Threshold Batched Identity-Based Encryption from Pairings in the Plain Model

Junqing Gong* Brent Waters† Hoeteck Wee‡ David J. Wu§

Abstract

In a batched identity-based encryption (IBE) scheme, ciphertexts are associated with a batch label tg^* and an identity id^* while secret keys are associated with a batch label tg and a *set* of identities S. Decryption is possible whenever $\mathsf{tg} = \mathsf{tg}^*$ and $\mathsf{id}^* \in S$. The primary efficiency property in a batched IBE scheme is that the size of the decryption key for a set S should be *independent* of the size of S. Batched IBE schemes provide an elegant cryptographic mechanism to support encrypted memory pools in blockchain applications.

In this work, we introduce a new algebraic framework for building pairing-based batched IBE. Our framework gives the following:

- First, we obtain a selectively-secure batched IBE scheme under a q-type assumption in the *plain* model. Both the ciphertext and the secret key consist of a constant number of group elements. This is the first pairing-based batched IBE scheme in the plain model. Previous pairing-based schemes relied on the generic group model *and* the random oracle model.
- Next, we show how to extend our base scheme to a *threshold* batched IBE scheme with *silent setup*. In this setting, users independently choose their own public and private keys, and there is a *non-interactive* procedure to derive the master public key (for a threshold batched IBE scheme) for a group of users from their individual public keys. We obtain a statically-secure threshold batched IBE scheme with silent setup from a q-type assumption in the plain model. As before, ciphertexts and secret keys in this scheme contain a constant number of group elements. Previous pairing-based constructions of threshold batched IBE with silent setup relied on the generic group model, could only support a polynomial number of identities (where the size of the public parameters scaled linearly with this bound), and ciphertexts contained $O(\lambda/\log\lambda)$ group elements, where λ is the security parameter.
- Finally, we show that if we work in the generic group model, then we obtain a (threshold) batched IBE scheme with shorter ciphertexts (by 1 group element) than all previous pairing-based constructions (and without impacting the size of the secret key).

Our constructions rely on classic algebraic techniques underlying pairing-based IBE and do not rely on the signature-based witness encryption viewpoint taken in previous works.

1 Introduction

Suppose we have a set of B ciphertexts $S = \{\mathsf{ct}_1, \ldots, \mathsf{ct}_B\}$ encrypted under a public key pk. The batch decryption problem is to derive a decryption key sk_S that can be used to decrypt the ciphertexts in S while still ensuring semantic security for all ciphertexts outside the set S. A trivial solution to this problem is to use hybrid encryption. Namely, to encrypt a message m, the encrypter samples a random symmetric key k and encrypts the message m using k and then encrypts the key k using the public key pk. A batch decryption key for the set $S = \{\mathsf{ct}_1, \ldots, \mathsf{ct}_B\}$ is the list of symmetric keys $k_1, \ldots, k_B$ associated with $\mathsf{ct}_1, \ldots, \mathsf{ct}_B$. Though simple, this trivial solution is inefficient because the size of the decryption key scales *linearly* with the number of ciphertexts. The goal in batch decryption is to support decryption keys for arbitrary sets of ciphertexts with size that is *sublinear* (and ideally, polylogarithmic) in the size of the set.

*East China Normal University and Shanghai Qi Zhi Institute. Email: jqgong@sei.ecnu.edu.cn.
†UT Austin and NTT Research. Email: bwaters@cs.utexas.edu.
‡NTT Research. Email: wee@di.ens.fr.
§UT Austin. Email: dwu4@cs.utexas.edu.

An application to mempool privacy. Batch decryption and its generalizations have received extensive study in the last two years [CGPP24, SAA24, BFOQ25, AFP25, CGPW25, BLT25, BCF+25]. A key motivation for studying batch decryption is it provides an elegant cryptographic solution for defending against market manipulation in blockchain and decentralized finance applications. Specifically, when users wish to post a transaction to a blockchain, they first submit the transaction to a public memory pool ("mempool"). Subsequently, miners select a subset of transactions in the mempool to include as part of the next block on the blockchain. Since miners have a lot of freedom to choose which transactions from the mempool to include in the next block, if the transaction details are public, then miners may re-order or selectively include or omit certain transactions to their own financial benefit and to the detriment of the transaction issuer. The additional value that a miner can derive through such front-running or back-running attacks is referred to as the miner extractable value or MEV [DGK+20].

A natural technique to defend against MEV attacks is for users to *encrypt* their transactions when they are submitted to the mempool [BO22, KLJD23, CGPP24]. Specifically, the blockchain specifies a public key pk and the associated secret key is secret shared across a decryption committee of nodes that operate the blockchain. When a user wants to perform a transaction, they encrypt it under pk before submitting it to the *encrypted mempool*. After the miners select a set of encrypted transactions to commit to the blockchain, the decryption committee publish shares of the *batch decryption* key that can be used to decrypt the transactions in the block. This way, transactions become public only *after* they have been committed to the blockchain; this is critical for defending against MEV attacks. Because the decryption shares for each batch of transactions must be published on the blockchain, it is important for the batch decryption key to be succinct.

Batched identity-based encryption. In this work, we focus on the notion of batched identity-based encryption (batched IBE) introduced in the recent work of Agarwal, Fernando, and Pinkas [AFP25]. The work of [AFP25] show that batched IBE directly implies a batch decryption scheme; analogously, threshold batched IBE implies the notion of threshold batch decryption considered in other works [CGPP24, SAA24, BFOQ25, CGPW25, BCF+25].

In a vanilla IBE scheme [Sha84, BF01, Coc01], one can encrypt a message m with respect to the (master) public key mpk and an identity id. The holder of the master secret key msk can in turn issue decryption keys $\mathsf{sk}_{\mathsf{id}}$ for an identity. The decryption key $\mathsf{sk}_{\mathsf{id}}$ can decrypt all ciphertexts encrypted to the associated identity id. In batched IBE, the holder of the master secret key msk can issue a decryption key sk_S associated with a *set* of identities. The key sk_S can decrypt ciphertexts encrypted to any identity $\mathsf{id} \in S$, and moreover, the size of sk_S should be sublinear in the size of the set S.

It is easy to see that batched IBE implies batch decryption. To encrypt a message m, the encrypter would sample a random identity $\mathsf{id} \xleftarrow{R} \{0,1\}^\lambda$ and encrypt m with respect to id. The batch decryption key for a collection of ciphertexts is simply a batch key for the set of identities S associated with the ciphertexts in the batch. Since the honest encryption algorithm samples the identity uniformly at random from $\{0,1\}^\lambda$, the probability that the id associated with an honestly-generated ciphertext (outside the decryption batch) is contained in S is negligible. Thus, sk_S does not compromise the semantic security of (honestly-generated) ciphertexts outside the batch.

Recent constructions of batched IBE and batch decryption impose an additional restriction on the functionality [CGPP24, SAA24, AFP25, CGPW25]. Namely, they assume that each ciphertext is additionally associated with a batch label tg (also called an "epoch" [CGPP24, SAA24, CGPW25]). Similarly, decryption keys are also associated with a batch label tg together with a set of identities (or ciphertexts). A decryption key with batch label tg can only decrypt ciphertexts with the *same* label. The main compromise is that semantic security holds only in the setting where the adversary sees a *single* decryption key associated with each batch label.[1] In the setting of MEV prevention for blockchain applications, the batch label could be the current block number. Since block numbers are unique, decrypters would only publish a single decryption key (or single set of decryption key shares) for each block number. The disadvantage of this is users would have to predict the block number for their transaction when they send their ciphertext to the encrypted mempool (or they need to prepare multiple independent ciphertexts).

1.1 Our Results

In this work, we introduce a new algebraic framework for constructing batched IBE from pairing groups. Our framework enables the following instantiations:

[1] The schemes we develop in this work can allow giving out K decryption keys for each batch label, at the cost of increasing the size of the decryption keys and the ciphertexts by an *additive* factor of K group elements or field elements. We refer to Section 2.1 and Appendix C for more details.

Scheme	$\lvert\mathsf{mpk}\rvert$	$\lvert\mathsf{ct}\rvert$	$\lvert\mathsf{sk}\rvert$	Assumption
[AFP25]	$2\lvert\mathbb{G}_1\rvert + B\lvert\mathbb{G}_2\rvert$	$3\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$	$\lvert\mathbb{G}_2\rvert$	GGM + ROM
Corollary 4.5	$5\lvert\mathbb{G}_1\rvert + B\lvert\mathbb{G}_2\rvert + \lvert\mathbb{G}_T\rvert$	$3\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$	$2\lvert\mathbb{G}_2\rvert + \lvert\mathbb{Z}_p\rvert$	q-type
Corollary D.6	$4\lvert\mathbb{G}_1\rvert + B\lvert\mathbb{G}_2\rvert$	$2\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$	$\lvert\mathbb{G}_2\rvert$	GGM + ROM

Table 1: Comparison of our batched IBE schemes with the [AFP25] scheme. For each scheme, we report the sizes of the master public key mpk, ciphertext ct, and decryption key sk as well as the underlying assumption. We write B to denote the batch size. We write $\lvert\mathbb{G}_1\rvert, \lvert\mathbb{G}_2\rvert, \lvert\mathbb{G}_T\rvert, \lvert\mathbb{Z}_p\rvert$ to refer to the sizes of an element from $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbb{Z}_p$, respectively. We write "GGM" to denote the generic bilinear group model and "ROM" to denote the random oracle model.

- **A selectively-secure scheme in the plain model.** Our first construction is a selectively-secure scheme[2] based on a q-type assumption in the *plain* model (Construction 4.2), where $q = O(B)$ and B is the batch size. Ciphertexts in this scheme consist of 4 group elements while the secret key consists of 2 group elements and 1 field element. This is the first pairing-based batched IBE scheme with security in the plain model. Previous batched IBE schemes (and batch decryption schemes) [SAA24, AFP25, CGPW25] all relied on the generic group model (together with the random oracle model). In Appendix C, we also describe ways to extend the construction to support adaptive security in the plain model, albeit with a longer common reference string (but without affecting the secret key size or the ciphertext size).
- **An adaptively-secure scheme in the generic group model.** If we work in the generic group (and random oracle) models, then we can obtain an adaptively-secure scheme with shorter ciphertexts (3 group elements) and secret keys (1 group element). Compared to the prior schemes for batched IBE and batch decryption [SAA24, AFP25, CGPW25], we save one group element in the ciphertext, one exponentiation during encryption and one pairing operation during decryption. Our master public key contains two extra group elements.

We provide a concrete comparison of the parameter sizes with the previous batched IBE scheme of [AFP25] in Table 1. We also compare the running times of encryption and decryption in Table 2.

Threshold batched IBE. Our techniques translate readily to the threshold setting where the secret key is shared across a decryption committee. In this work, we focus on two different threshold settings (though other combinations are also possible):

- **Threshold batched IBE with silent setup in the plain model.** First, we show how to extend our batched IBE in the plain model to a threshold batched IBE scheme with *silent setup* (Construction 5.5). In this context, silent setup [GKPW24, BCF+25] means that users in the decryption committee can derive their decryption shares *non-interactively* and without relying on a trusted dealer. Our work gives the first threshold batched IBE scheme with silent setup in the plain model. Ciphertexts in our scheme consists of 5 group elements and decryption key shares consist of 3 group/field elements. The public parameters contain $O(LB)$ group elements where L is the size of the decryption committee the scheme supports and B is a bound on the batch size.

  The best prior construction is the work of [BCF+25], which is a pairing-based scheme in the generic group model and only supports a polynomial-size identity space (albeit without needing to assume batch labels). Ciphertexts in their scheme contain $O(\ell + \log B)$ group elements (where B in their setting is also the bound on the size of the identity space), decryption shares consist of 3 group/field elements, and the public parameters contain $(\ell L + B)$ group elements, where $\ell = \Omega(\lambda/\log\lambda)$. In particular, the number of group elements in their scheme scales with the security parameter, whereas it is constant in our scheme. In the context of encrypted mempools,

[2] In a selectively-secure scheme, the adversary has to declare the challenge batch label tg^* and challenge identity id^* at the beginning of the game. This can be lifted to the standard adaptive security notion via complexity leveraging and assuming sub-exponential hardness (Remark 3.4) as well as via other techniques that do not need sub-exponential hardness (see Appendix C and Remark C.16).

it is important to support an exponential-size identity space, as otherwise, users would have to coordinate and associate distinct identities with their transactions when submitting transactions to an encrypted mempool; with an exponential-size identity space, users could simple associate a random identity with each transaction.

