell^k\} \otimes (L_2 \setminus \{\mathbf{S}_\ell^{-k}, (\mathbf{C}^{W_{\max}+1} \mathbf{S}_\ell^{-k} \mathbf{A}_{\ell,j})_{j \in [\hat{w}_\ell]}\})$:
Since $\mathbf{RS}_\ell^k$ has degree k for $\mathbf{S}_\ell$, only terms with $\mathbf{S}_\ell^{-k}$ can cancel the $\mathbf{S}_\ell$ out, otherwise, the degree of $\mathbf{S}_\ell$ would be non-zero. Note that the remaining terms are $\mathbf{R}$ and $\mathbf{RC}^{W_{\max}+1} \mathbf{A}_{\ell,j}$ for $\ell \in \mathcal{HS}$, $j \in [\hat{w}_\ell]$.

Removing cross terms $\mathbf{X}_i \mathbf{X}_j, \mathbf{X}_i \mathbf{A}_{\ell,j}, \mathbf{A}_{\ell,j} \mathbf{A}_{\ell',j'}$. Next, observe that $\mathbf{M}^*$ does not contain any monomial with combined degree of $\mathbf{X}_i, \mathbf{A}_{\ell,j}$ at least 2. Also, the polynomials in $\{\mathbf{RZ}\} \otimes (S_2 \cup S_4)$ contains such monomials and is a sum of only these monomials. Thus, we can argue that $\{\mathbf{RZ}\} \otimes (S_2 \cup S_4)$ does not contribute to $\mathbf{M}^*$. At this point, the remaining monomials are

$$\{\mathbf{R}\} \otimes (S_1 \cup S_2 \cup S_4) \cup \{\mathbf{RZ}\} \otimes S_1 \cup \{\mathbf{RA}_{\ell,j} \mathbf{C}^{W_{\max}+1}\}_{\ell \in \mathcal{HS}, j \in [w_\ell]}.$$

Change of basis for $\{\mathbf{RZ}\} \otimes \{\mathbf{C}^i\}_{i \in [W_{\max}]}$. In this step, we will argue that the linear span of the following two sets are identical:

$$S' := (\{\mathbf{RZ}\} \otimes \{\mathbf{C}^i\}_{i \in [W_{\max}]}) \cup (\{\mathbf{R}\} \otimes (S_2 \cup S_4)) \cup \{\mathbf{RA}_{\ell,j} \mathbf{C}^{W_{\max}+1}\}_{\ell \in \mathcal{HS}, j \in [w_\ell]}$$

$$S'' := \{\mathbf{RC}^{W_{\max}+1}(\mathbf{T}_i(\vec{\mathbf{X}}) + \mathbf{A}_{\ell[i],j[i]})\}_{i:\ell[i] \in I^* \setminus (\mathcal{HS} \cup \mathcal{C})} \cup \{\mathbf{RC}^{W_{\max}+1} \mathbf{T}_i(\vec{\mathbf{X}})\}_{i:\ell[i] \in \mathcal{C} \cup \mathcal{HS}} \\ \cup (\{\mathbf{R}\} \otimes (S_2 \cup S_4)) \cup \{\mathbf{RA}_{\ell,j} \mathbf{C}^{W_{\max}+1}\}_{\ell \in \mathcal{HS}, j \in [w_\ell]}$$

To show this, we will show that $\mathbf{RZC}^{W_{\max}+1-i'}$ for $i' \in [W]$ can be interchanged with the highlighted terms in S'' , and the same terms for $i' \in [W+1, W_{\max}]$ are already spanned by other terms in the set.

First, observe that for $i' \in [W]$

$$\begin{aligned} & \mathbf{RZ} \cdot \mathbf{C}^{W_{\max}+1-i'} \\ &= \mathbf{R} \left(\sum_{i \in [W]: \ell[i] \notin \mathcal{C}} \mathbf{C}^{W_{\max}+1+i-i'} (\mathbf{T}_i(\vec{\mathbf{X}}) + \mathbf{A}_{\ell[i],j[i]}) + \sum_{i \in [W]: \ell[i] \in \mathcal{C}} \mathbf{C}^{W_{\max}+1+i-i'} \mathbf{T}_i(\vec{\mathbf{X}}) \right) \end{aligned} \quad (24)$$

$$\begin{aligned} &= \sum_{i \neq i': \ell[i] \notin \mathcal{C}} \mathbf{RC}^{W_{\max}+1+i-i'} (\mathbf{T}_i(\vec{\mathbf{X}}) + \mathbf{A}_{\ell[i],j[i]}) + \sum_{i \neq i': \ell[i] \in \mathcal{C}} \mathbf{RC}^{W_{\max}+1+i-i'} \mathbf{T}_i(\vec{\mathbf{X}}) \\ &\quad + \begin{cases} \mathbf{RC}^{W_{\max}+1} \mathbf{T}_{i'}(\vec{\mathbf{X}}) & ; i' \text{ s.t. } \ell[i'] \in \mathcal{C} \\ \mathbf{RC}^{W_{\max}+1} (\mathbf{T}_{i'}(\vec{\mathbf{X}}) + \mathbf{A}_{\ell[i'],j[i']}) & ; i' \text{ s.t. } \ell[i'] \in I^* \setminus \mathcal{C} \end{cases} \end{aligned} \quad (25)$$

All the terms in the RHS are elements of S'' , meaning $\text{Span}(S') \subseteq \text{Span}(S'')$. On the other hand, the terms in RHS of Equation (25) with degree of $\mathbf{C}$ that is not $W_{\max} + 1$ already appears in $\{\mathbf{R}\} \otimes (S_2 \cup S_4)$, and $\mathbf{RC}^{W_{\max}+1} \mathbf{A}_{\ell[i'],j[i']}$ for i' such that $\ell[i'] \in \mathcal{HS}$ are also available. Therefore, the highlighted terms in S'' are linear combinations of polynomials in S' . Therefore, $\text{Span}(S'') \subseteq \text{Span}(S')$, so the spans of the two sets are identical. Thus, we can change $\{\mathbf{RZ}\} \otimes \{\mathbf{C}^i\}_{i \in [W]}$ with the highlighted terms in S'' .

Moreover, for $\mathbf{RZ} \cdot \mathbf{C}^{W_{\max}+1-i'}$ with $i' \in [W+1, W_{\max}]$, we can see that the right-hand side of Equation (24) consists of only terms that are in the span

of $\{R\} \otimes (S_2 \cup S_4)$. This is because Z only sums up to degree W in the variable C , so $RZ \cdot C^{W_{\max}+1-i'}$ is a sum of monomials with degree $\neq W_{\max} + 1$ in C . Hence, they can be removed.

Removing terms with C^i for $i \neq W_{\max} + 1$. Since M^* only contains a monomial with degree of C being $W_{\max} + 1$, terms that consist of monomials with degree of C not equals to $W_{\max} + 1$ cannot contribute in the linear combination. Hence, we can rule out the following terms:

- $\{RZ\} \otimes \{1, (C^i)_{i \in [W_{\max}+1, 2W_{\max}]}\}$: By how Z is defined, RZ only consists of monomials with degree of C at most $W_{\max}$, and RZC^i for $i \in [W_{\max} + 1, 2W_{\max}]$ consist of monomials with degree more than $W_{\max} + 1$.
- $\{R\} \otimes (\{1\} \cup (S_1 \setminus \{C^{W_{\max}+1}\}) \cup S_2 \cup S_4)$: Since $R \otimes S_5$ are already ruled out in prior step, the remaining terms in $\{R\} \otimes L_2$ with degree of C at $W_{\max} + 1$ is $RC^{W_{\max}+1}$.

The remaining terms are: (a) $RC^{W_{\max}+1}$, (b) $RC^{W_{\max}+1}(\mathbf{T}_i(\vec{X}) + \mathbf{A}_{\ell[i],j[i]})$ for i such that $\ell[i] \notin \mathcal{HS} \cup \mathcal{C}$, (c) $RC^{W_{\max}+1}\mathbf{A}_{\ell,j}$ for $\ell \in \mathcal{HS}, j \in [\hat{w}_\ell]$, and (d) $RC^{W_{\max}+1}\mathbf{T}_i(\vec{X})$ for i such that $\ell[i] \in \mathcal{HS} \cup \mathcal{C}$.

Final linear independence argument. Finally, we will argue that $M^* = RC^{W_{\max}+1}\mathbf{X}_1$ is linearly independent from the remaining terms. Suppose we can write

$$\begin{aligned} RC^{W_{\max}+1}\mathbf{X}_1 = & aRC^{W_{\max}+1} + \sum_{i \in [W]: \ell[i] \notin \mathcal{HS} \cup \mathcal{C}} b_i RC^{W_{\max}+1}(\mathbf{T}_i(\vec{X}) + \mathbf{A}_{\ell[i],j[i]}) \\ & + \sum_{\ell \in \mathcal{HS}, j \in [\hat{w}_\ell]} c_{\ell,j} RC^{W_{\max}+1}\mathbf{A}_{\ell,j} + \sum_{i: \ell[i] \in \mathcal{HS} \cup \mathcal{C}} d_i RC^{W_{\max}+1}\mathbf{T}_i(\vec{X}). \end{aligned}$$

First, $a = 0$ due to degree mismatch as other terms has degree at least $W_{\max} + 3$ while $RC^{W_{\max}+1}$ only has degree $W_{\max} + 2$. Then, since the set the variables $\mathbf{A}_{\ell,j}$ in the first and second summation are disjoint due to the two sums iterating over $\ell[i] \notin \mathcal{HS} \cup \mathcal{C}$ and $\ell \in \mathcal{HS}$, respectively, we have that $b_i = c_{\ell,j} = 0$ for all corresponding i, j, ℓ . Therefore, we have that $RC^{W_{\max}+1}\mathbf{X}_1 = \sum_{i: \ell[i] \in \mathcal{HS} \cup \mathcal{C}} d_i RC^{W_{\max}+1}\mathbf{T}_i(\vec{X})$. This is a contradiction, because of (1) $\sum_{\ell \in (\mathcal{HS} \cup \mathcal{C}) \cap I^*} w_\ell < T$, (2) $\mathbf{T}_j(\vec{X}) = \sum_{i=1}^T j^{i-1}\mathbf{X}_i$, and (3) the rank of the Vandermonde matrix $(j^i)_{j \in [W_{\max}], i \in [0, T-1]}$ is T .

Thus, we conclude that $RC^{W_{\max}+1}\mathbf{X}_1$ remains hidden from the adversary. $\square$

F Proof of Theorem 5.7

Let $\mathcal{A}$ be an adversary in T-IND-CCA game of fSWBTX, making at most $Q_{\text{PDec}} = Q_{\text{PDec}}$ queries to its partial decryption oracle and running in time $T_{\mathcal{A}} = T_{\mathcal{A}}(\lambda)$. Also, let Sim_{xx} be the simulator for the simulation-extractability property of Π_{xx} for $\text{xx} \in \{\text{pk}, \text{ct}\}$. We now consider the following sequence of games for $b \in \{0, 1\}$.

$\mathbf{G}_{0,b}$. This is exactly the game T-IND-CCA $_{\text{fSWBTX},b}^{\mathcal{A}}$.

$\mathbf{G}_{1,b}$. This game is identical to $\mathbf{G}_{0,b}$ except that the public parameters for Π_{pk} is generated as $(\text{pp}_{\text{pk}}, \text{td}_{\text{pk}}) \leftarrow \text{Sim}_{\text{pk}}.\text{Setup}(1^\lambda)$. Also, during key generation for

honest parties' keys (in the oracle $\mathcal{O}_{\text{HUser}}$) the game instead simulate the proof $\pi_{\text{pk}} \leftarrow \text{Sim}_{\text{pk}}(\text{td}_{\text{pk}}, \mathbb{X}_{\text{pk}})$ with $\mathbb{X}_{\text{pk}}$ defined as in fSWBTX.KeyGen . Moreover, during $\mathcal{O}_{\text{CUser}}$ oracle query, the game additionally does the following:

- If $(\text{pk}_\ell, \hat{w}_\ell) \in \{(\text{pk}_j, \hat{w}_j)\}_{j \in \mathcal{H}}$, return $\perp$.
- Runs the key verification algorithm $\text{VerKey}(\text{pk}_\ell)$.
- If VerKey returns 1, run the extractor $(s_\ell, (\alpha_{\ell,j})_{j \in [\hat{w}_\ell]}) \leftarrow \text{Sim}_{\text{pk}}.\text{Ext}(\text{td}_{\text{pk}}, \mathbb{X}_{\text{pk},\ell}, \pi_{\text{pk},\ell})$ with $\mathbb{X}_{\text{pk},\ell}$ parsed from pk_ℓ .
- If this extraction fails but the key verification passes, the game aborts.