- **Threshold batched IBE in the generic group model.** We also present a threshold version of our scheme in the generic group model (Construction E.2). The size of the scheme parameters are the same as for the centralized scheme in the generic group model described above. Like [AFP25], this scheme satisfies adaptive security with static corruptions. Following the same transformation from [AFP25] (i.e., taking the identity to be a random string), we also obtain a threshold batch decryption scheme [CGPP24, CGPW25].

  As was the case in our centralized scheme, this scheme reduces the size of the ciphertext from 4 group elements to 3 group elements compared to the current state-of-the-art [CGPW25, AFP25] while increasing the size of the master public key by 2 group elements ($B + 4$ group elements vs. $B + 2$ group elements from prior work).

Compared to many previous works on batch decryption [CGPP24, SAA24, AFP25, CGPW25, BCF+25], we take a conceptually-different approach in this work. These previous works typically start by setting the secret keys for their scheme to be signatures under some pairing-based signature scheme (e.g., [BLS01]) and then design a "compatible" ciphertext structure around it (i.e., a witness encryption scheme for for the verification relation of the signature scheme). In contrast, we design the ciphertext and secret key structure *in tandem* in our scheme. Specifically, we start with a simple base scheme satisfying a weak notion of security (e.g., security without key-generation queries) and gradually build up to a fully secure scheme. Along the way, we leverage techniques and insights from the pairing-based IBE literature [BF01, BB04] to derive our final constructions.

2 Technical Overview

We now provide an overview of our construction. We start by recalling the syntax of a batched IBE scheme, as formulated in [AFP25]. As mentioned before, we work in the model where secret keys and ciphertexts are both associated with a batch label tg and a secret key with batch label tg can only decrypt ciphertexts encrypted with respect to the same batch label. We now give the full syntax:

- **Setup:** The setup algorithm samples a master secret key msk (used to issue keys) and a master public key (used for encryption).
- **Encryption:** The encryption algorithm takes the master public key mpk, a batch label tg, an identity id, and a message m, and outputs a ciphertext $\mathsf{ct}_{\mathsf{id},\mathsf{tg}}$.
- **Key-generation:** The key-generation algorithm takes the master secret key msk, a batch label tg, and a set of identities $S = \{\mathsf{id}_1, \ldots, \mathsf{id}_B\}$, and outputs a decryption key $\mathsf{sk}_{S,\mathsf{tg}}$ associated with the batch label tg and the set S. The succinctness requirement is that sk_S should be sublinear (ideally, polylogarithmic) in the size of S.
- **Decryption:** Finally, the decryption algorithm takes as input a decryption key $\mathsf{sk}_{S,\mathsf{tg}}$ and a ciphertext $\mathsf{ct}_{\mathsf{id}^*,\mathsf{tg}^*}$. If $\mathsf{tg} = \mathsf{tg}^*$ (i.e., the batch label associated with the key matches that associated with the ciphertext), and $\mathsf{id}^* \in S$, then the decryption algorithm outputs the message. Otherwise, it outputs $\perp$.

The main security requirement for a batched IBE scheme is that a ciphertext $\mathsf{ct}_{\mathsf{id}^*,\mathsf{tg}^*}$ should computationally hide the underlying message against an adversary who only has decryption keys $\mathsf{sk}_{S,\mathsf{tg}}$ where either $\mathsf{tg} \neq \mathsf{tg}^*$ or $\mathsf{id}^* \notin S$. Following [AFP25], we also impose an additional restriction that the adversary is only allowed to ask for a single decryption key for each batch label.

Notation. We leverage asymmetric prime-order pairing groups to build our batched IBE scheme. An asymmetric prime-order pairing group consists of a tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T})$, where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}$ are groups of prime order p and there is an efficiently-computable non-degenerate bilinear map $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathsf{T}$. Throughout this work, we write group elements using implicit notation [EHK+13]. Specifically, if g_1 and g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, we write $[x]_1$, $[x]_2$, and $[x]_\mathsf{T}$ to denote g_1^x, g_2^x, and $e(g_1, g_2)^x$, respectively. Similarly, we write $[x]_1 \cdot [y]_2 := e([x]_1, [y]_2) = [xy]_\mathsf{T}$.

Starting point: a correct but insecure scheme. We begin by describing a simple template for building batched IBE. Our base construction will satisfy correctness and succinctness, but only provides security for adversaries that make *no* key-generation queries. We then show how to systematically introduce additional components to obtain a secure construction.

Following [AFP25], we assume identities are elements of $\mathbb{Z}_p$ and we take B to be a bound on the batch size (i.e., the size of the set S associated with each decryption key has size at most B). We allow the size of the master public key to grow with the batch size B. Similar to previous constructions [KZG10, CGPP24, SAA24, AFP25, CGPW25], we encode a set of identities $S \subset \mathbb{Z}_p$ using a polynomial $F_S(x) = \prod_{\mathsf{id} \in S}(x - \mathsf{id})$. Our base construction then proceeds as follows:

- **Setup:** The setup algorithm samples two exponents $\tau \xleftarrow{R} \mathbb{Z}_p$ and $\alpha \xleftarrow{R} \mathbb{Z}_p$. Then the master public key and master secret key are as follows:

$$\mathsf{mpk} = ([\tau]_1, [\tau]_2, \ldots, [\tau^B]_2, [\alpha]_T) \quad \text{and} \quad \mathsf{msk} = \alpha.$$

  As in [CGPP24, SAA24, AFP25, CGPW25], the "powers-of-τ" in the public parameters are used to encode the set S (specifically, the polynomial F_S) while α is used for encrypting the message.

- **Encryption:** To encrypt a message $[m]_T \in \mathbb{G}_T$ with respect to an identity $\mathsf{id} \in \mathbb{Z}_p$, the encrypter samples $s \xleftarrow{R} \mathbb{Z}_p$ and outputs the ciphertext

$$\mathsf{ct}_{\mathsf{id}} = ([s]_1, [s(\tau - \mathsf{id})]_1, [s\alpha]_T + [m]_T).$$

  Throughout this work, we take the target group element $[m]_T$ to be the message and decryption only needs to recover the group element $[m]_T$ rather than the exponent $m \in \mathbb{Z}_p$. If we are using hybrid encryption, the encryption algorithm would sample a random $[m]_T \xleftarrow{R} \mathbb{G}_T$ and use $[m]_T$ to derive a symmetric key that is then used to encrypt the payload.

- **Key-generation:** The secret key for a set $S \subset \mathbb{Z}_p$ of size at most B is $\mathsf{sk}_S = [\alpha + F_S(\tau)]_2$, where $F_S(\tau) = \prod_{\mathsf{id} \in S}(\tau - \mathsf{id})$ is the polynomial associated with the set S. Note that $[F_S(\tau)]_2$ can be computed using the powers-of-τ in the CRS as long as $|S| \le B$.

Since the secret key is *independent* of the size of the set S, succinctness is immediate. Moreover, when $\mathsf{id} \in S$, we can write $F_S(\tau) = (\tau - \mathsf{id}) \cdot F_{S \setminus \{\mathsf{id}\}}(\tau)$. Correctness then follows from the following observation:

$$[s\alpha]_T = \overbrace{[s]_1}^{\mathsf{ct}_{\mathsf{id}}} \cdot \overbrace{[\alpha + F_S(\tau)]_2}^{\mathsf{sk}_S} - \overbrace{[s(\tau - \mathsf{id})]_1}^{\mathsf{ct}_{\mathsf{id}}} \cdot \overbrace{[F_{S \setminus \{\mathsf{id}\}}(\tau)]_2}^{\mathsf{mpk}}.$$

This scheme is insecure as soon as the adversary makes a single key query. This is because the ciphertext is malleable. Specifically, the adversary can convert a ciphertext with respect to an identity id into one with respect to any identity id' by computing $[s(\tau - \mathsf{id})]_1 + (\mathsf{id} - \mathsf{id}') \cdot [s]_1 = [s(\tau - \mathsf{id}')]_1$.

A one-key secure scheme. We can defend against this mauling strategy by introducing an additional scalar $w \xleftarrow{R} \mathbb{Z}_p$ and replace $[s(\tau - \mathsf{id})]_1$ in the ciphertext with $[sw(\tau - \mathsf{id})]_1$.[3] To preserve correctness, we also include $[w]_1$ and $[w\tau]_1$ as part of the public key and update the secret key to be $\mathsf{sk}_S = [\alpha + w \cdot F_S(\tau)]_2$. Observe that these modifications are sufficient to recover correctness:

$$[s\alpha]_T = \overbrace{[s]_1}^{\mathsf{ct}_{\mathsf{id}}} \cdot \overbrace{[\alpha + w \cdot F_S(\tau)]_2}^{\mathsf{sk}_S} - \overbrace{[sw(\tau - \mathsf{id})]_1}^{\mathsf{ct}_{\mathsf{id}}} \cdot \overbrace{[F_{S \setminus \{\mathsf{id}\}}(\tau)]_2}^{\mathsf{mpk}}.$$

With this modification, it is possible to prove security of this scheme from a q-type assumption as long as the adversary is restricted to making at most *one* key-generation query.[4] On the other hand, this scheme becomes insecure if the

[3] A similar idea is used in the design of the Boneh-Boyen IBE scheme [BB04]. There, the corresponding ciphertext term is $[s(\tau - w \cdot \mathsf{id})]_1$, with $[w]_1$ instead of $[w\tau]_1$ in the public key. The prior schemes [SAA24, AFP25, CGPW25] implicitly defend against this mauling strategy by adding an additional group element to the ciphertext. We refer to Appendix A for a more direct comparison between our construction and the previous ones.

[4] Specifically, by adapting the techniques used to analyze our main constructions, we could show that this basic scheme satisfies *static* security where the adversary has to commit to the challenge identity as well as its key-generation query ahead of time. We elide the details since we view this scheme as a stepping stone to our main constructions.

adversary can make two key-generation queries. For instance, suppose the adversary requested secret keys sk_1 and sk_2 for the singleton sets $S_1 = \{1\}$ and $S_2 = \{2\}$, respectively. In this case, $F_{S_1}(x) = (x-1)$ and $F_{S_2}(x) := (x-2)$. Correspondingly, the associated secret keys are

$$\mathsf{sk}_1 = [\alpha + w \cdot (\tau - 1)]_2$$
$$\mathsf{sk}_2 = [\alpha + w \cdot (\tau - 2)]_2.$$

The adversary can now take a linear combination of sk_1 and sk_2 to obtain

$$\mathsf{sk}' = 2 \cdot \overbrace{[\alpha + w \cdot (\tau - 2)]_2}^{\mathsf{sk}_2} - \overbrace{[\alpha + w \cdot (\tau - 1)]_2}^{\mathsf{sk}_1} = [\alpha + w \cdot (\tau - 3)]_2,$$

which is a secret key for the set $S' = \{3\}$.

From one-key security to batched IBE. To go from a one-key scheme to a batched IBE scheme, previous works [CGPP24, SAA24, AFP25, CGPW25] introduced the concept of a batch label tg and associated the batch label with each decryption key. Similarly, these works also associate a batch label with each ciphertext and decryption is only possible when the batch label associated with the decryption key matches that associated with the ciphertext. In turn, the one-key restriction applies *per batch label*. Namely, the adversary is restricted to requesting at most one key for each batch label. In some sense, using the master public key and a batch label tg, one can implicitly derive a public key for a one-key scheme specific to tg.

This problem is reminiscent of the task of constructing vanilla identity-based encryption: the objective in IBE is to derive identity-specific public keys and secret keys from a single master public key and master secret key, respectively. Indeed, the previous work of [AFP25, CGPW25] can be viewed as taking a one-key-secure scheme and implicitly composing it with the Boneh-Franklin IBE scheme [BF01]. We take a similar approach here, except for our first construction, we integrate our one-key scheme with ideas from the Boneh-Boyen IBE scheme [BB04]. In conjunction with several additional ideas, this will ultimately allow us to prove security in the *plain* model.

Recall first that in the Boneh-Boyen IBE scheme, the master secret key is $\alpha, v, h \xleftarrow{R} \mathbb{Z}_p$, the master public key is $([\alpha]_T, [v]_1, [h]_1)$, an encryption of message $[m]_T$ for identity tg is $([s]_1, [s(v + h \cdot \mathsf{tg})]_1, [s\alpha]_T + [m]_T)$, and the secret key is a pair $([r]_2, [\alpha + r(v + h \cdot \mathsf{tg})]_2)$, where $s \xleftarrow{R} \mathbb{Z}_p$ is the encryption randomness and $r \xleftarrow{R} \mathbb{Z}_p$ is the key-generation randomness. If we integrate this structure to embed batch labels in our one-key secure scheme, we obtain the following scheme (where the additional Boneh-Boyen elements are highlighted in green):

$$\begin{aligned}
\mathsf{mpk} &= ([w]_1, [w\tau]_1, \textcolor{green}{[v]_1}, \textcolor{green}{[h]_1}, [\tau]_2, \ldots, [\tau^B]_2, [\alpha]_T) \\
\mathsf{ct}_{\mathsf{id},\mathsf{tg}} &= ([s]_1,\ [sw(\tau - \mathsf{id})]_1,\ \textcolor{green}{[s(v + h \cdot \mathsf{tg})]_1},\ [s\alpha]_T + [m]_T) \\
\mathsf{sk}_{S,\mathsf{tg}} &= (\textcolor{green}{[r]_2},\ [\alpha + \textcolor{green}{r(v + h \cdot \mathsf{tg})} + w \cdot F_S(\tau)]_2).
\end{aligned}$$

Correctness follows via the composition of correctness for our one-key scheme together with correctness of the Boneh-Boyen IBE scheme:

$$[s\alpha]_T = \overbrace{[s]_1}^{\mathsf{ct}_{\mathsf{id},\mathsf{tg}}} \cdot \overbrace{[\alpha + r(v + h \cdot \mathsf{tg}) + w \cdot F_S(\tau)]_2}^{\mathsf{sk}_{S,\mathsf{tg}}} - \overbrace{[s(v + h \cdot \mathsf{tg})]_1}^{\mathsf{ct}_{\mathsf{id},\mathsf{tg}}} \cdot \overbrace{[r]_2}^{\mathsf{sk}_{S,\mathsf{tg}}} - \overbrace{[sw(\tau - \mathsf{id})]_1}^{\mathsf{ct}_{\mathsf{id},\mathsf{tg}}} \cdot \overbrace{[F_{S\setminus\{\mathsf{id}\}}(\tau)]_2}^{\mathsf{mpk}}. \tag{2.1}$$

Proving security in the plain model. We now describe our approach to proving security in the plain model. We prove selective security (where the adversary declares the batch label tg^* and the identity id^* associated with the challenge ciphertext) from the following q-type assumption:

$$\text{given} \quad \left(\begin{array}{c} [b]_1, [s]_1, [\hat{\tau}]_1, [ab]_1, [ab\hat{\tau}]_1, [abs\hat{\tau}]_1, \\ [a]_2, [b]_2, [\hat{\tau}]_2, \ldots, [\hat{\tau}^B]_2, [ab\hat{\tau}]_2, \ldots, [ab\hat{\tau}^B]_2, \end{array} \right) \quad \text{distinguish} \quad [abs]_T \ \text{ from } \ [z]_T \tag{2.2}$$

where $a, b, \hat{\tau}, s, z \xleftarrow{R} \mathbb{Z}_p$ are random exponents. In Appendix B, we show that this assumption holds unconditionally in the generic bilinear group model [Sho97, BBG05]. A key challenge in proving security in the plain model is designing

a reduction strategy that has the ability to generate keys for all batch labels tg and sets S where $\mathsf{tg} \neq \mathsf{tg}^*$ or ($\mathsf{tg} = \mathsf{tg}^*$ and $\mathsf{id} \notin S$). To do so, our proof combines the classic Boneh-Boyen puncturing strategy (which allows the reduction to simulate keys when $\mathsf{tg} \neq \mathsf{tg}^*$) with a new puncturing strategy (to allow the reduction to simulate keys where $\mathsf{tg} = \mathsf{tg}^*$ but $\mathsf{id} \notin S$). We now describe our overall proof strategy:

- Consider a selective adversary $\mathcal{A}$ for the batched IBE security game. Algorithm $\mathcal{A}$ starts by committing to the challenge batch label tg^* and the challenge identity id^*.
- The reduction algorithm takes the challenge from Eq. (2.2) and programs tg^* into the Boneh-Boyen parameters (v, h) and id^* into the powers-of-τ. Specifically, it sets the components of the public parameters as follow:
  - The reduction implicitly sets $v = \tilde{v} + b \cdot \mathsf{tg}^*$ and $h = \tilde{h} - b$ where $\tilde{v}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$ are random scalars chosen by the reduction. This is the same strategy used to prove selective security of the Boneh-Boyen IBE scheme.
  - The reduction implicitly sets $\tau = \hat{\tau} + \mathsf{id}^*$.
  - Finally, the reduction implicitly sets $w = \tilde{y} \cdot ab$ and $\alpha = \tilde{\alpha} - ab$ where $\tilde{\alpha} \xleftarrow{R} \mathbb{Z}_p$ and $\tilde{y} \in \mathbb{Z}_p$ will be specified later.

  Observe that the components given out in Eq. (2.2) allows the reduction to simulate each of these components, and moreover, they are distributed according to the real scheme.
- To answer the key-generation queries on a batch label tg and a set $S \subset \mathbb{Z}_p$, the reduction needs to simulate the secret key $\mathsf{sk}_{S,\mathsf{tg}} = ([r]_2, [\alpha + r(v + h \cdot \mathsf{tg}) + w \cdot F_S(\tau)]_2)$. With the implicit setting of the variables described above, the secret key $\mathsf{sk}_{S,\mathsf{tg}}$ must satisfy

$$\begin{aligned} \mathsf{sk}_{S,\mathsf{tg}} &= ([r]_2, [\tilde{\alpha} - ab + r(\tilde{v} + b \cdot \mathsf{tg}^* + \tilde{h} \cdot \mathsf{tg} - b \cdot \mathsf{tg}) + \tilde{y}ab \cdot F_S(\tau)]_2) \\ &= ([r]_2, [\tilde{\alpha} + r(\tilde{v} + \tilde{h} \cdot \mathsf{tg}) + rb(\mathsf{tg}^* - \mathsf{tg}) + ab(\tilde{y} \cdot F_S(\tau) - 1)]_2). \end{aligned} \tag{2.3}$$

  Our first observation is to rewrite $F_S(\tau)$ as

$$F_S(\tau) = F_S(\mathsf{id}^*) + (F_S(\tau) - F_S(\mathsf{id}^*)).$$

  For a set $S \subset \mathbb{Z}_p$, define the polynomial $G_S(x) = F_S(x + \mathsf{id}^*) - F_S(\mathsf{id}^*)$. Since $\tau = \hat{\tau} + \mathsf{id}^*$,

$$F_S(\tau) = F_S(\mathsf{id}^*) + (F_S(\tau) - F_S(\mathsf{id}^*)) = F_S(\mathsf{id}^*) + G_S(\tau - \mathsf{id}^*) = F_S(\mathsf{id}^*) + G_S(\hat{\tau}).$$

  Moreover, by construction, $G_S(0) = F_S(\mathsf{id}^*) - F_S(\mathsf{id}^*) = 0$, which means the constant term of G_S is 0. This means $[ab \cdot G_S(\hat{\tau})]_2$ can be written as a linear combination of $[ab\hat{\tau}]_2, \ldots, [ab\hat{\tau}^B]_2$, which are all terms given out in the assumption. Substituting back into Eq. (2.3), we can now write

$$\begin{aligned} \mathsf{sk}_{S,\mathsf{tg}} &= ([r]_2, [\tilde{\alpha} + r(\tilde{v} + \tilde{h} \cdot \mathsf{tg}) + rb(\mathsf{tg}^* - \mathsf{tg}) + ab(\tilde{y} \cdot F_S(\tau) - 1)]_2) \\ &= ([r]_2, [\tilde{\alpha} + r(\tilde{v} + \tilde{h} \cdot \mathsf{tg}) + rb(\mathsf{tg}^* - \mathsf{tg}) + ab(\tilde{y} \cdot F_S(\mathsf{id}^*) - 1) + ab\tilde{y} \cdot G_S(\hat{\tau})]_2). \end{aligned} \tag{2.4}$$

  The challenge in simulating $\mathsf{sk}_{S,\mathsf{tg}}$ is the fact that the reduction algorithm does *not* know $[ab]_2$, and indeed, the assumption would be false if the reduction could compute this term. Thus, simulating the secret keys requires a cancellation of the highlighted term. As argued above, simulating $[ab\tilde{y} \cdot G_S(\hat{\tau})]_2$ is possible using $[ab\hat{\tau}]_2, \ldots, [ab\hat{\tau}^B]_2$ from the assumption (the reduction chooses the exponent $\tilde{y}$ itself). We now consider two cases:
  - Suppose $\mathsf{tg} \neq \mathsf{tg}^*$. In this case, we use the classic Boneh-Boyen cancellation strategy to simulate the secret key. Namely, the reduction algorithm can sample $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$ and then implicitly set

$$r = \tilde{r} - a(\mathsf{tg}^* - \mathsf{tg})^{-1}(\tilde{y} \cdot F_S(\mathsf{id}^*) - 1).$$

    Observe that with setting of r, the term $rb(\mathsf{tg}^* - \mathsf{tg})$ will cancel out the $ab(\tilde{y} \cdot F_S(\tau) - 1)$ term in Eq. (2.4). One can check that the remaining elements can be built from terms in the assumption (see the proof of Theorem 4.4 for the details).

– Suppose $\mathsf{tg} = \mathsf{tg}^*$ but $\mathsf{id}^* \notin S$. In this case, we cannot rely on the Boneh-Boyen cancellation anymore because $rb(\mathsf{tg}^* - \mathsf{tg}) = 0$. Thus, we need a new strategy to simulate $ab(\tilde{y} \cdot F_S(\mathsf{id}^*) - 1)$.

  One approach is to consider a static adversary that commits to the set S (associated with its key-generation query on the challenge batch label tg^*) at the beginning of the security game. In this case, the reduction algorithm can set $\tilde{y} = 1/F_S(\mathsf{id}^*)$ so that $ab(\tilde{y} \cdot F_S(\mathsf{id}^*) - 1) = 0$. Note that $F_S(\mathsf{id}^*) \neq 0$ since $\mathsf{id}^* \notin S$. Effectively, this approach programs the set S associated with the key-generation query into the public parameters. However, this approach comes at the cost of requiring the adversary to commit to its key-generation query in advance. Ideally, we would like to avoid this.[5]

  To prove security against an adversary that can make make an adaptive key-generation query, we introduce one more randomization term to the secret keys. Namely, when constructing a secret key for a batch label tg and a set $S \subset \mathbb{Z}_p$, the key-generation algorithm samples a randomization factor $y \xleftarrow{R} \mathbb{Z}_p$. The secret key $\mathsf{sk}_{S,\mathsf{tg}}$ is then

$$\mathsf{sk}_{S,\mathsf{tg}} = (y, [r]_2, [\alpha + r(v + h \cdot \mathsf{tg}) + yw \cdot F_S(\tau)]_2).$$

  Since $y \in \mathbb{Z}_p$ is given out in the clear, decryption is mostly unchanged from Eq. (2.1):

$$[s\alpha]_T = [s]_1 \cdot [\alpha + r(v + h \cdot \mathsf{tg}) + yw \cdot F_S(\tau)]_2 - [s(v + h \cdot \mathsf{tg})]_1 \cdot [r]_2 - y \cdot [sw(\tau - \mathsf{id})]_1 \cdot [F_{S\setminus\{\mathsf{id}\}}(\tau)]_2.$$

  The extra scalar y gives the reduction one additional degree of freedom. Namely, Eq. (2.4) now becomes

$$\mathsf{sk}_{S,\mathsf{tg}} = ([r]_2, [\tilde{\alpha} + r(\tilde{v} + \tilde{h} \cdot \mathsf{tg}) + rb(\mathsf{tg}^* - \mathsf{tg}) + ab(y\tilde{y} \cdot F_S(\mathsf{id}^*) - 1) + aby\tilde{y} \cdot G_S(\hat{\tau})]_2).$$

  Observe that for the challenge key, the reduction can simply set $y = 1/(\tilde{y} \cdot F_S(\mathsf{id}^*))$. In this case, the term $ab(y\tilde{y} \cdot F_S(\tau) - 1) = 0$. The reduction can simulate the remaining components using the terms given out in the assumption. It remains to argue that y has the correct distribution. This is the case because we can show that the value of $\tilde{y}$ chosen by the reduction is information-theoretically hidden from the view of the adversary. Thus, over the choice of $\tilde{y} \xleftarrow{R} \mathbb{Z}_p$, the distribution of y is correctly distributed. Observe that this strategy only works if the adversary makes a single key-generation query for the batch label tg^*. The reduction algorithm only has a single degree of freedom (i.e., the value $\tilde{y}$).