Here, the view of $\mathcal{A}$ changes with the new aborts or the simulated proof. Note that the added check $\text{VerKey}(\text{pk}_\ell)$ does not matter since VerKey check will be made again during challenge oracle inside Prep . Due to the key distinction check, the game only extracts on statement and proof pair that does not correspond to prior simulation. It is then easy to see that by simulation extractability of Π_{pk} , there exists $\mathcal{B}_{\text{simext},1}$ running in time roughly that of $\mathcal{A}$ such that

$$|\Pr[\mathbf{G}_{1,b}^{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{G}_{0,b}^{\mathcal{A}}(\lambda) = 1]| \leq \text{Adv}_{\Pi_{\text{pk}}, \text{Sim}_{\text{pk}}}^{\text{simext}}(\mathcal{B}_{\text{simext},1}, \lambda).$$

$\mathbf{G}_{2,b}$. This game is identical to $\mathbf{G}_{1,b}$ except that the public parameters for Π_{ct} is generated as $(\text{pp}_{\text{ct}}, \text{td}_{\text{ct}}) \leftarrow \text{Sim}_{\text{ct}}.\text{Setup}(1^\lambda)$. Also, during challenge ciphertext generation (in the oracle $\mathcal{O}_{\text{Chal}}$) the game instead simulate the proof $\pi_{\text{ct}} \leftarrow \text{Sim}_{\text{ct}}(\text{td}_{\text{ct}}, \mathbb{X}_{\text{ct}})$ with $\mathbb{X}_{\text{ct}}$ defined as in fSWBTX.Enc . Moreover, when the adversary queries the partial decryption oracle for the ℓ -th ($\ell \in [N]$) party with a batch $(\text{ct}_k)_{k \in [B]}$, the game does the following:

- First, compute the subset $S \subseteq [B]$ such that $\text{ct}_k \neq \text{ct}^*$ for $k \in S$.
- Compute the subset $U \subseteq [B]$ such that for $k \in U$, $\Pi_{\text{ct}}.\text{Ver}(\mathbb{X}_{\text{ct},k}, \pi_{\text{ct},k}) = 1$ with $\mathbb{X}_{\text{ct},k}, \pi_{\text{ct},k}$ parsed from ct_k as in the decryption algorithm.
- For $k \in S \cap U$, extract $r_k \leftarrow \text{Sim}_{\text{ct}}.\text{Ext}(\text{td}_{\text{ct}}, \mathbb{X}_{\text{ct},k}, \pi_{\text{ct},k})$ and computes $\mu_k = \text{ct}_{k,3} - r_k \cdot \llbracket c^{W_{\max}+1} t \rrbracket_{\text{T}}$. If the extraction fails, the game aborts.

Here, the view of $\mathcal{A}$ only changes from the simulated proof and the added abort from extraction. Due to the check for non-challenge ciphertext, the game only extracts on statement and proof pair that does not correspond to the prior simulation. It is then easy to see that by simulation-extractability of Π_{ct} , there exists $\mathcal{B}_{\text{simext},2}$ running in time roughly that of $\mathcal{A}$ such that

$$|\Pr[\mathbf{G}_{2,b}^{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{G}_{1,b}^{\mathcal{A}}(\lambda) = 1]| \leq \text{Adv}_{\Pi_{\text{ct}}, \text{Sim}_{\text{ct}}}^{\text{simext}}(\mathcal{B}_{\text{simext},2}, \lambda).$$

Finally, we show that there exists an adversary $\mathcal{B}$

$$|\Pr[\mathbf{G}_{2,0}^{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{G}_{2,1}^{\mathcal{A}}(\lambda) = 1]| \leq \text{Adv}_{\text{fSWBTX}_{\text{cpa}}}^{\text{t-ind-cpa}}(\mathcal{B}, \lambda).$$

We consider the following adversary $\mathcal{B}$ which runs $\mathcal{A}$ as a subroutine.

- First, it forwards the parameters $(W_{\max}, B_{\max}, T)$ from $\mathcal{A}$ to its game. Then, it receives the public parameters pp' and runs the simulators $(\text{pp}_{\text{pk}}, \text{td}_{\text{pk}}) \leftarrow \text{Sim}_{\text{pk}}.\text{Setup}(1^\lambda)$ and $(\text{pp}_{\text{ct}}, \text{td}_{\text{ct}}) \leftarrow \text{Sim}_{\text{ct}}.\text{Setup}(1^\lambda)$. Next, it forwards $\text{pp} \leftarrow (\text{pp}', \text{pp}_{\text{pk}}, \text{pp}_{\text{ct}})$ to $\mathcal{A}$.

- Now, we explain how $\mathcal{B}$ answers $\mathcal{A}$'s oracle queries:
 - Oracle $\mathcal{O}_{\text{HUser}}$:** On a call to this oracle with input $\hat{w}$,
 - The reduction keeps track of the number of parties N and the honest and corrupted party sets $\mathcal{H}$ and $\mathcal{C}$ as in the game. Let ℓ be the index for the current user and set $\hat{w}_\ell \leftarrow \hat{w}$.
 - Calls $\mathcal{O}_{\text{HUser}}$ of its own game to receive $\text{pk}'_\ell \leftarrow \mathcal{O}_{\text{HUser}}(\hat{w}_\ell)$.
 - Run the simulator $\pi_{\text{pk},\ell} \leftarrow \text{Sim}_{\text{pk}}(\text{td}_{\text{pk}}, \mathbb{X}_{\text{pk},\ell})$ with $\mathbb{X}_{\text{pk},\ell}$ parsed from pk'_ℓ and set $\text{pk}_\ell \leftarrow (\text{pk}'_\ell, \pi_{\text{pk},\ell})$.
 - Return pk_ℓ to $\mathcal{A}$.
 - Oracle $\mathcal{O}_{\text{CUser}}$:** On a query $(\text{pk}_\ell = (\text{pk}'_\ell, \pi_{\text{pk},\ell}), \hat{w}_\ell)$. $\mathcal{B}$ does the following:
 - The reduction keeps track of the number of parties N and the honest and corrupted party sets $\mathcal{H}$ and $\mathcal{C}$ as in the game. Let ℓ be the index for the current user.
 - If $(\text{pk}_\ell, \hat{w}_\ell) \in \{(\text{pk}_j, \hat{w}_j)\}_{j \in \mathcal{H}}$, abort.
 - Verify the key using $\text{VerKey}(\text{pk}_\ell)$ algorithm.
 - If verification passes, run $(s_\ell, (\alpha_{\ell,j})_{j \in [\hat{w}_\ell]}) \leftarrow \text{Sim}_{\text{pk}}.\text{Ext}(\text{td}_{\text{pk}}, \mathbb{X}_{\text{pk},\ell}, \pi_{\text{pk},\ell})$ with $\mathbb{X}_{\text{pk},\ell}$ parsed from pk_ℓ . If the extractor fails to extract at any points, $\mathcal{B}$ aborts.
 - If VerKey fails, set $(s_\ell, (\alpha_{\ell,j})_{j \in [\hat{w}_\ell]}) = (1, 0, \dots, 0)$. (Note: This is simply to keep a dummy party so that the party count of $\mathcal{A}$ within the simulation and the IND-CPA game that $\mathcal{B}$ is playing is identical.)
 - Call $\mathcal{O}_{\text{CUser}}$ with input $\rho_\ell \leftarrow (s_\ell, (\alpha_{\ell,j})_{j \in [\hat{w}_\ell]})$ and $\hat{w}_\ell$. (Recall that the randomness of KeyGen is exactly these scalars.)
 - Challenge oracle $\mathcal{O}_{\text{Chal}}$:** On input $(J = (\ell_1, \dots, \ell_L) \in [N]^L, \mu_0, \mu_1, (w_\ell)_{\ell \in J})$, run first the preprocessing algorithm on $(\text{pk}_{\ell_j}, w_{\ell_j})_{j \in [L]}$ to obtain $((\text{ek}^*, \text{dk}^*), I' \subseteq [L])$ and abort if $\text{ek}^* = \perp$ or the chosen weights do not satisfy the constraints. (Note: I' only contains indices to valid keys where underlying randomness can be extracted since the preprocessing algorithm runs VerKey as a subroutine.) Then, $\mathcal{B}$ makes a query to its own challenge oracle with inputs $(I^* = (\ell_j)_{j \in I'}, \mu_0, \mu_1, (w_\ell)_{\ell \in I^*})$ and forwards the output $(\text{ct}^*, \pi_{\text{ct}})$ to $\mathcal{A}$ where $\pi_{\text{ct}} \leftarrow \text{Sim}_{\text{ct}}(\text{td}, \mathbb{X}_{\text{ct}})$ with $\mathbb{X}_{\text{ct}}$ constructed from ct^* .
 - Partial decryption oracle $\mathcal{O}_{\text{PDec}}$:** On input $(\text{ek}, \ell \in \mathcal{H}, (\text{ct}_k)_{k \in [B]})$ with $\text{ct}_k = (\text{ct}_{k,1}, \text{ct}_{k,2}, \text{ct}_{k,3}, \pi_{\text{ct},k})$ for all $k \in [B]$, do the following
 - Compute the set $U \subseteq [B]$ as in PartDec algorithm, i.e., by checking the NIZK proofs $\pi_{\text{ct},k}$.
 - For each $k \in U$, if $\text{ct}_k \neq \text{ct}^*$, run $r_k \leftarrow \text{Sim}_{\text{ct}}.\text{Ext}(\text{td}_{\text{ct}}, \mathbb{X}_{\text{ct},k}, \pi_{\text{ct},k})$ with $\mathbb{X}_{\text{ct},k}, \pi_{\text{ct},k}$ parsed from ct_k and set $\mu_k \leftarrow \text{ct}_{k,3} - r_k \cdot \llbracket c^{W_{\max}+1} t \rrbracket_{\text{T}}$. If the extraction fails, $\mathcal{B}$ aborts. Otherwise, set $\rho_k \leftarrow r_k$.
 - For each $k \in [B] \setminus U$, set $\text{ct}_k \leftarrow \text{Enc}(\text{ek}, \llbracket 0 \rrbracket_{\text{T}}; 0)$ and $\rho_k \leftarrow 0, \mu_k \leftarrow \llbracket 0 \rrbracket_{\text{T}}$.
 - Make a query to $\mathcal{O}_{\text{PDec}}(\text{ek}, \ell, (\text{ct}'_k = (\text{ct}_{k,1}, \text{ct}_{k,2}, \text{ct}_{k,3}), \mu_k, \rho_k)_{k \in [B]})$ to $\mathcal{B}$'s own oracle, and return the output pd_ℓ to $\mathcal{A}$.
- At the end of the game, output the bit that $\mathcal{A}$ outputs.

We can see that $\mathcal{B}$ runs in time roughly that of $\mathcal{A}$ and makes the same amount of partial decryption queries as $\mathcal{A}$. We now argue that the view of $\mathcal{A}$ is identical to

its view in games $\mathbf{G}_{2,b}$; particularly, the outputs of the two oracles are identically distributed:

Challenge oracle. By how we defined **Prep**, the keys in I^* that does not pass **VerKey** would not be aggregated. Accordingly, by how $\mathcal{B}$ extracts the underlying secret key, only the corrupted keys that pass the **VerKey** check will be forwarded to the T-IND-CPA game and registered as one of the keys. Thus, the subset I' only contains indices corresponding to either the honest parties or corrupted parties with extractable keys. Hence, the underlying oracle call to $\mathcal{O}_{\text{Chal}}$ would use the same set of public key during aggregation and output identically distributed ct^* .

Partial decryption oracle. Here, the ciphertext check and the extraction would rule out the non-challenge and malformed ciphertexts. Then, one can see that with the extracted r_k for non-challenge but well-formed ciphertext, $\mathcal{B}$ can recover μ_k such that $\text{ct}_k = \text{Enc}(\text{ek}, \mu_k; r_k)$. Forwarding the same batch of ciphertext with the malformed ones replaced with encryption of 0 results in the same decryption share as $\sum_{k \in U} s_\ell^k \text{ct}_{k,1} = \sum_{k \in [B]} s_\ell^k \text{ct}'_{k,1}$ where $\text{ct}'_{k,1} = \text{ct}_{k,1}$ for $k \in U$ and $\text{ct}'_{k,1} = \llbracket 0 \rrbracket_1$ for $k \notin U$.

Hence, we have that

$$|\Pr[\mathbf{G}_{2,0}^A(\lambda) = 1] - \Pr[\mathbf{G}_{2,1}^A(\lambda) = 1]| \leq \text{Adv}_{\text{fSWBTX}_{\text{cpa}}}^{\text{t-ind-cpa}}(\mathcal{B}, \lambda).$$

Combining all the bounds concludes the proof. $\square$

G Flexible Threshold Construction

In this section, we give the construction and discuss the security of the flexible-threshold version of the fSWBTX scheme in Sections G.1 and G.2, respectively.

G.1 Construction

We give our full construction in Figures 7 and 8, with the two figures separated into key generation algorithms and encryption-decryption algorithms. The construction follows similar ideas given in [30] which we give an overview below.

We can see our flexible threshold scheme as a fixed threshold scheme with threshold $W_{\max}$ and maximum weight $\widehat{W} = 2W_{\max} - 1$. Then, to accommodate flexible threshold, dummy parties are introduced into the public parameters. In particular, we introduce blinding factors $\beta_{W_{\max}+1}, \dots, \beta_{\widehat{W}}$ into the public parameters along with necessary group elements for decryption.

The general idea is that for an aggregated encryption key pk_{agg} from keys $(\text{pk}_\ell)_{\ell \in [L]}$ with total weight $\sum_{\ell \in [L]} w_\ell =: W \leq W_{\max}$ and threshold $T \leq W$, we want the following property: with total weight T decryption shares on ciphertext with randomness $r \in \mathbb{Z}_p$, one can derive at least $W_{\max}$ terms of the form $\llbracket rc^{\widehat{W}+1} t_j \rrbracket_{\text{T}}$.

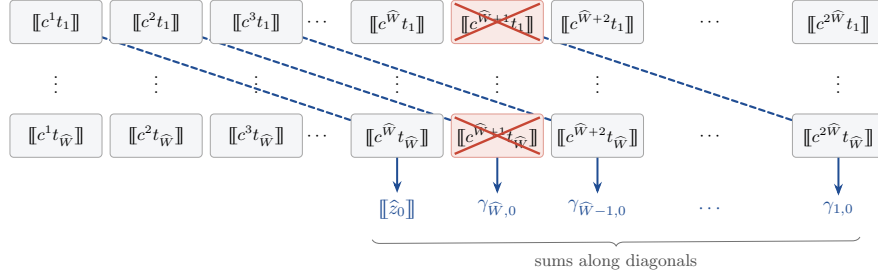


Fig. 6. Figure illustrating public parameters compression done in $\text{SWBTX.Setup}(1^\lambda)$.

Recall that the encryption key for $\widehat{W}$ maximum weight is of the form

$$\text{pk}_{\text{agg}} = \llbracket \sum_{i=1}^{\widehat{W}} c^i(t_i + \alpha_i^*) \rrbracket_1, \quad (26)$$

where α_i^* are the contributions from the individual public keys. In the case of our flexible threshold scheme, the α_i 's can be categorized as follows:

1. W slots derived from public keys of the aggregated parties. In particular, with the partitioning of $[W]$ into intervals $\Omega_\ell = [\Omega_\ell^- + 1, \Omega_\ell^+]$, we define the mappings for $i \in [W]$, $\ell[i] := \ell$ such that $i \in \Omega_\ell$ and $j[i] := i - \Omega_{\ell[i]}^-$. Then, we can think of the aggregated ek as setting $\alpha_i^* := \alpha_{\ell[i], j[i]}$.
2. $\widehat{W} - W_{\max} - T + 1 = W_{\max} - T$ free slots where the α_i are set to 0, i.e., these are the slots where $\llbracket rc^{\widehat{W}+1} t_i \rrbracket_{\text{T}}$ can be derived with public information.
3. $W_{\max} - W + T - 1$ slots filled with dummy parties in the public parameters, i.e., α_i^* is set to β_j for distinct j 's. These are slots that one cannot derive $\llbracket rc^{\widehat{W}+1} t_i \rrbracket_{\text{T}}$.

With at least T decryption shares corresponding to slots of type (1), we can now compute at least $W_{\max}$ values of $\llbracket rc^{\widehat{W}+1} t_j \rrbracket_{\text{T}}$ by combining with ones derived from type (2) slots.

Furthermore, we explain how to compress the group elements in the public parameters.

Compressing $\llbracket c^i t_i \rrbracket$. Notice that these terms are only used in constructing the element pk_{agg} within the encryption key, and the value always contain the sum $\sum_{i \in [\widehat{W}]} \llbracket c^i t_i \rrbracket_1$, so we can pre-aggregate this value in the public parameters.

Compressing $\llbracket c^i t_{i'} \rrbracket$ for $i \neq i'$. Similar to the above, notice that within the decryption algorithm (in the fixed weight scheme in Figure 4 with maximum weight set to $\widehat{W}$), only the sum $\sum_{i' \neq i} \llbracket c^{\widehat{W}+1-i+i'} t_{i'} \rrbracket_2$ are used within the computation. Hence, we could also sum these up beforehand in the public parameters as the terms $\gamma_{i,0}$. Refer to Figure 6 for how the pp is compressed.

Compressing dummy parties' shares. In this flexible threshold scheme, it does not matter which of the dummy parties (whose shares are represented with β_i 's) are aggregated into pk_{agg} as long as exactly $W_{\max} - W + T - 1$ are deterministically selected and aggregated. Hence, one only needs to ensure that any number of dummy shares within the range $[W_{\max} - 1]$ can be incorporated into pk_{agg} . To compress the public parameters' size using this observation, the prior work of Waters and Wu [30] relied on bit-decomposition. Assuming that $W_{\max} = 2^n$ for some positive integer n , the setup will pre-aggregate the shares of $1, 2, \dots, 2^{n-1}$ dummy parties into n blocks defined as

$$y_m = \sum_{i=W_{\max}+2^{m-1}}^{W_{\max}+2^m-1} \llbracket c^i \beta_i \rrbracket_1.$$

Then, at encryption time, bit-decompose $W_{\max} - W + T - 1 = \sum_{m \in [n]} b_m 2^{m-1}$ for $b_1, \dots, b_n \in \{0, 1\}$ during encryption, and aggregate the m -th blocks where $b_m = 1$ into pk_{agg} (see Enc in Figure 8). For decryption, since we do not individually require the cross-term for each β_i , we can also pre-aggregate the cross-terms for each block $m \in [n]$ and for each $i \in [\widehat{W}] \setminus [W_{\max} + 2^{m-1}, W_{\max} + 2^m - 1]$, the setup publishes

$$v_{m,i} = \sum_{j=W_{\max}+2^{m-1}}^{W_{\max}+2^m-1} \llbracket c^{\widehat{W}+1-i+j} \beta_j \rrbracket_2.$$

Note that the cross terms for $i \in [W_{\max} + 2^{m-1}, W_{\max} + 2^m - 1]$ are not needed at any point during the decryption procedure.

ADDING IND-CCA SECURITY. To achieve CCA security, we again rely on simulation extraction NIZKs Π_{pk} and Π_{ct} . We note that Π_{pk} proves a relation identical to $\mathcal{R}_{\text{pk}}$ in Equation (14), but Π_{ct} proves a slightly different relation $\mathcal{R}'_{\text{ct}}$ defined below:

$$\mathcal{R}'_{\text{ct}} := \left\{ \begin{array}{l} \mathbb{X}_{\text{ct}} = (W_{\max}, W, T, \llbracket 1 \rrbracket_1, \text{ek}, \text{ct}_1 = r \llbracket 1 \rrbracket_1 \wedge \text{ct}_2 = r(\text{pk}_{\text{agg}} + \sum_{m=1}^n b_m y_m) \\ \text{ct}' = (\text{ct}_1, \text{ct}_2, \text{ct}_3) \text{ , where } b_1, \dots, b_n \in \{0, 1\} \text{ and} \\ \mathbb{W} = r \mid W_{\max} - W + T - 1 = \sum_{m=1}^n b_m 2^{m-1} \end{array} \right\}. \quad (27)$$

In the equation above, ek is parsed as it is assigned in **Prep**. Note that essentially the NIZK still shows that the ciphertext is generated honestly with the changes due to the threshold being determined at encryption time.

EFFICIENCY. The following theorem summarizes the efficiency guarantees of SWBTX.

Theorem G.1. *The silent batched weighted threshold encryption SWBTX has the following efficiency metrics: Let $W_{\max}$ be the maximum weights, $B_{\max}$ be the maximum batch size, and $\widehat{w}_\ell$ be the maximum weight for party ℓ . Also, $\widehat{W} = 2W_{\max} - 1$ and $n = \log W_{\max}$.*

Alg. Setup($1^\lambda, W_{\max}, B_{\max}$): <pre> 1 : Assume $W_{\max} = 2^n$ for $n \in \mathbb{N}$, $\widehat{W} = 2W_{\max} - 1$ 2 : $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \circ) \leftarrow \text{GGen}(1^\lambda)$ // Implicit to all Alg. 3 : $\text{pp}' \leftarrow \text{fSWBTX.Setup}(1^\lambda, \widehat{W}, B_{\max}, \hat{T} = W_{\max})$ 4 : parse $\text{pp}' = (\widehat{W}, B_{\max}, \llbracket c^{\widehat{W}+1} t \rrbracket_T, (\llbracket c^i \rrbracket_1, \llbracket c^i \rrbracket_2)_{i \in [2\widehat{W}]})$ 5 : $(\llbracket c^i t_j \rrbracket_1, \llbracket c^i t_j \rrbracket_2)_{i \in \mathfrak{W}, j \in [\widehat{W}]}, \text{pp}_{\text{pk}}, \text{pp}_{\text{ct}}$ 6 : $\llbracket z_0 \rrbracket_1 \leftarrow \sum_{i \in [\widehat{W}]} \llbracket c^i t_i \rrbracket_1$ 7 : for $i \in [\widehat{W}]$: $\gamma_{i,0} \leftarrow \sum_{j \in [\widehat{W}] \setminus i} \llbracket c^{\widehat{W}+1+j-i} t_j \rrbracket_2$ 8 : // Generate $W_{\max} - 1$ dummy blinding factors. 9 : $\beta_{W_{\max}+1}, \dots, \beta_{\widehat{W}} \leftarrow \mathbb{Z}_p$ 10 : for $m \in [n]$: 11 : $X_m \leftarrow [W_{\max} + 2^{m-1}, W_{\max} + 2^m - 1]$ 12 : $y_m \leftarrow \sum_{i \in X_m} \beta_i \llbracket c^i \rrbracket_1$ 13 : for $i \in [\widehat{W}] \setminus X_m$: 14 : $v_{m,i} \leftarrow \sum_{j \in X_m} \beta_j \llbracket c^{\widehat{W}+1+j-i} \rrbracket_2$ 15 : $\text{pp} \leftarrow (W_{\max}, B_{\max}, \llbracket c^{\widehat{W}+1} t \rrbracket_T, (\llbracket c^i \rrbracket_1, \llbracket c^i \rrbracket_2)_{i \in [2\widehat{W}]})$ 16 : $\llbracket z_0 \rrbracket_1, (\gamma_{i,0})_{i \in [\widehat{W}]}, (y_m, (v_{m,i})_{i \in [\widehat{W}] \setminus X_m})_{m \in [n]}$ 17 : $\text{pp}_{\text{pk}}, \text{pp}_{\text{ct}}$ 18 : return pp </pre> Alg. KeyGen($\widehat{w}_\ell$): <pre> 1 : if $\widehat{w}_\ell > W_{\max}$ then return $\perp$ 2 : $s_\ell \leftarrow \mathbb{Z}_p^*$; $\alpha_{\ell,1}, \dots, \alpha_{\ell,\widehat{w}} \leftarrow \mathbb{Z}_p$ 3 : $\text{pk}'_\ell \leftarrow \left((\llbracket s_\ell^k \rrbracket_1, \llbracket s_\ell^k \rrbracket_2)_{k \in [-B_{\max}, B_{\max}]} \right)$ $(\xi_{\ell,k,1}, \xi_{\ell,k,2})$ 4 : $(\underbrace{(\llbracket \alpha_{\ell,j} \rrbracket_1, \llbracket \alpha_{\ell,j} \rrbracket_2)}_{(\varphi_{\ell,j,1}, \varphi_{\ell,j,2})}, \underbrace{(\llbracket c^i \alpha_{\ell,j} \rrbracket_1, \llbracket c^i \alpha_{\ell,j} \rrbracket_2)}_{\psi_{\ell,i,j,1}, \psi_{\ell,i,j,2}})_{i \in \mathfrak{W}}$ 5 : $(\underbrace{\llbracket c^{\widehat{W}+1} \alpha_{\ell,j} s^k \rrbracket_2}_{\chi_{\ell,j,k}})_{k \in \mathfrak{B}})_{j \in [\widehat{w}]}$ 6 : $\mathbb{X}_{\text{pk}} \leftarrow (\llbracket 1 \rrbracket_1, \text{pk}'_\ell, \widehat{w}_\ell)$; $\mathbb{W}_{\text{pk}} \leftarrow (s_\ell, (\alpha_{\ell,j})_{j \in [\widehat{w}]})$ 7 : $\pi_{\text{pk}} \leftarrow \Pi_{\text{pk}}.\text{Prove}(\mathbb{X}_{\text{pk}}, \mathbb{W}_{\text{pk}})$ 8 : return $(\text{pk}_\ell = (\text{pk}'_\ell, \pi_{\text{pk}}), \text{sk}_\ell = s_\ell)$ </pre>	Alg. Prep($\{(\text{pk}_\ell, w_\ell)\}_{\ell \in [L]}$): <pre> 1 : $I \leftarrow \{\ell \in [L] : \text{VerKey}(\text{pk}_\ell) = 1 \wedge w_\ell \leq \widehat{w}_\ell\}$ 2 : $W \leftarrow \sum_{\ell \in I} w_\ell$ 3 : if $W > W_{\max}$ then return $(\perp, I)$ 4 : Partition $[W]$ into $\Omega_\ell = [\Omega_\ell^- + 1, \Omega_\ell^+]$ s.t. $\Omega_\ell = w_\ell$ for $\ell \in I$ 5 : Define mappings $\ell[i] := \ell$ s.t. $i \in \Omega_\ell$ and $j[i] := i - \Omega_\ell^-[i] \in [w_{\ell[i]}]$ 6 : $\text{pk}_{\text{agg}} = \llbracket z_0 \rrbracket_1 + \sum_{i \in [W]} \psi_{\ell[i], i, j[i], 1}$ 7 : // $= \llbracket \sum_{i \in [W]} c^i (t_i + \alpha_{\ell[i], j[i]}) + \sum_{i \in [W+1, \widehat{W}]} c^i t_i \rrbracket_1$ 8 : $\text{ek} \leftarrow (W, \text{pk}_{\text{agg}}, \llbracket c^{\widehat{W}+1} t \rrbracket_T, (y_m)_{m \in [n]})$ 9 : for $i \in [\widehat{W}]$: 10 : $\gamma_i \leftarrow \gamma_{i,0} + \sum_{i' \in [W] \setminus \{i\}} \psi_{\ell[i'], \widehat{W}+1+i'-i, j[i'], 2}$ // $= \llbracket \sum_{i' \in [W] \setminus \{i\}} c^{\widehat{W}+1-i+i'} t_{i'} \rrbracket_2 +$ $\llbracket \sum_{i' \in [W] \setminus \{i\}} c^{\widehat{W}+1-i+i'} \alpha_{\ell[i'], j[i'], 2} \rrbracket_2$ 11 : $\text{dk} \leftarrow (\text{ek}, I, (\text{pk}_\ell, \Omega_\ell)_{\ell \in I}, (\llbracket c^i \rrbracket_2)_{i \in [\widehat{W}]})$ 12 : $(\gamma_i)_{i \in [\widehat{W}]}, (v_{m,i})_{m \in [n], i \in [\widehat{W}] \setminus X_m}$ 13 : $(\underbrace{\chi_{\ell[i], j[i], k}}_{\delta_{i,k} = \llbracket c^{\widehat{W}+1} \alpha_{\ell[i], j[i]} s^k_{\ell[i]} \rrbracket_2})_{i \in [W], k \in \mathfrak{B}})$ 14 : return $((\text{ek}, \text{dk}), I)$ </pre> Alg. VerKey(pk_ℓ): <pre> 1 : return $(\forall k \in \mathfrak{B} \setminus \{0\} : \xi_{\ell,k,1} \neq \llbracket 0 \rrbracket_1) \wedge \llbracket s \neq 0 \rrbracket$ 2 : $(\forall k \in \mathfrak{B} \cup \{0\} : \xi_{\ell,k,1} \circ \llbracket 1 \rrbracket_2 = \llbracket 1 \rrbracket_1 \circ \xi_{\ell,k,2}) \wedge$ 3 : $(\xi_{\ell,0,1} = \llbracket 1 \rrbracket_1) \wedge$ 4 : $(\forall k \in [-B_{\max}, B_{\max} - 1] :$ 5 : $\xi_{\ell,1,1} \circ \xi_{\ell,k,2} = \llbracket 1 \rrbracket_1 \circ \xi_{\ell,k+1,2})$ // $\xi_{\ell,k,*} = [s_\ell^k]$ 6 : $(\forall j \in [\widehat{w}_\ell] : \varphi_{\ell,j,1} \circ \llbracket 1 \rrbracket_2 = \llbracket 1 \rrbracket_1 \circ \varphi_{\ell,j,2} \wedge$ // $\mathbb{G}_1, \mathbb{G}_2$ consistent 7 : $(\forall i \in \mathfrak{W} : \psi_{\ell,i,j,*} = [c^i \alpha_{\ell,j}])$ 8 : $\psi_{\ell,i,j,1} \circ \llbracket 1 \rrbracket_2 = \llbracket 1 \rrbracket_1 \circ \psi_{\ell,i,j,2} \wedge$ 9 : $\varphi_{\ell,j,1} \circ \llbracket c^i \rrbracket_2 = \llbracket 1 \rrbracket_1 \circ \psi_{\ell,i,j,2} \wedge$ 10 : $(\forall k \in \mathfrak{B} : \chi_{\ell,j,k} = \llbracket c^{W_{\max}+1} \alpha_{\ell,j} s^k \rrbracket_2)$ 11 : $\xi_{\ell,-k,1} \circ \chi_{\ell,j,k} = \varphi_{\ell,j,1} \circ \llbracket c^{\widehat{W}+1} \rrbracket_2) \wedge$ 12 : $\Pi_{\text{pk}}.\text{Ver}(\mathbb{X}_{\text{pk}} = (\llbracket 1 \rrbracket_1, \text{pk}'_\ell, \widehat{w}_\ell), \pi_{\text{pk}}) = 1$ </pre>
---	--