- To complete the reduction, the reduction algorithm needs to simulate the components of the challenge ciphertext. With the implicit setting of the variables described above, a real ciphertext would have the following form:

$$\begin{aligned}\mathsf{ct}_{\mathsf{id}^*,\mathsf{tg}^*} &= ([s]_1, [sw(\tau - \mathsf{id}^*)]_1, [s(v + h \cdot \mathsf{tg}^*)]_1, [s\alpha]_T + [m]_T) \\ &= ([s]_1, [\tilde{y}abs\hat{\tau}]_1, [s(\tilde{v} + \tilde{h} \cdot \mathsf{tg}^*)]_1, [\tilde{\alpha}s - abs]_T + [m]_T).\end{aligned}$$

  The reduction takes s to be the corresponding elements from the assumption (see Eq. (2.2)). The first three components of $\mathsf{ct}_{\mathsf{id}^*,\mathsf{tg}^*}$ can thus be constructed using the components from Eq. (2.2). The reduction algorithm simulates the final component $[\tilde{\alpha}s - abs]_T + [m]_T$ as $\tilde{\alpha} \cdot [s]_1 \cdot [1]_2 - [z]_T + [m]_T$, where $[z]_T$ is the challenge component. When $z = abs$, this perfectly simulates the real ciphertext whereas if $z \xleftarrow{R} \mathbb{Z}_p$, then the ciphertexts perfectly hide the message. This completes the security reduction.

Taken together, this construction yields a batched IBE scheme in the plain model where the ciphertexts contain 4 group elements and the secret keys contain 2 group elements and 1 field element. Compared to the previous batched IBE and batch decryption schemes [SAA24, AFP25, CGPW25], this construction has the same ciphertext size but slightly longer keys. However, the prior work all relied on the generic group model and the random oracle model for the security analysis we can prove security in the *plain* model. We refer to Section 4 for the full description.

[5] We note that requiring the adversary to have to commit to its key-generation query is technically very different from asking it to commit to the challenge identity and batch label. Standard complexity leveraging allows us to go from a scheme that is selective in the challenge identity and batch label to one that is adaptively secure (see Remark 3.4). However, we cannot use complexity leveraging to lift a scheme that is selective in the set S to one that allows the adversary to pick S adaptively. This is because guessing the set S would blow up the ciphertext size and the size of the decryption keys by a factor of B, which breaks succinctness.

Better efficiency in the generic group model. Alternatively, we can also build a scheme with better efficiency by relying on the generic group and random oracle models. For example, take again our one-key scheme, but now, instead of integrating batch labels via the Boneh-Boyen approach, we did so with the Boneh-Franklin approach (similar to prior works [SAA24, AFP25, CGPW25]). Recall that in the Boneh-Franklin IBE scheme [BF01], the master secret key is $\alpha \xleftarrow{R} \mathbb{Z}_p$, the master public key is $[\alpha]_1$, an encryption of message $[m]_T$ for identity tg is $[s\alpha \cdot H(\mathsf{tg})]_T + [m]_T$, and the secret key for tg is $[\alpha \cdot H(\mathsf{tg})]_2$. Here, $s \xleftarrow{R} \mathbb{Z}_p$ is the encryption randomness and H is a hash function that takes a label tg and outputs the element $[H(\mathsf{tg})]_2$ in $\mathbb{G}_2$. If we apply this strategy to our one-key secure scheme, we arrive at the following scheme (where the additional Boneh-Franklin elements are highlighted in green):

$$\begin{aligned}
\mathsf{mpk} &= ([w]_1, [w\tau]_1, [\alpha]_1, [\tau]_2, \ldots, [\tau^B]_2) \\
\mathsf{ct}_{\mathsf{id},\mathsf{tg}} &= ([s]_1,\ [sw(\tau - \mathsf{id})]_1,\ [s\alpha \cdot H(\mathsf{tg})]_T + [m]_T) \\
\mathsf{sk}_{S,\mathsf{tg}} &= [\alpha \cdot H(\mathsf{tg}) + w \cdot F_S(\tau)]_2.
\end{aligned}$$

Correctness follows via the composition of correctness for our one-key scheme together with correctness of the Boneh-Franklin IBE scheme:

$$[s\alpha \cdot H(\mathsf{tg})]_T = \overbrace{[s]_1}^{\mathsf{ct}_{\mathsf{id},\mathsf{tg}}} \cdot \overbrace{[\alpha \cdot H(\mathsf{tg}) + w \cdot F_S(\tau)]_2}^{\mathsf{sk}_{S,\mathsf{tg}}} - \overbrace{[sw(\tau - \mathsf{id})]_1}^{\mathsf{ct}_{\mathsf{id},\mathsf{tg}}} \cdot \overbrace{[F_{S\setminus\{\mathsf{id}\}}(\tau)]_2}^{\mathsf{mpk}}.$$

This immediately gives a scheme with shorter ciphertexts (3 group elements instead of 4). Following a similar recipe as before (but using the Boneh-Franklin cancellation strategy to handle key-generation queries on tags $\mathsf{tg} \neq \mathsf{tg}^*$), we believe one could prove *static* security of this version from a q-type assumption and modeling H as a random oracle. Note that this would still not rely on the generic group model. If we introduce the randomizing scalar y into the secret key, then the same proof strategy should allow the adversary to make an adaptive key-generation query on the challenge batch label.

On the other hand, if we want to show full adaptive security for the scheme *without* the additional randomizing scalar in the secret key, then we can do so in the generic bilinear group model [Sho97, BBG05]. This yields a batched IBE scheme where the ciphertext consists of three group elements and the secret key consists of a single group element. Compared to the prior schemes, we save one group element in the ciphertext, one exponentiation during encryption and one pairing during decryption (see Table 1 for details). We describe this construction and its analysis in Appendix D.

Concurrent work. In a concurrent and independent work, Fernando, Policharla, Tonkikh, and Xiang [FPTX25] showed how to construct a (threshold) batched IBE scheme in the generic group model and the random oracle model where the ciphertext size consists of three group elements. This is the same level of efficiency achieved by our batched IBE scheme in the generic group and random oracle model (Corollary E.6). In addition, they show how to build a K-key threshold batched IBE scheme *without* batch labels (i.e., security holds as long as the adversary gets at most K secret keys on arbitrary sets of identities). In their scheme, the CRS size scales multiplicatively with K while the size of the ciphertext and the secret keys remain unchanged. Their construction can be viewed as taking a one-key scheme and compiling it to a K-key scheme. In addition, the key-generation algorithm in their construction is stateful (i.e., secret keys are associated with an index $i \in [K]$ and security assumes that the K keys are generated with respect to distinct indices). At a high-level, we can view their construction as including K copies of the CRS for a one-key secure scheme, and the i^{th} key-generation query is generated with respect to the i^{th} copy of the CRS for the one-key scheme. Moreover, by relying on linearity of decryption for the underlying one-key scheme, they can retain the same ciphertext structure as the underlying one-key scheme. The same approach is also applicable to the basic constructions we described above to obtain schemes with K-key security (without batch labels) and stateful key-generation.

The focus of this work is on schemes that do not impose any restriction on the total number of keys the adversary can request; instead, as in prior work [CGPP24, AFP25, CGPW25], we restrict the adversary to a single key (or up to K keys; see Section 2.1) per batch label. In addition, our primary goal in this work is to give constructions with security in the *plain* model, and our work gives the first pairing-based constructions of (threshold) batched IBE with security in the plain model. The work of [FPTX25] relies on both the generic group model and the random oracle model. In addition, we show how to construct a threshold scheme with *silent* setup (also in the plain model). The threshold scheme from [FPTX25] relies on a central setup to generate the individual decryption key shares.

2.1 Extensions to the Base Scheme

The algebraic structure of our batched IBE scheme is directly amenable to a number of extensions. We survey some of the main results here.

Giving out multiple keys with the same batch label. Like previous constructions [CGPP24, AFP25, CGPW25], our basic scheme only guarantees security against adversaries that can request a single key for each batch label. A stronger security notion would allow an adversary to request an arbitrary number of keys for each batch label. In this case, there is no longer a need for batch labels. Currently, the only constructions that support this capability either rely on lattice-based attribute-based encryption [BLT25] or are limited to a polynomial-size identity space [BFOQ25, BCF+25].

While it is unclear how to modify our scheme to support an arbitrary number of keys per batch label, it is straightforward to support giving out an a priori bounded number of keys for each batch label. Specifically, if K is the collusion bound (i.e., the number of keys we need to give out for each batch label), then we can obtain a scheme that allows the adversary to make up to K key-generation queries on the challenge batch label at the cost of increasing the secret keys by K field elements, the ciphertext by K group elements, and the public key by $2K$ group elements.

The basic idea in our construction is to replace the single scalar w in the previous construction with a vector $\mathbf{w} \in \mathbb{Z}_p^K$. Then, the secret key and the ciphertexts are defined as follows:

$$\begin{aligned}\mathsf{sk}_{S,\mathsf{tg}} &:= (\mathbf{y}, [r]_2, [\alpha + r(v + h \cdot \mathsf{tg}) + \mathbf{y}^\mathsf{T}\mathbf{w} \cdot F_S(\tau)]_2)\\ \mathsf{ct}_{\mathsf{id},\mathsf{tg}} &:= ([s]_1, [s\mathbf{w}(\tau - \mathsf{id})]_1, [s(v + h \cdot \mathsf{tg})]_1, [s\alpha]_\mathsf{T} + [m]_\mathsf{T}),\end{aligned}$$

where $\mathbf{y} \xleftarrow{\mathsf{R}} \mathbb{Z}_p^K$ in the secret key. The main property we require in the security analysis is that if $\mathbf{y}_1, \ldots, \mathbf{y}_K \in \mathbb{Z}_p^K$ are linearly-independent vectors, then the values of $\mathbf{w}^\mathsf{T}\mathbf{y}_1, \ldots, \mathbf{w}^\mathsf{T}\mathbf{y}_K$ are uniform and independent over $\mathbb{Z}_p$ when $\mathbf{w} \xleftarrow{\mathsf{R}} \mathbb{Z}_p^K$. This K-wise independence property enables security against K-collusions (i.e., an adversary that has K keys for a particular batch label). We describe this construction in Appendix C.

Thresholdizing the scheme. In a threshold batched IBE scheme [AFP25], the master secret key msk is instead secret shared across L different authorities. Each authority holds a share msk_i of the master secret key. Using msk_i, the authority can give out a share of the decryption key $\mathsf{sk}_{S,\mathsf{tg},i}$ for any batch label tg and set S. Finally, given the decryption key shares $\{\mathsf{sk}_{S,\mathsf{tg},i}\}_{i \in S'}$ for a sufficiently-large set $S' \subseteq S$, one can decrypt the ciphertext and recover the underlying message.