Fig. 7. The key generation and key aggregation algorithms for the flexible-threshold silent weighted scheme SWBTX adapted from fSWBTX. The NIZKs Π_{pk} and Π_{ct} are highlighted. The encryption and decryption algorithms are given in Figure 8. For convenience, denote $\mathfrak{W} = [2\widehat{W}] \setminus \{\widehat{W} + 1\}$ and $\mathfrak{B} = [-B_{\max}, B_{\max}] \setminus \{0\}$

Alg. Enc(ek, μ, T): <pre> 1 : if $T > W$ then return $\perp$ 2 : Let $b_1, \dots, b_n \in \{0, 1\}$ s.t. $\sum_{m \in [n]} b_m 2^{m-1} = W_{\max} - W + T - 1$ 3 : $r \leftarrow \mathbb{Z}_p$ 4 : $\widehat{\text{pk}}_{\text{agg}} \leftarrow \text{pk}_{\text{agg}} + \sum_{m \in [n]} b_m y_m$ 5 : $\text{ct}' \leftarrow (\llbracket r \rrbracket_1, r \cdot \widehat{\text{pk}}_{\text{agg}}, \llbracket r c^{\widehat{W}+1} t \rrbracket_{\mathbb{T}} + \mu)$ 6 : $\mathbb{x}_{\text{ct}} = (W_{\max}, W, T, \llbracket 1 \rrbracket_1, \text{ek}, \text{ct}')$, $\mathbb{w}_{\text{ct}} = r$ 7 : $\pi_{\text{ct}} \leftarrow \Pi_{\text{ct}}.\text{Prove}(\text{pp}_{\text{ct}}, \mathbb{x}_{\text{ct}}, \mathbb{w}_{\text{ct}})$ 8 : return $\text{ct} = (\text{ct}', \pi_{\text{ct}}, T)$ </pre>	Alg. PartDec(ek, sk = s, $(\text{ct}_k = (\text{ct}_{k,1}, \text{ct}_{k,2}, \text{ct}_{k,3}, \pi_{\text{ct},k}, T))_{k \in [B]}$): <pre> 1 : for $k \in [B]$: $\mathbb{x}_k \leftarrow (W_{\max}, W, T, \llbracket 1 \rrbracket_1, \text{ek}, \text{ct}'_k = (\text{ct}_{k,1}, \text{ct}_{k,2}, \text{ct}_{k,3}))$ 2 : $U \leftarrow [B]$; $U \leftarrow \{k \in [B] : \Pi_{\text{ct}}.\text{Ver}(\mathbb{x}_k, \pi_{\text{ct},k}) = 1\}$ 3 : $\text{pd} \leftarrow \sum_{k \in U} s^k \cdot \text{ct}_{k,1} \quad \parallel = \llbracket \sum_{k \in U} r_k s^k \rrbracket_1$ 4 : return pd </pre>
	Alg. PartVerify(ek, pk_ℓ, pd_ℓ, $(\text{ct}_k)_{k \in [B]}$): $\parallel B \leq B_{\max}$ <pre> 1 : for $k \in [B]$: $\mathbb{x}_k \leftarrow (W_{\max}, W, T, \llbracket 1 \rrbracket_1, \text{ek}, \text{ct}'_k = (\text{ct}_{k,1}, \text{ct}_{k,2}, \text{ct}_{k,3}))$ 2 : $U \leftarrow [B]$; $U \leftarrow \{k \in [B] : \Pi_{\text{ct}}.\text{Ver}(\mathbb{x}_k, \pi_{\text{ct},k}) = 1\}$ 3 : return $\text{pd}_\ell \circ \llbracket 1 \rrbracket_2 = \sum_{k \in U} \text{ct}_{k,1} \circ \xi_{\ell,k,2}$ </pre>
Alg. Dec(dk, $\{\text{pd}_\ell\}_{\ell \in S}, (\text{ct}_k)_{k \in [B]}$): <pre> 1 : assert all ct_k have the same threshold T. 2 : for $k \in [B]$: $\mathbb{x}_k \leftarrow (W_{\max}, W, T, \llbracket 1 \rrbracket_1, \text{ek}, (\text{ct}_{k,1}, \text{ct}_{k,2}, \text{ct}_{k,3}))$ 3 : $U \leftarrow [B]$; $U \leftarrow \{k \in [B] : \Pi_{\text{ct}}.\text{Ver}(\mathbb{x}_k, \pi_{\text{ct},k}) = 1\}$ 4 : $S' \leftarrow \{\ell \in S : \text{PartVerify}(\text{ek}, \text{pk}_\ell, \text{pd}_\ell, (\text{ct}_k)_{k \in [B]}) = 1\}$ 5 : $W \leftarrow \sum_{\ell \in I} \Omega_\ell$; $W_{S'} \leftarrow \sum_{\ell \in S'} \Omega_\ell$ 6 : Let $b_1, \dots, b_n \in \{0, 1\}$ s.t. $\sum_{m \in [n]} b_m 2^{m-1} = W_{\max} - W + T - 1$ 7 : if $S' \not\subseteq I \vee W_{S'} < T$ then return $\perp$ 8 : $S_{\text{pad}} \leftarrow [W + 1, W_{\max}] \cup \bigcup_{m: b_m = 0} X_m$ 9 : Let $\widehat{\omega}$ be reconstruction coeff. for $\bigcup_{\ell \in S'} \Omega_\ell \cup S_{\text{pad}} \subseteq [\widehat{W}]$ 10 : for $k \in U$: 11 : $\parallel$ Step 1: Recover $\llbracket r_k c^{\widehat{W}+1} \alpha_{\ell[i],j[i]} \rrbracket_{\mathbb{T}}$ 12 : for $\ell \in S', i \in \Omega_\ell$: $E_{k,i} \leftarrow \text{pd}_\ell \circ \delta_{i,-k} - \sum_{k' \in U \setminus \{k\}} \text{ct}_{k',1} \circ \delta_{i,k'-k} \quad \parallel = \llbracket r_k c^{\widehat{W}+1} \alpha_{\ell[i],j[i]} \rrbracket_{\mathbb{T}}$ 13 : $\parallel$ Step 2: Recover for $\llbracket r_k c^{\widehat{W}+1} (t_i + \alpha_{\ell[i],j[i]}) \rrbracket_{\mathbb{T}}$ 14 : for $\ell \in S', i \in \Omega_\ell$: 15 : $A_{k,i} \leftarrow \text{ct}_{k,2} \circ \llbracket c^{\widehat{W}+1-i} \rrbracket_2 - \text{ct}_{k,1} \circ (\gamma_i + \sum_{m: b_m = 1} v_{m,i}) \quad \parallel = \llbracket r_k c^{\widehat{W}+1} (t_i + \alpha_{\ell[i],j[i]}) \rrbracket_{\mathbb{T}}$ 16 : $\parallel$ Step 3: Recover $\llbracket r_k c^{\widehat{W}+1} t_i \rrbracket_{\mathbb{T}}$ for dummy parties 17 : for $i \in S_{\text{pad}}$: $A_{k,i} \leftarrow \text{ct}_{k,2} \circ \llbracket c^{\widehat{W}+1-i} \rrbracket_2 - \text{ct}_{k,1} \circ (\gamma_i + \sum_{m: b_m = 1} v_{m,i}) \quad \parallel A_{k,i} = \llbracket r_k c^{\widehat{W}+1} t_i \rrbracket_{\mathbb{T}}$ 18 : $\parallel$ Step 4: Reconstruction over virtual shares 19 : $B_k \leftarrow \sum_{\ell \in S', i \in \Omega_\ell} \omega_i (A_{k,i} - E_{k,i}) + \sum_{i \in S_{\text{pad}}} \omega_i A_{k,i} \quad \parallel = \llbracket r_k c^{\widehat{W}+1} t \rrbracket_{\mathbb{T}}$ 20 : $\mu_k \leftarrow \text{ct}_{k,3} - B_k$ 21 : for $k \in [B] \setminus U$: $\mu_k \leftarrow \perp$ 22 : return $(\mu_k)_{k \in [B]}$ </pre>	

Fig. 8. Encryption and decryption algorithms for SWBTX.

Public parameters. $1\mathbb{G}_{\mathbb{T}}$ element, $(2\widehat{W} + n + 1)\mathbb{G}_1$ elements, and $\widehat{W}(n + 3)\mathbb{G}_2$ elements.

Public key pk_ℓ . $2B_{\max} + 4\hat{w}_\ell W_{\max} \mathbb{G}_1$, $2B_{\max}(\hat{w}_\ell + 1) + 4\hat{w}_\ell W_{\max} \mathbb{G}_2$ elements, and $(\hat{w}_\ell + 2)\mathbb{Z}_p$ elements.

Aggregated encryption key ek . $(n+1)\mathbb{G}_1$ and $1\mathbb{G}_T$ elements.

Ciphertext. $2\mathbb{G}_1$, $1\mathbb{G}_T$, and $2\mathbb{Z}_p$ elements.

Partial decryption. $1\mathbb{G}_1$ element.

Decryption runtime. Let S' be the set of parties whose decryption shares are included, $N' = |S'|$, w_ℓ be the weight for party $\ell \in S'$, and B be the batch size. Ignoring the verification, this requires the following operations:

- $W_{\max}$ MSMs of size $(n+1)$ and 1 forward FFT in $\mathbb{G}_1$ can be preprocessed.
- 1 MSM in $\mathbb{G}_2$ of size $W_{\max}$
- 1 MSM in $\mathbb{G}_2$ of size at most $W_{\max}$.
- For each $\ell \in S'$, B MSMs in $\mathbb{G}_2$ of size w_ℓ .
- B multi-pairing evaluations of size N' .
- $2B$ MSMs in $\mathbb{G}_2$ of size $\sum_{\ell \in S'} w_\ell$.
- 1 forward FFT in $\mathbb{G}_2$ of size $2B$, 1 inverse FFT in $\mathbb{G}_T$ of size $2B$.
- $4B$ pairings and $3B$ $\mathbb{G}_T$ additions.

Proof. We now give the proof considering the sizes and the decryption running time.

Sizes. The public parameters, the individual public key, aggregated encryption key, and ciphertext sizes follow from inspection. Note that pp_{pk} and pp_{ct} are empty in our instantiations. The NIZK proofs π_{pk} and π_{ct} are of $(2 + \hat{w})\mathbb{Z}_p$ and $2\mathbb{Z}_p$ sizes, respectively.

Decryption runtime. Now, we analyze the running time of the decryption. Similarly to the fixed threshold scheme, we will consider the final step of recovering the one-time pad term B_k in each of the $k \in U$ (see Line 19 of the algorithm Dec in Figure 8). Here, we have that

$$\begin{aligned}
B_k &= \sum_{\ell \in S', i \in \Omega_\ell} \omega_i (A_{k,i} - E_{k,i}) + \sum_{i \in S_{\text{pad}}} \omega_i A_{k,i} \\
&= \sum_{\ell \in S', i \in \Omega_\ell} \omega_i \left(\text{ct}_{k,2} \circ \llbracket c^{\widehat{W}+1-i} \rrbracket_2 - \text{ct}_{k,1} \circ \left(\gamma_i + \sum_{m: b_m=1} v_{m,i} \right) \right. \\
&\quad \left. - \text{pd}_\ell \circ \delta_{i,-k} + \sum_{k' \in U \setminus \{k\}} \text{ct}_{k',1} \circ \delta_{i,k'-k} \right) \\
&\quad + \sum_{i \in S_{\text{pad}}} \omega_i \left(\text{ct}_{k,2} \circ \llbracket c^{\widehat{W}+1-i} \rrbracket_2 - \text{ct}_{k,1} \circ \left(\gamma_i + \sum_{m: b_m=1} v_{m,i} \right) \right) \\
&= \sum_{i \in \bigcup_{\ell \in S'} \Omega_\ell \cup S_{\text{pad}}} \omega_i \left(\text{ct}_{k,2} \circ \llbracket c^{\widehat{W}+1-i} \rrbracket_2 - \text{ct}_{k,1} \circ \left(\gamma_i + \sum_{m: b_m=1} v_{m,i} \right) \right) \\
&\quad - \sum_{\ell \in S', i \in \Omega_\ell} \omega_i \left(\text{pd}_\ell \circ \delta_{i,-k} - \sum_{k' \in U \setminus \{k\}} \text{ct}_{k',1} \circ \delta_{i,k'-k} \right)
\end{aligned}$$

$$\begin{aligned}
&= \text{ct}_{k,2} \circ \underbrace{\sum_{i \in \bigcup_{\ell \in S'} \Omega_\ell \cup S_{\text{pad}}} \omega_i \llbracket c^{\widehat{W}+1-i} \rrbracket_2}_{\text{identical for all } k} - \text{ct}_{k,1} \circ \underbrace{\sum_{i \in \bigcup_{\ell \in S'} \Omega_\ell \cup S_{\text{pad}}} \omega_i \left(\gamma_i + \sum_{m: b_m=1} v_{m,i} \right)}_{\text{identical for all } k} \\
&\quad - \sum_{\ell \in S'} \text{pd}_\ell \circ \left(\sum_{i \in \Omega_\ell} \omega_i \delta_{i,-k} \right) + \sum_{k' \in U \setminus \{k\}} \text{ct}_{k',1} \circ \underbrace{\sum_{\ell \in S', i \in \Omega_\ell} \omega_i \delta_{i,k'-k}}_{=: H_{k'-k}}. \quad (28)
\end{aligned}$$

We now explain how to compute each of the highlighted term in Equation (28) efficiently.

- $\sum_{i \in \bigcup_{\ell \in S'} \Omega_\ell \cup S_{\text{pad}}} \omega_i \llbracket c^{\widehat{W}+1-i} \rrbracket_2$: This term is shared by all $k \in U$, and we can compute it using one MSM in $\mathbb{G}_2$ of size $|\bigcup_{\ell \in S'} \Omega_\ell \cup S_{\text{pad}}| = W_{\text{max}}$, since we only need S_{pad} to be such that the number of combined virtual shares are at least W_{max} .
- $\sum_{i \in \bigcup_{\ell \in S'} \Omega_\ell \cup S_{\text{pad}}} \omega_i \left(\gamma_i + \sum_{m: b_m=1} v_{m,i} \right)$: This term is also shared by all $k \in U$, and we can compute it using one MSM in $\mathbb{G}_2$ of size at most $W_{\text{max}}(n+1)$ as there are at most $W_{\text{max}}(n+1)$ many terms in the summation.
- $\sum_{\ell \in S'} \text{pd}_\ell \circ \left(\sum_{i \in \Omega_\ell} \omega_i \delta_{i,-k} \right)$: This term is not shared by any of $k \in U$, and has to be computed individually. This can be computed as follows: (1) Perform $|S'|$ MSMs in $\mathbb{G}_2$, each of size w_ℓ to compute $\sum_{i \in \Omega_\ell} \omega_i \delta_{i,-k}$ for each $\ell \in S'$, and (2) Perform 1 multi-pairing evaluation to compute the desired term.
- $\sum_{k' \in U \setminus \{k\}} \text{ct}_{k',1} \circ H_{k'-k}$: We can observe that this term is a convolution over $(\text{ct}_{k',1})_{k' \in U}$ and $(H_k)_{k \in [-B+1, B-1]}$ where $H_k = \sum_{\ell \in S', i \in \Omega_\ell} \omega_i \delta_{i,k}$ for $k \neq 0$ and $H_0 = \llbracket 0 \rrbracket_2$ (recall the same discussion in the proof of Theorem 5.1). Similar to the fixed threshold scheme, this can be computed using (1) $2B-2$ MSMs in $\mathbb{G}_2$ of size $\sum_{\ell \in S'} w_\ell$ to compute $H_{k'-k}$, and (2) 1 FFT in each of the source group of size $2B$, and (3) 1 inverse FFT of size $2B$ in the target group. Note that the FFT in $\mathbb{G}_1$ can be precomputed before knowing the decryption quorum.

Combining everything, we have that the decryption requires the following operations, matching the theorem statement.

- 1 MSM in $\mathbb{G}_2$ of size at most W_{max}
- 1 MSM in $\mathbb{G}_2$ of size at most $W_{\text{max}}(n+1)$. Note that this can be decomposed into preprocessing W_{max} MSM in $\mathbb{G}_2$ of size $n+1$, and compute 1 MSM of size W_{max} in $\mathbb{G}_2$
- For each $\ell \in S'$, B MSMs in $\mathbb{G}_2$ of size w_ℓ . This amortizes to $O(WB)$ operations.
- B multi-pairing evaluations of size N' .
- $2B$ MSMs in $\mathbb{G}_2$ of size W .

- 1 forward FFT in $\mathbb{G}_1$ of size $2B$ that can be precomputed before receiving the partial decryption shares. 1 forward FFT in $\mathbb{G}_2$ of size $2B$. 1 inverse FFT in $\mathbb{G}_T$ of size $2B$.
- 2 pairings and 3 $\mathbb{G}_T$ additions at the end per each decrypted ciphertext, resulting in $2B$ pairings and $3B$ $\mathbb{G}_T$ additions.

This sums up to $O(B \log B + BW + W_{\max} \log W_{\max})$ group operations, and concludes the proof. $\square$

EXTENSIONS. Below, we discuss possible extensions and optimizations that can be made to the scheme in specific scenarios.

Remark G.2 (Compressing public keys). We note that one can employ similar ideas from compressing the public parameters to compress the public keys generated from **KeyGen** to be of size $O(W_{\max} \log \hat{w} + B_{\max} \hat{w})$ group elements. In particular, assuming that $\hat{w} = 2^h - 1$ for some integer $h \geq 1$, we can group the α_j 's into h blocks, each of size $1, 2, \dots, 2^{h-1}$. Then, instead of publishing $(\llbracket c^i \alpha_j \rrbracket_1, \llbracket c^i \alpha_j \rrbracket_2)_{i \in [2\hat{W}] \setminus \{\hat{W}+1\}, j \in [\hat{w}]}$, the key generation outputs for $m \in [h]$, denoting $X_m = [2^{m-1}, 2^m - 1]$,

$$\begin{aligned} y'_{m,i} &:= \sum_{j \in X_m} \alpha_j \llbracket c^{j+i} \rrbracket_1 && \text{for } i \in [W_{\max}] \\ v'_{m,i} &:= \sum_{j \in X_m : j+i \neq \hat{W}+1} \alpha_j \llbracket c^{j+i} \rrbracket_2 && \text{for } i \in [2\hat{W} - 2^{m-1}]. \end{aligned}$$

The elements $y'_{m,i}$ will be used to compute $\mathbf{pk}_{\text{agg}}$ in the encryption key, and the elements $v'_{m,i}$ will be used to compute the cross terms γ_i in $\mathbf{dk}$ and used during decryption. In particular, we need to still ensure that exactly w_ℓ values of α_j is incorporated into $\mathbf{pk}_{\text{agg}}$; this is achieved through bit decomposition. Note that this changes how the mappings $j[i]$ are computed in **Prep**, but it is still deterministic based on the weights w_ℓ .

Remark G.3 (Non-power-of-2 weights). As in [30], our scheme can also be extended to $W_{\max}$ that is not a power-of-two integer. We now briefly recall how it was done in the prior work. Let $n = \lceil \log W_{\max} \rceil$ and set $\hat{W} = 2W_{\max} - 1$, and for each $m \in [n]$, we define the blocks

$$X_m := \begin{cases} [W_{\max} + 2^{m-1}, W_{\max} + 2^m - 1] & \text{for } m \in [n-1] \\ [W_{\max} + 2^{n-1}, 2W_{\max} - 1] & \text{for } m = n \end{cases}.$$

The structural changes are made to the encryption and decryption algorithms: mainly on how the bits b_m for $m \in [n]$ are chosen. Given threshold T and total weight W of aggregated parties, we define b_m as follows:

- Set $b_n = 1$ if $W_{\max} - W + T - 1 \in [2^{n-1}, W_{\max} - 1]$, and 0 otherwise.
- Then, for $m < n$, if $b_m = 1$, set $b_1, \dots, b_{n-1}$ to be the bit decomposition of $W_{\max} - W + T - 1 - (W_{\max} - 2^{n-1}) = 2^{n-1} - W + T - 1 \in [0, 2^n - 1]$. Otherwise, if $b_n = 0$, set $b_1, \dots, b_{n-1}$ to be the bit decomposition of $W_{\max} - W + T - 1$.

With this, we still have that $\sum_{m \in [n]} b_m |X_m| = W_{\max} - W + T - 1$, meaning we have added enough dummy parties to the encryption key. Moreover, note that the bits $b_1, \dots, b_n$ are deterministically chosen based on $W_{\max}, W$ and T .

Remark G.4 (Fixing threshold at key aggregation time.). In some applications, one does not need to determine the threshold at encryption time but only at key aggregation time. For example in blockchains, once the set of validators for the current epoch is decided, we would set the threshold to be $W/3$ where W is the total weights of the validators. In this case, we can further shrink $|\mathbf{ek}|$ by pre-aggregating the group element $\widehat{\mathbf{pk}}_{\text{agg}}$ and put it in $\mathbf{ek}$, instead of the element being computed within Enc algorithm. This reduces $|\mathbf{ek}|$ from $O(\log W_{\max})$ elements to $O(1)$ elements.

G.2 Correctness and Security

We state a general theorem for the correctness and security below. The security proof follows similarly to the fixed threshold scheme, so we only give a proof sketch below.

CORRECTNESS. The scheme SWBTX is perfectly correct.

Theorem G.5 (Correctness.). *If Π_{pk} and Π_{ct} are perfectly complete, the scheme SWBTX is perfectly correct.*

Proof. The proof follows almost identically to the correctness proof of fSWBTX (Theorem 5.4) when the maximum weight is set to $\widehat{W} = 2W_{\max} - 1$ and the threshold is set to $W_{\max}$. The main difference is dealing with the extra terms in the public parameters. As alluded to in the overview above, we can treat the β_i 's in the public parameters as shares from additional dummy parties. The public parameter compression can be viewed as pre-computation of relevant elements in $\mathbf{ek}$ and $\mathbf{dk}$ that will later be output by Prep . $\square$

ROBUSTNESS. The following theorem then establishes perfect robustness of our SWBTX scheme.

Theorem G.6. *If Π_{pk} and Π_{ct} are perfectly complete, for any adversary $\mathcal{A}$ in the SBTE-ROB game of SWBTX, we have that $\text{Adv}_{\text{SWBTX}}^{\text{rob}}(\mathcal{A}, \lambda) = 0$.*

Proof. Similar to the correctness proof, the robustness proof follows almost identically to that of the fixed threshold scheme. The difference is again dealing with extra β_i terms in the public parameters. Mapping this to the robustness proof in Appendix D, these β_i terms will correspond to additional honest party's shares, while the entries $\alpha_i^* = 0$ in the aggregated public key $\mathbf{pk}_{\text{agg}}$ (see Equation (26)) will correspond to malicious parties whose keys are generated using α_j . The rest of the proof then follows similarly. $\square$

INDISTINGUISHABILITY. We show IND-CCA security of this flexible threshold scheme SWBTX in two steps. First, we show that the IND-CPA security of $\text{SWBTX}_{\text{cpa}}$ (denoted as the version without the NIZKs) follows from the IND-CPA security of the fixed threshold scheme $\text{fSWBTX}_{\text{cpa}}$. This is established in Theorem G.7. Then, we show the IND-CCA security of SWBTX follows from IND-CPA security of $\text{SWBTX}_{\text{cpa}}$ and simulation extractability of the two NIZKs Π_{pk} and Π_{ct} . This is established in Theorem G.8 and proved below.¹³

Theorem G.7. *If $\text{fSWBTX}_{\text{cpa}}$ is T-IND-CPA secure, then $\text{SWBTX}_{\text{cpa}}$ is T-IND-CPA secure.*

Although this security implication is intuitively simple with the flexible threshold scheme being built from the fixed threshold scheme directly, we note that there are subtle case distinctions due to the syntax changes from flexible and fixed threshold schemes that make the formal proof slightly complicated. However, the general idea is that our reduction can generate dummy parties, both honest (representing the β_i shares in pp) and corrupted (representing zero shares used in the aggregation). We refer to the formal proof in Section G.3.