Similar to [AFP25], it is straightforward to obtain a threshold version of the scheme. For simplicity, we first describe the approach for the basic one-key secure scheme (without batch labels). Recall in that scheme that the master secret key is (α, w). To obtain a threshold version, we simply secret share α and w. Concretely, for a threshold T, let $(\alpha_1, \ldots, \alpha_L)$ be a T-out-of-L Shamir secret sharing of α and $(w_1, \ldots, w_L)$ be a T-out-of-L Shamir secret sharing of w. We now enumerate the scheme components (using the same syntax as before):

$$\begin{aligned}\mathsf{msk} &= (\alpha, w) & \mathsf{sk}_{S,\ell} &= [\alpha_i + w_i \cdot F_S(\tau)]_2\\ \mathsf{mpk} &= ([w]_1, [w\tau]_1, [\tau]_1, [\tau]_2, \ldots, [\tau^B]_2, [\alpha]_\mathsf{T}) & \mathsf{ct}_\mathsf{id} &= ([s]_1, [sw(\tau - \mathsf{id})]_1, [s\alpha]_\mathsf{T} + [m]_\mathsf{T}).\end{aligned}$$

Given a collection of secret keys $\{\mathsf{sk}_{S,\ell}\}_{\ell \in U}$ for a set S where $|U| \geq T$ and a ciphertext encrypted to an identity $\mathsf{id} \in S$, the decryption algorithm first computing the reconstruction coefficients $\omega_\ell \in \mathbb{Z}_p$ where $\sum_{\ell \in U} \omega_\ell \alpha_\ell = \alpha$ and $\sum_{\ell \in U} \omega_\ell w_\ell = w$. Then, the decryption algorithm computes

$$[s\alpha]_\mathsf{T} = \left(\sum_{\ell \in U} \omega_\ell \cdot \overbrace{[s]_1}^{\mathsf{ct}_\mathsf{id}} \cdot \overbrace{[\alpha_\ell + w_\ell \cdot F_S(\tau)]_2}^{\mathsf{sk}_{S,\ell}}\right) - \overbrace{[sw(\tau - \mathsf{id})]_1}^{\mathsf{ct}_\mathsf{id}} \cdot \overbrace{[F_{S \setminus \{\mathsf{id}\}}(\tau)]_2}^{\mathsf{mpk}}.$$

We can now extend to a threshold batched IBE by introducing the batch label as we described previously. Note that this basic approach for thresholding is not compatible with the randomizing scalar y we introduced to handle adaptive

key-generation queries on the challenge batch label in the security proof (for correctness, the different decrypters would need to choose the same y when generating key). Thus, we can only apply this technique to our batched IBE scheme in the generic group model (Construction D.1) that does not need randomization scalars or the variant of our first scheme (Construction 4.2) without the randomizing scalars. In the latter case, we would work in a static security model where the adversary has to declare its key-generation query for the challenge batch label in advance. We refer to Appendix E for the formal description for getting a batched threshold encryption scheme in the generic group model.

A caveat of this thresholding approach is that we only achieve security in a model where the adversary needs to specify the *same* set of identities S to every decryption authority for a given batch label tg. A stronger requirement would allow the adversary to ask K decryption authorities to issue a key share for *different* sets $S_1, \ldots, S_L$ on batch label tg. Note that each decryption authority still only gives out a single key with batch label tg. The more restricted notion (considered in [AFP25]) would require decrypters to coordinate so as to never inadvertently release decryption key shares for different sets with respect to the same batch label. The stronger definition is more natural for threshold settings in that decryption authorities can operate independently. As we discuss more in Section 5 and Remark 5.3, our techniques for construction threshold batched IBE with silent setup (discussed in more detail below) simultaneously achieves this stronger security notion.

Supporting silent thresholds. In the basic threshold batched IBE scheme, we would need to either assume a trusted dealer generates the shares of the master secret key msk_ℓ for each authority, or alternatively, that the authorities engage in an interactive (and oftentimes, computationally expensive [TCZ+20]) distributed key-generation protocol to jointly sample their keys. A line of recent works [RSY21, GKPW24, ADM+24, DJWW25, WW25a] have introduced an appealing alternative model of threshold cryptography with a full *non-interactive* setup phase. In this model, users can independently choose a public key pk and secret key sk. Then, there is a public and deterministic aggregation algorithm that takes any set of public keys $\{\mathsf{pk}_\ell\}_{\ell \in [L]}$ and aggregates them into a short public key mpk. The aggregated public key mpk now functions as the public key for a threshold batch decryption scheme where the individual user secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_L$ play the role of key shares. Threshold encryption with silent setup is part of an extensive line of recent works focused on realizing advanced encryption capabilities without a trusted authority. Similar notions in this line of work include registration-based encryption [GHMR18, GHM+19, DKL+23, GKMR23, FKdP23], registered attribute-based encryption [HLWW23, ZZGQ23, FWW23, CHW25, WW25b], registered functional encryption [FFM+23, DPY24], and distributed broadcast encryption [WQZD10, BZ14, KMW23, CW24].

Very recently, the work of [BCF+25] show how to construct a threshold batch decryption scheme from pairings by integrating the batch decryption scheme from [BFOQ25] with the threshold encryption scheme with silent setup from [GKPW24]. This construction essentially gives a threshold batched IBE scheme where (1) the identity space has polynomial size; (2) the ciphertexts contain $O(\lambda/\log \lambda)$ group elements; and (3) security relies on the generic group (and random oracle model). In this work, we show how to integrate our batched IBE scheme with the recent silent threshold IBE scheme from [WW25a] to obtain a threshold batched IBE scheme with silent setup that improves upon each of these axis. Namely, (1) our scheme supports arbitrary identities (but with batch labels); ciphertexts contain a constant number of group elements; and (3) security is based on a q-type assumption in the plain model. On the other hand, the public parameters in our scheme contain $O(LB)$ group elements where L is the maximum number of users in a decryption committee, and B is the batch size. The work of [BCF+25] requires public parameters with $O(\lambda L/\log \lambda + B)$ group elements.

One way to view our construction is to again start with our one-key scheme, but now we integrate it with the threshold IBE scheme with silent setup from [WW25a] (which shares a similar structure with pairing-based broadcast encryption [BGW05]). For simplicity, we show how the main ideas apply to the one-key scheme. We refer to Section 5 for the full construction and analysis:

- **Public parameters:** Let B be the batch size, L be the size of a decryption committee, and T be the desired threshold.[6] The public parameters now contain an additional set of group elements $[c]_1, [c^2]_1, \ldots, [c^{2L}]_1$ that will be used to aggregate individual user public keys. In addition, it will also contain cross terms $[c^i \tau^j]_1$ and $[c^i \tau^j]_2$ for all $i \in [2L]$ and $j \in [0, B]$. In addition, let $t \xleftarrow{R} \mathbb{Z}_p$ be a target value and let $t_1, \ldots, t_L \in \mathbb{Z}_p$ be a

[6] We believe we can extend our scheme to support *dynamic* thresholds (where the threshold is declared at encryption time) via the powers-of-two strategy from [WW25a] (see Remark 5.10). This incurs logarithmic overhead in the size of the public parameters. For ease of exposition in this work, we just focus on the setting of a fixed threshold, which captures the main technical challenge of supporting silent setup.

T-out-of-N Shamir secret sharing of t. Following [WW25a], let $z_0 = \sum_{\ell\in[L]} c^\ell t_\ell$ be the *aggregated* shares. The public parameters now contain

$$\mathsf{pp} = (\{[c^\ell \tau^j]_1, [c^\ell \tau^j]_2\}_{\ell\in[0,2L],\tau\in[0,B]}, [c^{L+1}t]_\mathsf{T}, [z_0]_2, \{[c^{L+1-\ell+i}t_i]_2\}_{i\in[L],\ell\neq i}).$$

In the full construction (Construction 5.5), we pre-aggregate the cross terms $[c^{L+1-\ell+i}t_i]_2$ to obtain shorter public parameters (of size $O(LB)$ rather than $O(L^2 + LB)$). For ease of exposition, we elide this step here.

- **User key generation:** Each user samples their own secret key $\alpha_\ell, w_\ell \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ for the underlying one-key scheme. Their public key is $([c^{L+1}\alpha_\ell]_\mathsf{T}, [w_\ell]_1)$. In addition, they also publish a collection of cross terms $[c^i\alpha_\ell]_2$, $[c^i w_\ell \tau^j]_2$ for all $i \in [2L] \setminus \{L+1\}$ and $j \in [0, B]$ that will be used to aggregate public keys.
- **Key aggregation:** Take any collection of public keys $\{([\alpha_\ell c^{L+1}]_\mathsf{T}, [w_\ell]_1)\}_{\ell\in[L]}$ together with their aggregation hints. We define the aggregated key components to be

$$[z]_2 = \sum_{\ell\in[L]} [c^\ell(t_\ell + \alpha_\ell)]_2 \quad \text{and} \quad [w]_2 = \sum_{\ell\in[L]} [c^\ell w_\ell]_2 \quad \text{and} \quad [w\tau]_2 = \sum_{\ell\in[L]} [c^\ell w_\ell \tau]_2.$$

All of these components can be computed using the public parameters $[z_0]_2$ and each user's public aggregation components $[c^\ell\alpha_\ell]_2$, $[c^\ell w_\ell]_2$, and $[c^\ell w_\ell\tau]_2$.

- **Decryption share computation:** Each user issues decryption shares in a similar manner as in the underlying one-key scheme. Namely, if the user's secret key is (α_ℓ, w_ℓ), then the decryption share for a set $S \subset \mathbb{Z}_p$ is $\mathsf{sk}_{S,\ell} = [c^{L+1}(\alpha_\ell + w_\ell \cdot F_S(\tau))]_2$.
- **Encryption:** To encrypt $[m]_\mathsf{T}$ with respect to identity $\mathsf{id} \in \mathbb{Z}_p$, the encrypter samples $s \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and then constructs

$$\mathsf{ct}_\mathsf{id} = ([s]_1, [sw(\tau - \mathsf{id})]_2, [sz]_2, [sc^{L+1}t]_\mathsf{T} + [m]_\mathsf{T}).$$

- **Decryption:** Take any collection of decryption shares for $\mathsf{sk}_{S,\ell}$ for $\ell \in U \subseteq [L]$ where $|U| > T$. The decrypter computes for each $\ell \in U$:

$$\overbrace{[s]_1}^{\mathsf{ct}_\mathsf{id}} \cdot \overbrace{[c^{L+1}(\alpha_\ell + w_\ell \cdot F_S(\tau))]_2}^{\mathsf{sk}_{S,\ell}} - \overbrace{[c^{L+1-\ell}F_{S\setminus\{\mathsf{id}\}}(\tau)]_1}^{\mathsf{pp}} \cdot \overbrace{[sw(\tau - \mathsf{id})]_2}^{\mathsf{ct}_\mathsf{id}}.$$

Since $w = \sum_{i\in[L]} c^i w_i$, this means

$$c^{L+1-\ell}F_{S\setminus\mathsf{id}}(\tau) \cdot sw(\tau - \mathsf{id}) = c^{L+1}sw_\ell F_S(\tau) + \sum_{i\neq\ell} c^{L+1-\ell+i}sw_i F_S(\tau).$$

The decrypter now computes

$$[c^{L+1-\ell+i}sw_i F_S(\tau)]_\mathsf{T} = \overbrace{[s]_1}^{\mathsf{ct}_\mathsf{id}} \cdot \overbrace{[c^{L+1-\ell+i}w_i\tau^j]_2}^{\mathsf{pk}_i}.$$

Taken together, the decrypter is able to obtain $[c^{L+1}s\alpha_\ell]_\mathsf{T}$ for all $\ell \in U$. Next, using the fact that $z = \sum_{i\in[L]} c^i(t_i + \alpha_i)$, the decrypter can compute for each $\ell \in U$

$$\overbrace{[c^{L+1-\ell}]_1}^{\mathsf{pp}} \cdot \overbrace{[sz]_2}^{\mathsf{ct}_\mathsf{id}} = [c^{L+1}s(t_\ell + \alpha_\ell)]_\mathsf{T} + \sum_{i\neq\ell}[c^{L+1-\ell+i}s(t_i + \alpha_i)]_\mathsf{T}.$$

Again, using the cross terms, the decrypter can compute for all $i \neq \ell$,

$$[c^{L+1-\ell+i}s(t_i + \alpha_i)]_\mathsf{T} = \overbrace{[s]_1}^{\mathsf{ct}_\mathsf{id}} \cdot \overbrace{[c^{L+1-\ell+i}t_i]_2}^{\mathsf{pp}} + \overbrace{[s]_1}^{\mathsf{ct}_\mathsf{id}} \cdot \overbrace{[c^{L+1-\ell+i}\alpha_i]_2}^{\mathsf{pk}_i}$$

Taken together, the decrypter now obtains $[c^{L+1}s(t_\ell + \alpha_\ell)]_\mathsf{T}$ for all $\ell \in U$. From above, the decrypter has already computed $[c^{L+1}s\alpha_\ell]_\mathsf{T}$ for all $\ell \in U$, so in total, the decrypter obtains $[sc^{L+1}t_\ell]_\mathsf{T}$ for each $\ell \in U$. Since $t_1, \ldots, t_N$ is a T-out-of-N secret sharing of t, as long as $|U| \geq T$, the decrypter can now recover $[sc^{L+1}t]_\mathsf{T}$, which is sufficient to recover the message.