Theorem G.8. *If $\text{SWBTX}_{\text{cpa}}$ is T-IND-CPA secure and Π_{pk} and Π_{ct} are simulation extractable, then SWBTX is T-IND-CCA secure.*

Proof. The IND-CPA to IND-CCA proof follows similar steps as in the fixed-threshold scheme. The security proof applies simulation extractability of the two proof systems to simulate the proofs π_{pk} that honest parties generate and π_{ct} that is generated with the challenge ciphertext. Then, the proof extracts from the maliciously chosen keys (within $\mathcal{O}_{\text{CUser}}$ queries) the underlying key generation randomness and from the maliciously chosen ciphertexts (from $\mathcal{O}_{\text{PDec}}$ queries). These extracted values allow the reduction to IND-CCA security to make corresponding queries to its own oracles with semi-honest keys and ciphertexts.

G.3 Proof of Theorem G.7

Let $\mathcal{A}$ be an adversary in T-IND-CPA $_{\text{fSWBTX}_{\text{cpa}}}$ game making at most $Q_{\text{PDec}} = Q_{\text{PDec}}(\lambda)$ queries to the partial decryption oracle and running in time $T_{\mathcal{A}} = T_{\mathcal{A}}(\lambda)$. Also, let $W_{\text{max}} = W_{\text{max}}(\lambda)$, $B_{\text{max}} = B_{\text{max}}(\lambda)$, $N = N(\lambda)$ be the maximum weight, batch size, and number of parties.

We will prove the theorem by constructing a reduction $\mathcal{B}$ defined below. At a high level, the reduction simply introduces dummy parties to the game, both honest and corrupted.

Now, consider the following reduction $\mathcal{B}$ playing the game T-IND-CPA $_{\text{SWBTX}_{\text{cpa}}}$ and running $\mathcal{A}$ as a subroutine:

¹³ We cannot directly prove the IND-CCA security of SWBTX directly from the IND-CCA security of its fixed threshold version due to the different relations being proved by the NIZK Π_{ct} .

Counters and states $\mathcal{B}$ keep track of counters N and N' for the number of parties registered in the game by $\mathcal{A}$ and by itself with the underlying game, respectively. Also, it keeps track of all the values initialized so far.

Public parameters As $\mathcal{A}$ outputs its parameter choices $(W_{\max} = 2^n, B_{\max})$ ¹⁴, $\mathcal{B}$ outputs to its game $(\widehat{W} = 2W_{\max} - 1, B_{\max}, \widehat{T} = W_{\max})$. Then, $\mathcal{B}$ receives the public parameters $\mathbf{pp}'$ of the form:

$$\mathbf{pp}' = (\widehat{W}, B_{\max}, \llbracket c^{\widehat{W}+1} t \rrbracket_{\top}, (\llbracket c^i \rrbracket_1, \llbracket c^i \rrbracket_2)_{i \in [2\widehat{W}]}, (\llbracket c^i t_j \rrbracket_1, \llbracket c^i t_j \rrbracket_2)_{i \in [2\widehat{W}] \setminus \{\widehat{W}+1\}, j \in [\widehat{W}]}).$$

Then, to set up the public parameters for $\mathcal{A}$, $\mathcal{B}$ does the following:

- Make n queries to $\mathcal{O}_{\text{HUser}}$ to obtain the honest public keys $\mathbf{pk}_\ell \leftarrow^s \mathcal{O}_{\text{HUser}}(\widehat{w}_\ell)$ for $\ell \in [n]$ and $\widehat{w}_\ell = 2^{\ell-1}$. Each of these are parsed as

$$\begin{aligned} \mathbf{pk}_\ell = & \left(\underbrace{(\llbracket s_\ell^k \rrbracket_1, \llbracket s_\ell^k \rrbracket_2)}_{\xi_{\ell,k,1}, \xi_{\ell,k,2}}_{k \in [-B_{\max}, B_{\max}]} \right. \\ & \left(\underbrace{(\llbracket \alpha_{\ell,j} \rrbracket_1, \llbracket \alpha_{\ell,j} \rrbracket_2)}_{\varphi_{\ell,j,1}, \varphi_{\ell,j,2}}, \underbrace{(\llbracket c^i \alpha_{\ell,j} \rrbracket_1, \llbracket c^i \alpha_{\ell,j} \rrbracket_2)}_{\psi_{\ell,i,j,1}, \psi_{\ell,i,j,2}} \right)_{i \in [2\widehat{W}] \setminus \{\widehat{W}+1\}}, \\ & \left. \underbrace{(\llbracket c^{\widehat{W}+1} \alpha_{\ell,j} s^k \rrbracket_2)}_{\chi_{\ell,j,k}}_{k \in [-B_{\max}, B_{\max}] \setminus \{0\}} \right)_{j \in [\widehat{w}_\ell]}. \end{aligned}$$

- Make $n + 1$ queries to $\mathcal{O}_{\text{CUser}}(\widehat{w}_\ell, \rho_\ell)$ for $\ell \in [n + 1, 2n + 1]$. Here, we define for each ℓ

$$\begin{aligned} \widehat{w}_\ell &= \begin{cases} W_{\max} & \ell = 2n + 1 \\ 2^{\ell-n} & \ell \in [n + 1, 2n] \end{cases} \\ \rho_\ell &= (s_\ell = 1, (\alpha_{\ell,j} = 0)_{j \in [\widehat{w}_\ell]}). \end{aligned}$$

These first two steps increase N' to $2n + 1$ but N is still 0. Note also that the public keys derived from these are of the form: for $\ell \in [n + 1, 2n + 1]$

$$\begin{aligned} \mathbf{pk}_\ell = & \left(\underbrace{(\llbracket 1 \rrbracket_1, \llbracket 1 \rrbracket_2)}_{\xi_{\ell,k,1}, \xi_{\ell,k,2}}_{k \in [-B_{\max}, B_{\max}]} \right. \\ & \left(\underbrace{(\llbracket 0 \rrbracket_1, \llbracket 0 \rrbracket_2)}_{\varphi_{\ell,j,1}, \varphi_{\ell,j,2}}, \underbrace{(\llbracket 0 \rrbracket_1, \llbracket 0 \rrbracket_2)}_{\psi_{\ell,i,j,1}, \psi_{\ell,i,j,2}} \right)_{i \in [2\widehat{W}] \setminus \{\widehat{W}+1\}}, \\ & \left. \underbrace{(\llbracket 0 \rrbracket_2)}_{\chi_{\ell,j,k}}_{k \in [-B_{\max}, B_{\max}] \setminus \{0\}} \right)_{j \in [\widehat{w}_\ell]}. \end{aligned}$$

- Define $X_m = [W_{\max} + 2^{m-1}, W_{\max} + 2^m - 1]$ for $m \in [n]$, which are partitionings X_m of $[W_{\max} + 1, \widehat{W}]$. Then, for each $m \in [n]$ and $j \in X_m$, treat β_j as $\alpha_{m,j-W_{\max}-2^{m-1}+1}$.

¹⁴ We keep the simplicity of the proof by assuming $W_{\max}$ is a power-of-2. Extending this to general integer $W_{\max}$ can be down using similar ideas from Remark G.3

- Then, compute $\mathbf{pp}$ from $\mathbf{pp}'$ as

$$\mathbf{pp} = (W_{\max}, B_{\max}, \llbracket c^{\widehat{W}+1} t \rrbracket_{\top}, (\llbracket c^i \rrbracket_1, \llbracket c^i \rrbracket_2)_{i \in [2\widehat{W}]}, \llbracket z_0 \rrbracket_1, (v_{i,0})_{i \in [\widehat{W}]}, (y_m, (\gamma_{m,j})_{j \in [\widehat{W}] \setminus X_m})_{m \in [n]})$$

where $\llbracket z_0 \rrbracket_1$, $\llbracket v_{i,0} \rrbracket_2$, y_m , and $\llbracket \gamma_{m,j} \rrbracket_2$ are defined as follows:

$$\begin{aligned} \llbracket z_0 \rrbracket_1 &= \sum_{i \in [\widehat{W}]} \llbracket c^i t_i \rrbracket_1 \\ \gamma_{i,0} &= \sum_{j \in [\widehat{W}] \setminus \{i\}} \llbracket c^{\widehat{W}+1+j-i} t_j \rrbracket_2 && \text{for } i \in [\widehat{W}] \\ y_m &= \sum_{i \in X_m} \beta_i \llbracket c^i \rrbracket_1 = \sum_{i \in X_m} \llbracket \alpha_{m,i-W_{\max}-2^{m-1}+1} c^i \rrbracket_1 && \text{for } m \in [n] \\ v_{m,i} &= \sum_{j \in X_m} \beta_j \llbracket c^{\widehat{W}+1+j-i} \rrbracket_2 \\ &= \sum_{j \in X_m} \llbracket \alpha_{m,j-W_{\max}-2^{m-1}+1} c^{\widehat{W}+1+j-i} \rrbracket_2 && \text{for } m \in [n], i \in [\widehat{W}] \setminus X_m \end{aligned}$$

- Forward $\mathbf{pp}$ to $\mathcal{A}$.

Oracle queries: The reduction $\mathcal{B}$ answers the queries from $\mathcal{A}$ as follows:

- $\mathcal{O}_{\text{HUser}}(\hat{w})$ **oracle:** Here, $\mathcal{B}$ queries the $\mathcal{O}_{\text{HUser}}(\hat{w})$ oracle of its own game to receive $\mathbf{pk}_{N'}$. Increment N and N' accordingly. Note that it always hold that $N' = N + 2n + 1$. Note that the ℓ -th key of $\mathcal{B}$ is the $(\ell - 2n - 1)$ -th key in the view of $\mathcal{A}$.
- $\mathcal{O}_{\text{CUser}}$ **oracle:** When $\mathcal{A}$ makes a query $(\hat{w}, \rho)$, increment N and N' set $\mathbf{pk}_{N'} \leftarrow \text{KeyGen}(\hat{w}; \rho)$ and make a query to $\mathcal{O}_{\text{CUser}}(\hat{w}, \rho)$ to register this key.
- **Challenge oracle:** On query $(J = (\ell_1, \dots, \ell_L) \in [N]^L, \mu_0, \mu_1, (w_\ell)_{\ell \in J}, T)$, $\mathcal{B}$ does the following:
 - Check that there are no duplicates $\ell_i = \ell_j$.
 - Run $((\mathbf{ek}, \mathbf{dk}), I' \subseteq [L]) \leftarrow \text{Prep}((\mathbf{pk}_{\ell_j+2n+1}, w_{\ell_j})_{j \in [L]})$. Check and abort according to the IND-CPA game.
 - Let $I^* = (\ell_j)_{j \in I'}$ and $W = \sum_{\ell \in I^*} w_\ell$. Then, let $b_1, \dots, b_n$ be the bit-decomposition of $W_{\max} - W + T - 1$.
 - Compute the following vector of indices $\hat{I} = ((\ell + 2n + 1)_{\ell \in I^*}, 2n + 1, (m + (1 - b_m) \cdot n)_{m \in [n]})$ with vector of weights $\vec{w}' = ((w_\ell)_{\ell \in I^*}, W_{\max} - W, (2^{m-1})_{m \in [n]})$.
 - Query $\mathbf{ct}^* \leftarrow \mathcal{O}_{\text{Chal}}(\hat{I}, \mu_0, \mu_1, \vec{w}')$ and return $\mathbf{ct}^*$ to $\mathcal{A}$.
- **Partial decryption oracle:** On the query $(\ell \in [N], \mathbf{ek}, (\mathbf{ct}_k, (\mu_k, \rho_k, T_k))_{k \in [B]})$, $\mathcal{B}$ does the following:
 - First, check whether $\mathbf{ct}_k = \text{Enc}(\mathbf{ek}, \mu_k, T_k; \rho_k)$ for each $k \in [B]$ where $\mathbf{ct}_k \neq \mathbf{ct}^*$. If this does not hold, return $\perp$.
 - Also, parse $\mathbf{ek} = (W, \mathbf{pk}'_{\text{agg}}, \mathbf{ek}_{\text{pad}}, (y'_m)_{m \in [n]})$.
 - Next, for each $k \in [B]$ where $\mathbf{ct}_k \neq \mathbf{ct}^*$,
 - * Let $b_1, \dots, b_n \in \{0, 1\}$ be such that $\sum_{m \in [n]} b_m 2^{m-1} = W_{\max} - W + T_k - 1$.

- * Let $\llbracket z_k \rrbracket_1 = \text{pk}'_{\text{agg}} + \sum_{m \in [n]} b_m y'_m$ and $\text{ek}' = (W, \llbracket z_k \rrbracket_1, \text{ek}_{\text{pad}})$
- * Make a query $\text{pd}'_{\ell, k} \leftarrow \mathcal{O}_{\text{PDec}}(\text{ek}', \ell + 2n + 1, (\text{ct}'_{k'}, \mu'_{k'}, \rho'_{k'})_{k' \in [B]})$ where $\mu'_{k'} = \llbracket 0 \rrbracket_{\text{T}}$ and $\rho'_{k'} = 0$ for $k' \neq k$, $\mu'_k = \mu_k$ and $\rho'_k = \rho_k$, and $\text{ct}'_{k'}$ is generated using the fixed threshold encryption algorithm with the key ek' on $(\mu'_{k'}, \rho'_{k'})$ pairs for all $k' \in [B]$.
- Then, let $U = \{k \in [B] : \text{ct}_k = \text{ct}^*\}$ and make a query $\text{pd}'_{\ell} \leftarrow \mathcal{O}_{\text{PDec}}(\text{ek}^*, \ell + 2n + 1, (\text{ct}'_{k'}, \mu'_{k'}, \rho'_{k'})_{k' \in [B]})$ where $\mu'_{k'} = \llbracket 0 \rrbracket_{\text{T}}$ and $\rho'_{k'} = 0$ for $k' \in [B]$, and $\text{ct}'_{k'} = \text{Enc}(\text{ek}^*, \mu'_{k'}, T^*; \rho'_{k'})$ for $k' \notin U$ while $\text{ct}'_{k'} = \text{ct}^*$ for $k' \in U$.
- Finally, return $\text{pd}_{\ell} = \text{pd}'_{\ell} + \sum_{k: \text{ct}_k \neq \text{ct}^*} \text{pd}'_{\ell, k}$.

We note that this involved partial decryption is to accommodate batches with different threshold for each ciphertext.

Guess bit: The reduction $\mathcal{B}$ simply returns the bit b' that $\mathcal{A}$ outputs.

There are two things we need to show: (1) that the view of $\mathcal{A}$ is identical to its view in the IND-CPA game of SWBTX, and (2) that the total weight of corrupted parties whose keys are aggregated into ek^* does not exceed the fixed threshold W_{max} .

For (1), we can first observe that the public parameters are identically distributed as the dummy parties introduced has uniformly random $\alpha_{\ell, j}$. Similarly, the honestly generated public keys have the same distribution in both fixed-threshold and flexible threshold schemes. Finally, we need to ensure that (1.1) the challenge ciphertext ct^* has the same distribution as the one computed in the game, and (1.2) the partial decryption is of the same format as in the game. For (1.1), Recall that the ciphertext structure remains the same except for the component $\llbracket \hat{z} \rrbracket_1$ which in the T-IND-CPA_{SWBTX} game is computed as follows. Let I^* be the list of indices in $[N]$ as defined in the reduction and $\ell[i], j[i]$ be the mappings defined as in the preprocessing algorithm that is run within $\mathcal{O}_{\text{Chal}}$.

$$\begin{aligned}
\widehat{\text{pk}}_{\text{agg}} &= \text{pk}_{\text{agg}} + \sum_{m \in [n]} b_m y_m \\
&= \underbrace{\sum_{i=1}^{\widehat{W}} \llbracket c^i t_i \rrbracket_1 + \sum_{\ell \in I^*, i \in \Omega_{\ell}} \psi_{\ell+2n+1, i, j[i], 1}}_{=\text{pk}_{\text{agg}}} + \sum_{m \in [n]: b_m=1} \underbrace{\sum_{i \in X_m} \llbracket c^i \beta_i \rrbracket_1}_{=y_m} \\
&= \sum_{i=1}^{\widehat{W}} \llbracket c^i t_i \rrbracket_1 + \underbrace{\sum_{\ell \in I^*, i \in \Omega_{\ell}} \psi_{\ell+2n+1, i, j[i], 1}}_{\text{only for } i \in [W]} + \sum_{m \in [n]: b_m=1} \sum_{i \in X_m} \llbracket c^i \alpha_{m, i-W_{\text{max}}-2^{m-1}+1} \rrbracket_1 \\
&\quad + \sum_{m \in [n]: b_m=0} \sum_{i \in X_m} 0 \cdot \llbracket c^i \rrbracket_1 + \sum_{i=W_{\text{max}}+1}^{W_{\text{max}}} 0 \cdot \llbracket c^i \rrbracket_1
\end{aligned}$$

$$\begin{aligned}
&= \sum_{i=1}^{\widehat{W}} \llbracket c^i t_i \rrbracket_1 + \underbrace{\sum_{\ell \in I^*, i \in \Omega_\ell} \psi_{\ell+2n+1, i, j[i], 1}}_{\text{only for } i \in [W]} + \sum_{m \in [n]: b_m = 1} \sum_{i \in X_m} \psi_{m, i, i - W_{\max} - 2^{m-1} + 1, 1} \\
&\quad + \sum_{m \in [n]: b_m = 0} \sum_{i \in X_m} \psi_{n+m, i, i - W_{\max} - 2^{m-1} + 1, 1} + \sum_{i=W+1}^{W_{\max}} \psi_{2n+1, i, i-W, 1}
\end{aligned}$$

The first equality follows from how $\widehat{\mathbf{pk}}_{\text{agg}}$ is defined. The second equality follows from how we set β_i within $\mathcal{B}$ and how y_m are defined. The third and fourth equalities follow from how $\mathbf{pp}$ and $\mathbf{pk}_\ell$ are defined by $\mathcal{B}$.

Finally, we can see that the value on the last line corresponds to how the keys will be aggregated in the $\mathcal{O}_{\text{Chal}}$ call made by $\mathcal{B}$. In particular, the reduction $\mathcal{B}$ makes the query to $\mathcal{O}_{\text{Chal}}$ with the party indices and weights $\hat{I} = ((\ell + 2n + 1)_{\ell \in I^*}, 2n+1, (m + (1-b_m) \cdot n)_{m \in [n]})$ and $\vec{w}' = ((w_\ell)_{\ell \in I^*}, W_{\max} - W, (2^{m-1})_{m \in [n]})$. We can see that

1. The first $|I^*|$ indices coincide with the second term in the sum.
2. The $(2n+1)$ -th party corresponds to the zero shares in the last summation term.
3. The $(m + (1-b_m) \cdot n)_{m \in [n]}$ will correspond to the dummy parties' shares when $b_m = 1$ and correspond to the zero shares when $b_m = 0$.

Thus, the encryption key used within the $\mathcal{O}_{\text{Chal}}$ made by $\mathcal{B}$ is identical to what the game would use, so the ciphertext is identically distributed.

For (1.2), we see that for each query to $\mathcal{O}_{\text{PDec}}$ made by $\mathcal{A}$, $\mathcal{B}$ makes individual queries to its own $\mathcal{O}_{\text{PDec}}$ for each slot. In particular, we can see that for each $k \in [B]$ with $\text{ct}_k \neq \text{ct}^*$,

$$\mathbf{pd}'_{\ell, k} = s_\ell^k \text{ct}_{k, 1} ,$$

where s_ℓ is the secret key of party $\ell \in [N]$. This is because, the query made by $\mathcal{B}$ to obtain $\mathbf{pd}'_{\ell, k}$ consists of all zero ciphertexts except for the k -th position in the batch. Similarly, we also have that

$$\mathbf{pd}'_\ell = \sum_{k \in [B]: \text{ct}_k = \text{ct}^*} s_\ell^k \text{ct}_1^* .$$

Therefore, the output $\mathbf{pd}_\ell = \sum_{k \in [B]} s_\ell^k \text{ct}_{k, 1}$ as in the T-IND-CPA game of SWBTX.

For (2) (total weight claim), assume that $\mathcal{A}$ does not corrupt parties with total weight $\sum_{\ell \in \mathcal{C} \cap I^*} w_\ell^* \geq T$. Then, with how the $\mathcal{O}_{\text{Chal}}$ call by $\mathcal{B}$ is made, only $(W_{\max} - W) + (W_{\max} - 1 - (W_{\max} - W + T - 1)) = W_{\max} - T$ corrupted weights are added to the aggregated key. Therefore, the total corrupted weight is $< W_{\max} - T + T = W_{\max}$, meaning the corruption does not exceed the threshold.

Therefore, we can see that

$$\text{Adv}_{\text{SWBTX}_{\text{cpa}}}^{\text{t-ind-cpa}}(\mathcal{A}, \lambda) \leq \text{Adv}_{\text{fSWBTX}_{\text{cpa}}}^{\text{t-ind-cpa}}(\mathcal{B}, \lambda) ,$$

concluding the proof. $\square$

H Instantiations for NIZKs

In this section, we give the constructions for the NIZK proof systems $\Pi_{\text{pk}}, \Pi_{\text{ct}}$ used in both constructions. We note that for all of the constructions, we will consider NIZK proofs derived from Fiat-Shamir transformation on Schnorr-like Σ -protocols for linear relations.

H.1 NIZK Constructions

We give the constructions of the proof systems Π_{pk} and Π_{ct} in Figure 9. The NIZK Π_{pk} is a proof system for the relation $\mathcal{R}_{\text{pk}}$ (Equation (14)). For Π_{ct} , we give two versions: one proving knowledge of a witness for $\mathcal{R}_{\text{ct}}$ (Equation (15)) and one for $\mathcal{R}'_{\text{ct}}$ (Equation (27)). The proof systems follow the general structure of Fiat-Shamir compiled Schnorr-like protocol that first computes a random nonce $\vec{A}$, hashes it along with the whole statement to obtain the challenge c , and finally computes the response $\vec{\zeta}$. Note that these proof systems have empty setup algorithms that return $\perp$. Also, we present the proofs with the first message $\vec{A}$, but they can be omitted as they can be reconstructed from c and $\vec{\zeta}$.

The following theorems then establish the required properties of Π_{pk} and Π_{ct} .

Theorem H.1. *The NIZK Π_{pk} for the relation $\mathcal{R}_{\text{pk}}$ (Equation (14)) is perfectly correct and simulation extractable assuming the generic group model and the random oracle model.*

Theorem H.2. *The NIZK Π_{ct} for the relation $\mathcal{R}_{\text{ct}}$ (Equation (15)) and $\mathcal{R}'_{\text{ct}}$ (Equation (27)) is perfectly correct and simulation extractable assuming the generic group model and the random oracle model.*

H.2 Proof of Theorem H.1

CORRECTNESS. Perfect correctness follows simply by inspection.

SIMULATION EXTRACTABILITY. We now prove simulation extractability in the GGM and ROM. First, we define the following stateful simulator $\text{Sim} = (\text{Sim.Setup}, \text{Sim.Sim}, \text{Sim.Ext})$ with access to $\mathcal{A}$'s random oracle and generic group oracle queries. Without loss of generality, assume that any labels for group elements output by $\mathcal{A}$ correspond to a label that was an output of the group operation oracles.

- $\text{Sim.Setup}(1^\lambda)$: returns $(\perp, \perp)$.
- $\text{Sim.Sim}^{\text{H}_{\text{pk}}}(\text{st}_{\text{Sim}}, \mathbb{x})$: Parse the statement as in $\Pi_{\text{pk}}.\text{Prove}$. Let $\hat{w}$ be the integer parsed out from the statement, and $\vec{X}$ be as in $\Pi_{\text{pk}}.\text{Ver}$. Then, sample $c \leftarrow \mathbb{Z}_p, \vec{\zeta} \leftarrow \mathbb{Z}_p^{\hat{w}+1}$, set $\vec{A} = \llbracket \vec{\zeta} \rrbracket_1 - c \cdot \vec{X}$, program $\text{H}_{\text{pk}}(\mathbb{x}, \vec{A}) \leftarrow c$. Finally, return $\pi_{\text{pk}} = (\vec{A}, c, \vec{\zeta})$.
- $\text{Sim.Ext}(\text{st}_{\text{Sim}}, \mathbb{x}, \pi)$: Parse the statement and set $\vec{X} = (\xi_{1,1}, \varphi_{1,1}, \dots, \varphi_{\hat{w},1})$ as in $\Pi.\text{Ver}$. The simulator checks the corresponding scalars $(s, \alpha_1, \dots, \alpha_{\hat{w}})$ underlying these labels in the GGM. Then, it returns the witness $(s, \alpha_1, \dots, \alpha_{\hat{w}})$.