The basic approach here illustrates how we can incorporate the aggregation and silent setup mechanism from [WW25a] into our batched IBE scheme to support a silent setup functionality. We refer to Section 5 for the description of the full scheme (which includes both the batch labels as well as the per-key randomization term needed to prove security).

3 Preliminaries

Throughout this work, we write λ to denote the security parameter. For a positive integer $n \in \mathbb{N}$, we write $[n] \coloneqq \{1, \ldots, n\}$. For a set S, we write $\{x_i\}_{i \in S}$ to denote the set of pairs $\{(i, x_i)\}_{i \in S}$. For a finite set S, we write $x \xleftarrow{\mathsf{R}} S$ to denote that x is sampled uniformly at random from S. We write $\mathsf{poly}(\lambda)$ to denote a function that is bounded by a fixed polynomial in λ. We write $\mathsf{negl}(\lambda)$ to denote a function that is $o(\lambda^{-c})$ for all constants $c \in \mathbb{N}$. We say an event E_λ (indexed by a security parameter) happens with overwhelming probability if it occurs with probability $1 - \mathsf{negl}(\lambda)$. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. We say two distributions are computationally indistinguishable if no efficient adversary can distinguish them except with negligible probability. We say they are statistically indistinguishable if their statistical distance is bounded by a negligible function. Throughout, we use boldface lowercase letters (e.g., $\mathbf{u}, \mathbf{v}$) to denote vectors and boldface uppercase letters (e.g., $\mathbf{A}, \mathbf{B}$) to denote matrices. We use non-boldface letters to denote their components (e.g., $\mathbf{v} = [v_1, \ldots, v_n]$).

Prime-order pairing groups. Throughout this work, we use prime-order asymmetric pairing groups, which we define formally below:

Definition 3.1 (Prime-Order Pairing Group)**.** An asymmetric prime-order pairing group consists of an efficient algorithm GroupGen that takes as input the security parameter 1^λ and outputs the description of a pairing group $\mathcal{G} \coloneqq (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e)$ where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}$ are cyclic groups of prime order $p > 2^\lambda$, $e \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathsf{T}$ is a non-degenerate bilinear map, and $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. The group operation in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}$ as well as the pairing e are all efficiently-computable. For convenience, we will sometimes assume that there is a fixed function $p = p(\lambda)$ such that $\mathsf{GroupGen}(1^\lambda)$ always outputs a group with order $p(\lambda)$.

Implicit notation. We describe group elements using implicit notation [EHK+13]. Specifically, for a pairing group $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e)$ and a matrix $\mathbf{M} \in \mathbb{Z}_p^{n \times m}$, we write $[\mathbf{M}]_1 \coloneqq g_1^{\mathbf{M}} \in \mathbb{G}_1^{\mathbf{M}}$ to denote the matrix of group elements where exponentiation is applied component-wise to the elements of the matrix. We define $[\mathbf{M}]_2 \coloneqq g_2^{\mathbf{M}}$ and $[\mathbf{M}]_\mathsf{T} \coloneqq e(g_1, g_2)^{\mathbf{M}}$. For matrices $\mathbf{A}, \mathbf{B}$ and $s \in \{1, 2, \mathsf{T}\}$, we write $[\mathbf{A}]_s \pm [\mathbf{B}]_s \coloneqq [\mathbf{A} \pm \mathbf{B}]_s$ and $\mathbf{A} \cdot [\mathbf{B}]_s \coloneqq [\mathbf{A}]_s \cdot \mathbf{B} = [\mathbf{AB}]_s$. Finally, we write $[\mathbf{A}]_1 \cdot [\mathbf{B}]_2 = [\mathbf{AB}]_\mathsf{T}$, where the individual components of the product $[\mathbf{AB}]_\mathsf{T}$ are computed using the pairing.

Linear secret sharing for threshold policies. A linear secret sharing scheme for T-out-of-N threshold policy over $\mathbb{Z}_p$ can be described by a share-generation matrix $\mathbf{M} \in \mathbb{Z}_p^{N \times T}$ with the following properties:

- For every set $U \subseteq [N]$ of size $\geq T$, there exists a vector $\boldsymbol{\omega} \in \mathbb{Z}_p^N$ where $\boldsymbol{\omega}^\mathsf{T}\mathbf{M} = \mathbf{e}_i^\mathsf{T}$ and $\omega_i = 0$ for $i \notin U$.
- For every set $U \subseteq [N]$ of size $< T$, there exists a vector $\mathbf{w} \in \mathbb{Z}_p^T$ such that $w_1 = 1$ and $\mathbf{m}_i^\mathsf{T}\mathbf{w} = 0$ for all $i \in U$ where $\mathbf{m}_i^\mathsf{T}$ is the i^{th} row of $\mathbf{M}$.

To share a secret $\alpha \in \mathbb{Z}_p$, we sample $v_2, \ldots, v_T \leftarrow \mathbb{Z}_p$ and set $\mathbf{v} = (\alpha, v_2, \ldots, v_T)^\mathsf{T}$. Then $\mathbf{m}_i^\mathsf{T}\mathbf{v}$ is the share belonging to i^{th} party. The classic Shamir secret sharing scheme [Sha79] satisfies this property whenever $N < p$. Concretely, we would take the matrix $\mathbf{M}$ to be a Vandermonde matrix associated with the interpolation points $[N]$.

Batched identity-based encryption. Next, we recall the formal definition of batched identity-based encryption (IBE) from [AFP25].

Definition 3.2 (Batched Identity-Based Encryption [AFP25])**.** A batched identity-based encryption scheme Π_{BatchIBE} is a tuple of efficient algorithms $\Pi_{\mathsf{BatchIBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Digest}, \mathsf{ComputeKey}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: On input the security parameter $\lambda \in \mathbb{N}$, the setup algorithm outputs a set of public parameters pp. We assume that the public parameters (implicitly) specifies the message space $\mathcal{M}$, identity space $\mathcal{I}$, and batch label space $\mathcal{T}$ for the encryption scheme.
- $\mathsf{KeyGen}(\mathsf{pp}, 1^B) \to (\mathsf{mpk}, \mathsf{msk})$: On input the public parameters pp and an upper bound on the batch size B, the key-generation algorithm outputs a master public key mpk and a master secret key msk. We assume that mpk and msk also include an implicit description of the message space $\mathcal{M}$, identity space $\mathcal{I}$, and batch label space $\mathcal{T}$ (from pp).
- $\mathsf{Encrypt}(\mathsf{mpk}, m, \mathsf{id}, \mathsf{tg}) \to \mathsf{ct}$: On input the master public key mpk, a message $m \in \mathcal{M}$, an identity $\mathsf{id} \in \mathcal{I}$, and a batch label $\mathsf{tg} \in \mathcal{T}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Digest}(\mathsf{mpk}, S) \to \mathsf{dig}$: On input the master public key mpk and a set of identities S, the digest algorithm outputs a digest dig. This algorithm is deterministic.
- $\mathsf{ComputeKey}(\mathsf{msk}, \mathsf{dig}, \mathsf{tg}) \to \mathsf{sk}$: On input the master secret key msk, a digest dig, and a batch label tg, the key-computation algorithm outputs a secret key sk associated with dig and tg.
- $\mathsf{Decrypt}(\mathsf{mpk}, \mathsf{sk}, S, (\mathsf{id}, \mathsf{tg}), \mathsf{ct}) \to m$: On input the master public key mpk, a secret key sk, a set of identities S, an identity-label pair $(\mathsf{id}, \mathsf{tg})$, and a ciphertext ct, the decryption algorithm outputs a message $m \in \mathcal{M}$ (or possibly a special symbol $\perp$ to indicate decryption failed). This algorithm is deterministic.

We require Π_{BatchIBE} satisfy the following properties:

- **Correctness:** For all $\lambda, B \in \mathbb{N}$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda)$, all messages $m \in \mathcal{M}$, identities $\mathsf{id} \in \mathcal{I}$, and batch labels $\mathsf{tg} \in \mathcal{T}$ (where $\mathcal{M}, \mathcal{I}, \mathcal{T}$ are the message, identity, and batch label spaces associated with pp, respectively), all sets $S \subseteq \mathcal{I}$ of size B where $\mathsf{id}^* \in S$, we have

$$\Pr\left[\mathsf{Decrypt}(\mathsf{mpk}, \mathsf{sk}, S, (\mathsf{id}, \mathsf{tg}), \mathsf{ct}) = m : \begin{array}{c}(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, 1^B) \\ \mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, m, \mathsf{id}, \mathsf{tg}) \\ \mathsf{dig} = \mathsf{Digest}(\mathsf{mpk}, S) \\ \mathsf{sk} \leftarrow \mathsf{ComputeKey}(\mathsf{msk}, \mathsf{dig}, \mathsf{tg})\end{array}\right] = 1.$$

- **Adaptive security:** For a security parameter λ, a batch size B, a bit $\beta \in \{0, 1\}$, and an adversary $\mathcal{A}$, we define the batched IBE security game as follows:
  - The challenger starts by computing $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, 1^B)$. It gives $(1^\lambda, 1^B, \mathsf{pp}, \mathsf{mpk})$ to $\mathcal{A}$. Let $\mathcal{M}, \mathcal{I}, \mathcal{T}$ be the message space, identity space, and batch label space associated with pp.
  - Algorithm $\mathcal{A}$ can now make key-computation queries. On each query, algorithm $\mathcal{A}$ specifies a set $S \subseteq \mathcal{I}$ where $|S| \leq B$ and a batch label $\mathsf{tg} \in \mathcal{T}$. The challenger replies with the secret key $\mathsf{sk} \leftarrow \mathsf{ComputeKey}(\mathsf{msk}, \mathsf{Digest}(\mathsf{mpk}, S), \mathsf{tg})$.
  - After $\mathcal{A}$ is finished making key-computation queries, it outputs two messages $m_0, m_1 \in \mathcal{M}$ and a challenge identity-label pair $(\mathsf{id}^*, \mathsf{tg}^*)$. The challenger responds with a challenge ciphertext $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, m_\beta, \mathsf{id}^*, \mathsf{tg}^*)$.
  - Algorithm $\mathcal{A}$ can continue to make key-computation queries. The challenger answers the queries exactly as before.

– At the end of the game, algorithm $\mathcal{A}$ outputs a bit $\beta' \in \{0,1\}$, which is the output of the experiment.

We say an adversary $\mathcal{A}$ is admissible if the following two conditions hold:

– Algorithm $\mathcal{A}$ makes at most one key-computation query on the challenge batch label tg^*.

– Algorithm $\mathcal{A}$ does not make a key-computation query on a pair (S, tg) where $\mathsf{tg} = \mathsf{tg}^*$ and $\mathsf{id}^* \in S$.

We say Π_{BatchIBE} is secure if for all polynomials $B = B(\lambda)$ and all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\beta' = 1 : \beta = 0] - \Pr[\beta' = 1 : \beta = 1]| = \mathsf{negl}(\lambda) \tag{3.1}$$

in the above security game. We say Π_{BatchIBE} is secure for a specific batch size $B = B(\lambda)$ if the above holds for the specific function B.

- **Succinctness:** There exists a universal polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda, B \in \mathbb{N}$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda)$, all $(\mathsf{mpk}, \mathsf{msk})$ in the support of $\mathsf{KeyGen}(\mathsf{pp}, 1^B)$, all digests dig in the support of $\mathsf{Digest}(\mathsf{mpk}, \cdot)$, and all batch labels $\mathsf{tg} \in \mathcal{T}$ (where $\mathcal{T}$ is the batch label space associated with pp), the running time of $\mathsf{ComputeKey}(\mathsf{msk}, \mathsf{dig}, \mathsf{tg})$ and the size of the digest dig is $\mathsf{poly}(\lambda)$ and in particular, independent of B.

Definition 3.3 (Selective Security)**.** For a batched IBE scheme Π_{BatchIBE}, we define the *selective* security game exactly as we defined the adaptive security game in Definition 3.2, except we require that the adversary declare the challenge identity id^* and the challenge batch label tg^* at the beginning of the security game (i.e., after seeing the public parameters pp output by Setup, but before seeing the master public key output by KeyGen). Then, we say that Π_{BatchIBE} is selectively secure if for all polynomials $B = B(\lambda)$ and all efficient and admissible adversaries $\mathcal{A}$, the adversary's advantage in the selective security game (i.e., the analog of Eq. (3.1)) is negligible. We say that Π_{BatchIBE} is selectively secure for a specific batch size $B = B(\lambda)$ if this holds for the specific function B.