Alg. $\Pi_{pk}.Prove^{H_{pk}}(\mathbb{x}, w)$ <pre> 1 : parse $\mathbb{x} = (\llbracket 1 \rrbracket_1, pk, \hat{w})$ 2 : Parse pk as in Figure 4 3 : $a_s, a_1, \dots, a_{\hat{w}} \leftarrow \mathbb{Z}_p$ 4 : $\vec{A} \leftarrow (\llbracket a_s \rrbracket_1, \llbracket a_1 \rrbracket_1, \dots, \llbracket a_{\hat{w}} \rrbracket_1)$ 5 : $c \leftarrow H_{pk}(\mathbb{x}, \vec{A})$ 6 : $\vec{\zeta} \leftarrow (a_s, a_1, \dots, a_{\hat{w}}) + c \cdot w$ 7 : return $\pi_{pk} = (\vec{A}, c, \vec{\zeta})$ </pre> Alg. $\Pi_{ct}.Prove^{H_{ct}}(\mathbb{x}, w)$ <pre> 1 : parse $\mathbb{x} = (\llbracket 1 \rrbracket_1, (pk_{agg}, ek_{pad}), (ct_1, ct_2, ct_3))$ 2 : parse $\mathbb{x} = (W_{max}, W, T, \llbracket 1 \rrbracket_1, ek, (ct_1, ct_2, ct_3))$ 3 : Let $b_1, \dots, b_n \in \{0, 1\}$ s.t. 4 : $W_{max} - W + T - 1 = \sum_{m=1}^n b_m 2^{m-1}$ 5 : $\widehat{pk}_{agg} \leftarrow pk_{agg} + \sum_{m=1}^n b_m y_m$ 6 : parse $w = (r)$ 7 : $a \leftarrow \mathbb{Z}_p$ 8 : $\vec{A} \leftarrow (\llbracket a \rrbracket_1, a\widehat{pk}_{agg})$ 9 : $c \leftarrow H_{ct}(\mathbb{x}, \vec{A})$ 10 : $\zeta \leftarrow a + c \cdot r$ 11 : return $\pi_{ct} = (\vec{A}, c, \zeta)$ </pre>	Alg. $\Pi_{pk}.Ver^{H_{pk}}(\mathbb{x}, \pi_{pk})$ <pre> 1 : parse $\mathbb{x} = (\llbracket 1 \rrbracket_1, pk, \hat{w})$ 2 : Parse pk as in Figure 4 3 : parse $\pi_{pk} = (\vec{A}, c, \vec{\zeta})$ 4 : $\vec{X} \leftarrow (\xi_{1,1}, \varphi_{1,1}, \dots, \varphi_{\hat{w},1})$ 5 : return $c = H_{pk}(\mathbb{x}, \vec{A}) \wedge \vec{A} + c\vec{X} = \llbracket \vec{\zeta} \rrbracket_1$ </pre> Alg. $\Pi_{ct}.Ver^{H_{ct}}(\mathbb{x}, \pi_{ct})$ <pre> 1 : parse $\mathbb{x} = (\llbracket 1 \rrbracket_1, (pk_{agg}, ek_{pad}), (ct_1, ct_2, ct_3))$ 2 : parse $\mathbb{x} = (W_{max}, W, T, \llbracket 1 \rrbracket_1, ek, (ct_1, ct_2, ct_3))$ 3 : Let $b_1, \dots, b_n \in \{0, 1\}$ s.t. 4 : $W_{max} - W + T - 1 = \sum_{m=1}^n b_m 2^{m-1}$ 5 : $\widehat{pk}_{agg} \leftarrow pk_{agg} + \sum_{m=1}^n b_m y_m$ 6 : parse $\pi_{ct} = (\vec{A}, c, \zeta)$ 7 : $\vec{X} \leftarrow (ct_1, ct_2)$ 8 : return $c = H_{ct}(\mathbb{x}, \vec{A}) \wedge \vec{A} + c\vec{X} = \zeta(\llbracket 1 \rrbracket_1, \widehat{pk}_{agg})$ </pre>
---	--

Fig. 9. NIZK proof systems Π_{pk} and Π_{ct} . We denote the distinctions for the two versions of Π_{ct} for **fixed** and **flexible** threshold schemes via the highlighting.

Consider an adversary $\mathcal{A}$ making Q_{Prove} and Q_{Ver} queries to the oracles $\mathcal{O}_{\text{Prove}}$ and $\mathcal{O}_{\text{Ver}}$, respectively. Also, $\mathcal{A}$ is a generic adversary making at most Q queries to the group operation oracles in the GGM and it makes at most Q_{H} queries to the random oracle H_{pk} . We now consider the following sequence of games.

Game G_0 : This is the game where $\mathcal{A}$ has access to the oracles $\mathcal{O}_{\text{Prove},0}$ and $\mathcal{O}_{\text{Ver},0}$.

Game G_1 : In this game, for any query to $\mathcal{O}_{\text{Prove},0}$, the game aborts if the hash query $H_{\text{pk}}(\mathbf{x}, \vec{A})$ to be made during proof generation has already been made either in a prior oracle query or by $\mathcal{A}$.

The view of $\mathcal{A}$ only changes if this abort occurs. We can analyze this bad event via the union bound. Consider the i -th query to $\mathcal{O}_{\text{Prove},0}$, we have that $\vec{A}$ that is in the hash query is independent from any of the prior queries made by $\mathcal{A}$ or within $\mathcal{O}_{\text{Prove},0}$. Moreover, $\vec{A}$ is uniformly random in $G_1^{\hat{w}+1}$, so we have that in the i -th query, the probability of this abort occurring is at most $\frac{Q_{\text{H}}+i}{p} \leq \frac{Q_{\text{H}}+Q_{\text{Prove}}}{p}$. Via the union bound, we have that

$$|\Pr[G_0^{\mathcal{A}} \Rightarrow 1] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1]| \leq \frac{(Q_{\text{H}} + Q_{\text{Prove}})Q_{\text{Prove}}}{p}.$$

Game G_2 : In this game, for each $\mathcal{O}_{\text{Prove}}$ query, given that the abort does not occur, we first sample $c \leftarrow \mathbb{Z}_p$ and program $H_{\text{pk}}(\mathbf{x}, \vec{A}) \leftarrow c$ instead. Due to the abort, this does not change the view of $\mathcal{A}$, so

$$\Pr[G_1^{\mathcal{A}} \Rightarrow 1] = \Pr[G_2^{\mathcal{A}} \Rightarrow 1]$$

Game G_3 : For each $\mathcal{O}_{\text{Prove}}$ query, the game now first samples $\vec{\zeta}$ and c and computes $\vec{A}$ as in `Sim.Sim`. Then, it checks for the introduced abort. Finally, program H_{pk} as in the prior game. Since $(\vec{A}, c, \vec{\zeta})$ still has the same distribution as $\mathbf{x}$ is in the language by how the game is defined, we have that

$$\Pr[G_2^{\mathcal{A}} \Rightarrow 1] = \Pr[G_3^{\mathcal{A}} \Rightarrow 1]$$

Game G_4 : In this game, we remove the introduced abort in $\mathcal{O}_{\text{Prove}}$. The $\mathcal{O}_{\text{Prove}}$ oracle in this game is now equivalent to $\mathcal{O}_{\text{Prove},1}$. Via similar argument as in G_1 , we have

$$|\Pr[G_3^{\mathcal{A}} \Rightarrow 1] - \Pr[G_4^{\mathcal{A}} \Rightarrow 1]| \leq \frac{(Q_{\text{H}} + Q_{\text{Prove}})Q_{\text{Prove}}}{p}.$$

Game G_5 : In this game, the verification oracle is now implemented with $\mathcal{O}_{\text{Ver},1}$. In particular, the game checks if $(\mathbf{x}, \pi)$ pair corresponds to any prior $\mathcal{O}_{\text{Prove}}$ query. Then, we use the GGM states to extract the underlying witnesses from the statements as in `Sim.Ext`, and check whether the extracted witness is valid for the relation. This game is now identical to the $b = 1$ case in the definition of simulation extractability.

We claim that

$$\Pr[G_4^{\mathcal{A}} \Rightarrow 1] = \Pr[G_5^{\mathcal{A}} \Rightarrow 1].$$

By how the labelling oracle is implemented in the GGM and our assumption that any label outputted by the adversary corresponds to a registered label in the model, the extracted underlying scalar corresponds directly to the

discrete logarithm of that element with respect to $\llbracket 1 \rrbracket_1$. Moreover, we note that any statement of the form that we consider has an underlying witness as $(s, \alpha_1, \dots, \alpha_{\widehat{w}}) \mapsto (\llbracket s \rrbracket_1, \llbracket \alpha_1 \rrbracket_1, \dots, \llbracket \alpha_{\widehat{w}} \rrbracket_1)$ is a bijection. Thus, it cannot be the case that the output of $\mathcal{O}_{\text{Ver},0}$ and $\mathcal{O}_{\text{Ver},1}$ differs. $\square$

H.3 Proof of Theorem H.2

CORRECTNESS. Correctness follows simply by inspection.

SIMULATION EXTRACTABILITY. For simulation extractability, we proceed similarly as in the proof of Theorem H.1 with the only differences being the simulator definition and the final argument in $\mathbf{G}_5$.

First, consider the following simulator:

- $\text{Sim.Setup}(1^\lambda)$: returns $(\perp, \perp)$.
- $\text{Sim.Sim}^{\text{H}_{\text{ct}}}(\text{st}_{\text{Sim}}, \mathbb{X})$: Parse the statement as in $\Pi_{\text{ct}}.\text{Prove}$ (the key distinctions in the two versions of Π_{ct} is how $\widehat{\text{pk}}_{\text{agg}}$ is computed). Let $\vec{X} = (\text{ct}_1, \text{ct}_2)$ be as in $\Pi_{\text{ct}}.\text{Ver}$. Then, sample $c \leftarrow \mathbb{Z}_p$, $\zeta \leftarrow \mathbb{Z}_p^1$, set $\vec{A} = (\llbracket \zeta \rrbracket_1, \zeta \widehat{\text{pk}}_{\text{agg}} - c \cdot \vec{X}$, program $\text{H}_{\text{ct}}(\mathbb{X}, \vec{A}) \leftarrow c$. Finally, return $\pi_{\text{ct}} = (\vec{A}, c, \zeta)$.
- $\text{Sim.Ext}(\text{st}_{\text{Sim}}, \mathbb{X}, \pi)$: Parse the statement and set $\vec{X} = (\text{ct}_1, \text{ct}_2)$ as in $\Pi.\text{Ver}$. The simulator checks the corresponding scalars r underlying the label of ct_1 in the GGM. Then, it returns the witness r .

We consider an adversary $\mathcal{A}$ with the same query complexity as in the prior section. We also consider a similar sequence of games $\mathbf{G}_0$ to $\mathbf{G}_5$ briefly described below drawing the parallels to the prior proof.

Game $\mathbf{G}_0$: This is the game where $\mathcal{A}$ gets access to $\mathcal{O}_{\text{Prove},0}$ and $\mathcal{O}_{\text{Ver},0}$.

Game $\mathbf{G}_1$: In this game, an abort is introduced in $\mathcal{O}_{\text{Prove}}$ when the to-be-made H_{ct} query was already made either by the game or by $\mathcal{A}$.

Game $\mathbf{G}_2$: In this game, we sample c first and program H_{ct} later.

Game $\mathbf{G}_3$: In this game, the game samples $\zeta, c \leftarrow \mathbb{Z}_p$ and set $\vec{A}$ accordingly, and program H_{ct} .

Game $\mathbf{G}_4$: In this game, we remove the abort introduced in $\mathbf{G}_1$. The $\mathcal{O}_{\text{Prove}}$ oracle is now identical to $\mathcal{O}_{\text{Prove},1}$.

Game $\mathbf{G}_5$: In this game, we replace $\mathcal{O}_{\text{Ver},0}$ with $\mathcal{O}_{\text{Ver},1}$, i.e., the game now verifies the proof, checks if the statement and proof pair corresponds to prior ones, and runs the extractor to obtain the witness.

With the same arguments, we can show that the transition from $\mathbf{G}_0$ to $\mathbf{G}_4$ has the following bound

$$|\Pr[\mathbf{G}_0^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{G}_4^{\mathcal{A}} \Rightarrow 1]| \leq \frac{2(Q_{\text{H}} + Q_{\text{Prove}})Q_{\text{Prove}}}{p}.$$

Now, we argue the transition from $\mathbf{G}_4$ to $\mathbf{G}_5$. Note that the only difference between the two games is the event that $\Pi_{\text{ct}}.\text{Ver}(\mathbb{X}, \pi) = 1$ but the extracted witness $w \leftarrow \text{Sim.Ext}(\text{st}_{\text{Sim}}, \mathbb{X}, \pi)$ is such that $(\mathbb{X}, w) \notin \mathcal{R}_{\text{ct}}$ (or $\mathcal{R}'_{\text{ct}}$). First, notice

that the extracted witness r is such that $\text{ct}_1 = \llbracket r \rrbracket_1$ because of how the GGM keeps track of the underlying scalars for each label. Then, we only have to show that for each query $(\mathbb{x}, \pi)$ to the oracle $\mathcal{O}_{\text{Ver}}$ the extracted r is also such that $\text{ct}_2 = r\widehat{\text{pk}}_{\text{agg}}$.

For the latter, suppose the statement contains $\text{ct}_2 \neq r\widehat{\text{pk}}_{\text{agg}}$. We know that the corresponding hash query made during the proof verification is not programmed earlier in $\mathcal{O}_{\text{Prove}}$, since the statement in $\mathcal{O}_{\text{Prove}}$ needs to have a corresponding witness. Then, for the proof $\pi = (\vec{A}, c, \zeta)$ to be valid for a statement containing $(\text{ct}_1, \text{ct}_2)$ such that $\text{ct}_1 = \llbracket r \rrbracket_1$ and $\text{ct}_2 \neq r\widehat{\text{pk}}_{\text{agg}}$, we have that

$$\begin{aligned}\vec{A} + c \cdot (\text{ct}_1, \text{ct}_2) &= \zeta \cdot (\llbracket 1 \rrbracket_1, \widehat{\text{pk}}_{\text{agg}}), \\ c &= H_{\text{ct}}(\mathbb{x}, \vec{A}).\end{aligned}$$

However, suppose there are two challenges $c \neq c'$ to allow a valid response ζ, ζ' , it implies that

$$(c - c') \cdot (\text{ct}_1, \text{ct}_2) = (\zeta - \zeta') \cdot (\llbracket 1 \rrbracket_1, \widehat{\text{pk}}_{\text{agg}}).$$

Hence, $(\text{ct}_1, \text{ct}_2) = w \cdot (\llbracket 1 \rrbracket_1, \widehat{\text{pk}}_{\text{agg}})$, which is a contradiction. Therefore, there exists at most one challenge c that allows the adversary to output a valid proof. Therefore, for each $\mathcal{O}_{\text{Ver}}$ query with the statement containing $\text{ct}_2 \neq r\widehat{\text{pk}}_{\text{agg}}$, the probability that this results in an invalid witness extraction is at most Q_{H}/p , so via the union bound

$$|\Pr[\mathbf{G}_4^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{G}_5^{\mathcal{A}} \Rightarrow 1]| \leq \frac{Q_{\text{H}}Q_{\text{Ver}}}{p}. \quad \square$$