Remark 3.4 (Adaptive Security via Complexity Leveraging)**.** Our notion of selective security essentially coincides with the usual notion of selective security for vanilla identity-based encryption [CHK03, BB04] (where the adversary has to declare its challenge identity at the beginning of the security game). As in the standard case of IBE, we can achieve full adaptive security by complexity leveraging and relying on sub-exponential hardness of the underlying computational assumption. With complexity leveraging, the reduction algorithm would guess the challenge identity id^* and the challenge batch label tg^* at the beginning of the security game.

4 Batched Identity-Based Encryption

In this section, we give our construction of batched identity-based encryption scheme [AFP25] from bilinear maps. We prove security from a q-type assumption over prime-order asymmetric pairing groups. Our assumption is a variant of the bilinear Diffie-Hellman exponent assumption from [BBG05, BGW05]. In Appendix B, we show this assumption holds in the standard generic bilinear group model [Sho97, BBG05]. We now state the assumption we use and then give our construction and security analysis.

Assumption 4.1 (N-Bilinear Diffie-Hellman Exponent Variant)**.** Let GroupGen be a prime-order bilinear group generator. For a security parameter λ, a parameter $N \in \mathbb{N}$, and a bit $\beta \in \{0,1\}$, we define the distribution $\mathcal{D}_{\lambda,N,\beta}$ as follows:

- Sample $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$. Sample exponents $a, b, s, \tau \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. Define

$$\mathsf{params} = \left(\begin{array}{c} 1^\lambda, \mathcal{G}, [b]_1, [s]_1, [\tau]_1, [ab]_1, [ab\tau]_1, [abs\tau]_1, \\ [a]_2, [b]_2, [\tau]_2, \ldots, [\tau^N]_2, [ab\tau]_2, \ldots, [ab\tau^N]_2, \end{array} \right). \tag{4.1}$$

- If $\beta = 0$, let $z = abs \in \mathbb{Z}_p$ and if $\beta = 1$, sample $z \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. Output $(\mathsf{params}, [z]_\mathsf{T})$.

We say Assumption 4.1 holds with respect to GroupGen and parameter $N = N(\lambda)$ if the distributions $\mathcal{D}_0 = \{\mathcal{D}_{\lambda,N(\lambda),0}\}_{\lambda\in\mathbb{N}}$ and $\mathcal{D}_1 = \{\mathcal{D}_{\lambda,N(\lambda),1}\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable.

Construction 4.2 (Batched Identity-Based Encryption)**.** Let GroupGen be a prime-order bilinear group generator. We construct a batched IBE scheme $\Pi_{\mathsf{BatchIBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Digest}, \mathsf{ComputeKey}, \mathsf{Decrypt})$ as follows:

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter λ, the setup algorithm samples $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and outputs the public parameters $\mathsf{pp} = \mathcal{G}$. The message space associated with pp is $\mathbb{G}_\mathsf{T}$, the identity space is $\mathbb{Z}_p$, and the batch label space is $\mathbb{Z}_p$.

- $\mathsf{KeyGen}(\mathsf{pp}, 1^B)$: On input the public parameters $\mathsf{pp} = \mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e)$ and a bound on the batch size B, the key-generation algorithm samples exponents $\tau, w, v, h, \alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and outputs the master public key

$$\mathsf{mpk} = \left(\mathcal{G},\ [\tau]_1,\ [\tau]_2,\ [\tau^2]_2,\ \ldots,\ [\tau^B]_2,\ [w]_1,\ [w\tau]_1,\ [v]_1,\ [h]_1,\ [\alpha]_\mathsf{T}\right) \tag{4.2}$$

  and the master secret key $\mathsf{msk} = (w, v, h, \alpha)$.

- $\mathsf{Encrypt}(\mathsf{mpk}, [m]_\mathsf{T}, \mathsf{id}, \mathsf{tg})$: On input the master public key mpk (parsed according to Eq. (4.2)), a message $[m]_\mathsf{T} \in \mathbb{G}_\mathsf{T}$, an identity $\mathsf{id} \in \mathbb{Z}_p$, and a batch label $\mathsf{tg} \in \mathbb{Z}_p$, the encryption algorithm samples $s \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. It then outputs the ciphertext

$$\begin{aligned}\mathsf{ct} &= \left([s]_1,\ s[w\tau]_1 - (s\cdot\mathsf{id})[w]_1,\ s([v]_1 + \mathsf{tg}\cdot[h]_1),\ s[\alpha]_\mathsf{T} + [m]_\mathsf{T}\right)\\ &= \left([s]_1,\ [sw(\tau-\mathsf{id})]_1,\ [s(v+h\cdot\mathsf{tg})]_1,\ [s\alpha]_\mathsf{T} + [m]_\mathsf{T}\right).\end{aligned}$$

- $\mathsf{Digest}(\mathsf{mpk}, S)$: On input the master public key mpk (parsed according to Eq. (4.2)) and a set of identities $S \subseteq \mathbb{Z}_p$ where $|S| \le B$, the digest algorithm defines the polynomial $F_S(x) = \prod_{\mathsf{id}\in S}(x - \mathsf{id})$ whose roots are the identities $\mathsf{id} \in S$. Write $F_S(x) = \sum_{i\in[0,|S|]} f_i x^i$. Output the digest

$$\mathsf{dig} = \sum_{i\in[0,|S|]} f_i \cdot [\tau^i]_2 = [F_S(\tau)]_2.$$

- $\mathsf{ComputeKey}(\mathsf{msk}, \mathsf{dig}, \mathsf{tg})$: On input the master secret key $\mathsf{msk} = (w, v, h, \alpha)$, a digest $\mathsf{dig} = [d]_2$, and a batch label $\mathsf{tg} \in \mathbb{Z}_p$, the key-computation algorithm samples random $r \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and $y \xleftarrow{\mathrm{R}} \mathbb{Z}_p^*$ and outputs the secret key

$$\mathsf{sk} = \left(y,\ [r]_2,\ [\alpha + r(v + h\cdot\mathsf{tg})]_2 + yw\cdot[d]_2\right).$$

- $\mathsf{Decrypt}(\mathsf{mpk}, \mathsf{sk}, S, (\mathsf{id}, \mathsf{tg}), \mathsf{ct})$: On input the master public key mpk (parsed according to Eq. (4.2)), a secret key $\mathsf{sk} = (y, [u_1]_2, [u_2]_2)$, the set of identities $S \subseteq \mathbb{Z}_p$, an identity $\mathsf{id} \in S$, a batch label $\mathsf{tg} \in \mathbb{Z}_p$, and the ciphertext $\mathsf{ct} = ([\mathrm{ct}_1]_1, [\mathrm{ct}_2]_1, [\mathrm{ct}_3]_1, [\mathrm{ct}_4]_\mathsf{T})$, the decryption algorithm proceeds as follows:
  - First, it defines the polynomial
$$F_{S\setminus\{\mathsf{id}\}}(x) = \prod_{\mathsf{id}'\in S\setminus\{\mathsf{id}\}} (x - \mathsf{id}').$$
    Compute $[F_{S\setminus\{\mathsf{id}\}}(\tau)]_2 = \sum_{i\in[0,|S|-1]} f_i[\tau^i]_2$, where $F_{S\setminus\{\mathsf{id}\}}(x) = \sum_{i\in[0,|S|-1]} f_i x^i$.
  - Then it computes and outputs
$$[\mathrm{ct}_4]_\mathsf{T} - \left(([\mathrm{ct}_1]_1 \cdot [u_2]_2) - (y\cdot[\mathrm{ct}_2]_1 \cdot [F_{S\setminus\{\mathsf{id}\}}(\tau)]_2) - ([\mathrm{ct}_3]_1 \cdot [u_1]_2)\right). \tag{4.3}$$

Theorem 4.3 (Correctness)**.** *Construction 4.2 is correct.*

Proof. Take any $\lambda, B \in \mathbb{N}$ and any $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e)$ in the support of $\mathsf{GroupGen}(1^\lambda)$. Take any $[m]_\mathsf{T} \in \mathbb{G}_\mathsf{T}$, any $\mathsf{id}^* \in \mathbb{Z}_p$, batch label $\mathsf{tg}^* \in \mathbb{Z}_p$, set $S \subseteq \mathbb{Z}_p$ of size at most B where $\mathsf{id}^* \in S$. Sample $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, 1^B)$ and $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, [m]_\mathsf{T}, \mathsf{id}^*, \mathsf{tg}^*)$. Compute $\mathsf{dig} = \mathsf{Digest}(\mathsf{mpk}, S)$ and $\mathsf{sk} = \mathsf{ComputeKey}(\mathsf{msk}, \mathsf{dig}, \mathsf{tg}^*)$. By construction, this means

$$\begin{aligned}\mathsf{mpk} &= \left(\mathcal{G}, [\tau]_1, [\tau]_2, [\tau^2]_2, \ldots, [\tau^B]_2, [w]_1, [w\tau]_1, [v]_1, [h]_1, [\alpha]_\mathsf{T}\right)\\ \mathsf{ct} &= \left([s]_1, [sw(\tau-\mathsf{id}^*)]_1, [s(v+h\cdot\mathsf{tg}^*)]_1, [s\alpha]_\mathsf{T} + [m]_\mathsf{T}\right)\\ \mathsf{sk} &= \left(y, [r]_2, [\alpha + yw\cdot F_S(\tau) + (v+h\cdot\mathsf{tg}^*)r]_2\right),\end{aligned}$$

where $F_S(x) = \prod_{\mathsf{id}\in S}(x - \mathsf{id})$. Consider now $\mathsf{Decrypt}(\mathsf{mpk}, \mathsf{sk}, S, (\mathsf{id}^*, \mathsf{tg}^*), \mathsf{ct})$. If we write $\mathsf{sk} = (y, [u_1]_2, [u_2])$ and $\mathsf{ct} = ([\mathsf{ct}_1]_1, [\mathsf{ct}_2]_1, [\mathsf{ct}_3]_1, [\mathsf{ct}_4]_\mathsf{T})$, then the decryption algorithm computes

$$\begin{aligned}
\mathsf{ct}_1 \cdot u_2 &= \alpha s + syw \cdot F_S(\tau) + rs(v + h \cdot \mathsf{tg}^*) \\
y \cdot \mathsf{ct}_2 \cdot F_{S\setminus\{\mathsf{id}^*\}}(\tau) &= syw(\tau - \mathsf{id}^*) \cdot \prod_{\mathsf{id}\in S\setminus\{\mathsf{id}^*\}} (\tau - \mathsf{id}) = syw \cdot \prod_{\mathsf{id}\in S}(\tau - \mathsf{id}) = syw \cdot F_S(\tau) \\
\mathsf{ct}_3 \cdot u_1 &= rs(v + h \cdot \mathsf{tg}^*).
\end{aligned}$$

This means

$$\mathsf{ct}_1 \cdot u_2 - y \cdot \mathsf{ct}_2 \cdot F_{S\setminus\{\mathsf{id}^*\}}(\tau) - \mathsf{ct}_3 \cdot u_1 = \alpha s + syw \cdot F_S(\tau) + rs(v + h \cdot \mathsf{tg}^*) - syw \cdot F_S(\tau) - rs(v + h \cdot \mathsf{tg}^*) = \alpha s.$$

The decryption relation (Eq. (4.3)) now yields:

$$[\mathsf{ct}_4 - (\mathsf{ct}_1 \cdot u_2 - y \cdot \mathsf{ct}_2 \cdot F_{S\setminus\{\mathsf{id}^*\}}(\tau) - \mathsf{ct}_3 \cdot u_1)]_\mathsf{T} = [s\alpha + m - \alpha s]_\mathsf{T} = [m]_\mathsf{T}$$

and correctness holds. □

Theorem 4.4 (Selective Security). *Take any polynomial $B = B(\lambda)$ and suppose Assumption 4.1 with parameter B holds with respect to* GroupGen. *Then, Construction 4.2 is selectively secure for batch size B.*

Proof. Let $\mathcal{A}$ be an efficient and admissible adversary for the selective security experiment for Construction 4.2 with batch size B. We define a simple sequence of hybrid experiments, each indexed by a bit $\beta \in \{0, 1\}$:

- $\mathsf{Hyb}_0^{(\beta)}$: This is the selective batched IBE security game with bit β.
- $\mathsf{Hyb}_1^{(\beta)}$: Same as $\mathsf{Hyb}_0^{(\beta)}$, except when constructing the challenge ciphertext $\mathsf{ct} = ([\mathsf{ct}_1]_1, [\mathsf{ct}_2]_1, [\mathsf{ct}_3]_1, [\mathsf{ct}_4]_\mathsf{T})$, the challenger samples $\mathsf{ct}_4 \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. In this experiment, the adversary's view is independent of the message.

Let $\mathsf{Hyb}_1^{(0)}(\mathcal{A})$ and $\mathsf{Hyb}_1^{(1)}(\mathcal{A})$ denote the output distribution of an execution of experiment $\mathsf{Hyb}_1^{(0)}$ and $\mathsf{Hyb}_1^{(1)}$ with adversary $\mathcal{A}$, respectively. By construction, experiments $\mathsf{Hyb}_1^{(0)}$ and $\mathsf{Hyb}_1^{(1)}$ are identical so it suffices to show that the outputs of $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$ are computationally indistinguishable for each $\beta \in \{0, 1\}$. Suppose there exists a $\beta \in \{0, 1\}$ and a non-negligible ε such that

$$\left|\Pr[\mathsf{Hyb}_0^{(\beta)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(\beta)}(\mathcal{A}) = 1]\right| \geq \varepsilon.$$

Then, we use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks Assumption 4.1 with parameter B and the same advantage ε:

1. At the beginning of the game, algorithm $\mathcal{B}$ receives a challenge $(\mathsf{params}, [z]_\mathsf{T})$ where

$$\mathsf{params} = \left(\begin{array}{c} 1^\lambda, \mathcal{G}, [b]_1, [s]_1, [\hat{\tau}]_1, [ab]_1, [ab\hat{\tau}]_1, [abs\hat{\tau}]_1, \\ [a]_2, [b]_2, [\hat{\tau}]_2, \ldots, [\hat{\tau}^B]_2, [ab\hat{\tau}]_2, \ldots, [ab\hat{\tau}^B]_2, \end{array} \right).$$

   and either $z = abs$ or $z \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. We use $\hat{\tau}$ to denote the powers-of-τ that appear in the assumption since the reduction algorithm below will program the challenge identity id^* into the simulated powers-of-τ.

2. Algorithm $\mathcal{B}$ checks if $[b]_1 = [0]_1$. If so, it outputs 1 if $[z]_\mathsf{T} = [0]_\mathsf{T}$ and 0 otherwise.

3. Algorithm $\mathcal{B}$ sets $\mathsf{pp} = \mathcal{G}$ and gives pp to $\mathcal{A}$. Algorithm $\mathcal{A}$ now commits to a challenge identity $\mathsf{id}^* \in \mathbb{Z}_p$ and label $\mathsf{tg}^* \in \mathbb{Z}_p$. Algorithm $\mathcal{B}$ now constructs the public key as follows. In the following description, we will use a "tilde" (e.g., $\tilde{\alpha}, \tilde{v}$) to denote an exponent that is chosen by (or otherwise known to) the reduction algorithm.

   - Algorithm $\mathcal{B}$ implicitly sets $\tau = \hat{\tau} + \mathsf{id}^*$. For each $i \in [B]$, algorithm $\mathcal{B}$ computes

$$[\tau^i]_2 = \sum_{j\in[0,i]} \binom{i}{j} \cdot (\mathsf{id}^*)^{i-j} \cdot [\hat{\tau}^j]_2 = [(\hat{\tau} + \mathsf{id}^*)^i]_2.$$

     Similarly, it sets $[\tau]_1 = [\hat{\tau}]_1 + [\mathsf{id}^*]_1 = [\hat{\tau} + \mathsf{id}^*]_1$.

- Algorithm $\mathcal{B}$ samples $\tilde{\alpha} \xleftarrow{R} \mathbb{Z}_p$ and implicitly sets $\alpha = \tilde{\alpha} - ab$:

$$[\alpha]_T = [\tilde{\alpha}]_T - [ab]_1 \cdot [1]_2 = [\tilde{\alpha} - ab]_T.$$

- Algorithm $\mathcal{B}$ samples $\tilde{y}^* \xleftarrow{R} \mathbb{Z}_p^*$ and implicitly sets $w = \tilde{y}^* ab$:

$$[w]_1 = \tilde{y}^* \cdot [ab]_1 = [\tilde{y}^* ab]_1$$
$$[w\tau]_1 = \tilde{y}^* \cdot ([ab\hat{\tau}]_1 + \mathsf{id}^* \cdot [ab]_1).$$

- Algorithm $\mathcal{B}$ samples $\tilde{v}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$ and implicitly sets $v = \tilde{v} - b \cdot \mathsf{tg}^*$ and $h = \tilde{h} + b$. Concretely, algorithm $\mathcal{B}$ defines

$$[v]_1 = [\tilde{v}]_1 - \mathsf{tg}^* \cdot [b]_1 = [\tilde{v} - b \cdot \mathsf{tg}^*]_1$$
$$[h]_1 = [\tilde{h}]_1 + [b]_1 = [\tilde{h} + b]_1.$$

Algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with the master public key

$$\mathsf{mpk} = (\mathcal{G}, [\tau]_1, [\tau]_2, \ldots, [\tau^B]_2, [w]_1, [w\tau]_1, [v]_1, [h]_1, [\alpha]_T).$$

4. When algorithm $\mathcal{A}$ makes a key-computation query on a set of identities $S \subseteq \mathbb{Z}_p$ where $|S| \leq B$ and a batch label $\mathsf{tg} \in \mathbb{Z}_p$, algorithm $\mathcal{B}$ defines the following two polynomials over $\mathbb{Z}_p$:

$$F_S(x) = \prod_{\mathsf{id} \in S} (x - \mathsf{id})$$
$$G_S(x) = F_S(x + \mathsf{id}^*) - F_S(\mathsf{id}^*). \tag{4.4}$$

Write $F_S(x) = \sum_{i=[0,|S|]} \tilde{f}_i x^i$. Next, observe that the constant term of $G_S(x)$ is $G_S(0) = F_S(\mathsf{id}^*) - F_S(\mathsf{id}^*) = 0$. This means $G_S(x) = \sum_{i \in [|S|]} \tilde{g}_i x^i$. Algorithm $\mathcal{B}$ now proceeds as follows:

- If $\mathsf{tg} \neq \mathsf{tg}^*$, algorithm $\mathcal{B}$ samples $\tilde{y} \xleftarrow{R} \mathbb{Z}_p^*$ and $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$. Then, algorithm $\mathcal{B}$ computes the following

$$[u_1]_2 = [\tilde{r}]_2 + (\mathsf{tg} - \mathsf{tg}^*)^{-1}(1 - \tilde{y}\tilde{y}^* \cdot F_S(\mathsf{id}^*)) \cdot [a]_2$$
$$[u_2]_2 = [\tilde{\alpha}]_2 + \tilde{r} \cdot (\mathsf{tg} - \mathsf{tg}^*) \cdot [b]_2 + (\tilde{v} + \tilde{h} \cdot \mathsf{tg}) \cdot [u_1]_2 + \sum_{i \in [|S|]} \tilde{g}_i \tilde{y}\tilde{y}^* \cdot [ab\hat{\tau}^i]_2.$$

- If $\mathsf{tg} = \mathsf{tg}^*$, algorithm $\mathcal{B}$ first sets $\tilde{y} = 1/(\tilde{y}^* F_S(\mathsf{id}^*))$. Note that this is well-defined since $\tilde{y}^* \in \mathbb{Z}_p^*$ and when $\mathsf{tg} = \mathsf{tg}^*$, it must be the case that $\mathsf{id}^* \notin S$, so $F_S(\mathsf{id}^*) \neq 0$ by definition of F_S. Next, algorithm $\mathcal{B}$ samples $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$ and sets

$$[u_1]_2 = [\tilde{r}]_2$$
$$[u_2]_2 = [\tilde{\alpha} + \tilde{r}(\tilde{v} + \tilde{h} \cdot \mathsf{tg}^*)]_2 + \sum_{i \in [|S|]} \tilde{g}_i \tilde{y}\tilde{y}^* \cdot [ab\hat{\tau}^i]_2.$$

In both cases, algorithm $\mathcal{B}$ responds to $\mathcal{A}$ with the secret key $\mathsf{sk} = (\tilde{y}, [u_1]_2, [u_2]_2)$.

5. In the challenge phase, algorithm $\mathcal{A}$ outputs two messages $[m_0]_T$ and $[m_1]_T$. Algorithm $\mathcal{B}$ computes the following:

$$[\mathsf{ct}_1]_1 = [s]_1$$
$$[\mathsf{ct}_2]_1 = \tilde{y}^* \cdot [abs\hat{\tau}]_1$$
$$[\mathsf{ct}_3]_1 = (\tilde{v} + \tilde{h} \cdot \mathsf{tg}^*) \cdot [s]_1$$
$$[\mathsf{ct}_4]_T = \tilde{\alpha} \cdot [s]_1 \cdot [1]_2 - [z]_T + [m_\beta]_T$$

Algorithm $\mathcal{B}$ responds with the ciphertext $\mathsf{ct} = ([\mathsf{ct}_1]_1, [\mathsf{ct}_2]_1, [\mathsf{ct}_3]_1, [\mathsf{ct}_4]_T)$.

6. Algorithm $\mathcal{A}$ can continue to make key-computation queries. Algorithm $\mathcal{B}$ responds as described above.

7. At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $\beta' \in \{0,1\}$, which algorithm $\mathcal{B}$ also outputs.

If the challenger samples $b \xleftarrow{R} \mathbb{Z}_p$ and $b = 0$ (which happens with probability $1/p$), then algorithm $\mathcal{B}$ outputs 1 with probability 1 when $z = abs$ and with probability $1/p$ when $z \xleftarrow{R} \mathbb{Z}_p$. It suffices to show that algorithm $\mathcal{B}$ achieve non-negligible distinguishing advantage when $b \neq 0$. We show in this case that depending on the distribution of the challenge element z, algorithm $\mathcal{A}$ either perfectly simulates an execution of $\mathsf{Hyb}_0^{(\beta)}$ or $\mathsf{Hyb}_1^{(\beta)}$. We first consider the distribution of the public parameters. By construction, algorithm $\mathcal{B}$ constructs the public parameters by implicitly setting

$$\begin{array}{lll} \tau = \hat{\tau} + \mathsf{id}^* & \alpha = \tilde{\alpha} - ab & v = \tilde{v} - b \cdot \mathsf{tg}^* \\ & w = \tilde{y}^* ab & h = \tilde{h} + b \end{array} \tag{4.5}$$

Since the challenger samples $\hat{\tau}, a \xleftarrow{R} \mathbb{Z}_p$ and $b \neq 0$ and algorithm $\mathcal{B}$ samples $\tilde{\alpha}, \tilde{v}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$, $\tilde{y}^* \xleftarrow{R} \mathbb{Z}_p^*$, the distribution of τ, w, v, h, α are all independent and uniform over $\mathbb{Z}_p$, exactly as in the real scheme. Moreover, the public parameters perfectly hide the value of $\tilde{y}^*$. Next, consider the components of the challenge ciphertext. We claim that algorithm $\mathcal{B}$ generates the challenge ciphertext according to the specification of $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$ where the encryption randomness $s \xleftarrow{R} \mathbb{Z}_p$ is the corresponding exponent sampled by the challenger. We consider each component separately:

- By construction, algorithm $\mathcal{B}$ sets $\mathsf{ct}_1 = s$ which matches the distribution in $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$.
- Consider ct_2. In the reduction, algorithm $\mathcal{B}$ implicitly sets $\tau = \hat{\tau} + \mathsf{id}^*$ and $w = \tilde{y}^* ab$. Now, in $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$, the experiment would set $\mathsf{ct}_2 = sw(\tau - \mathsf{id}^*)$. Substituting in algorithm $\mathcal{B}$'s choice of w and τ, we have

  $$\mathsf{ct}_2 = sw(\tau - \mathsf{id}^*) = s(\tilde{y}^* ab)(\hat{\tau} + \mathsf{id}^* - \mathsf{id}^*) = \tilde{y}^* \cdot abs\hat{\tau},$$

  which coincides with how $\mathcal{B}$ constructs ct_2 in the reduction.
- Consider ct_3. In $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$, the experiment would set $\mathsf{ct}_3 = s(v + h \cdot \mathsf{tg}^*)$. Substituting in algorithm $\mathcal{B}$'s choice of v and h, we have

  $$\mathsf{ct}_3 = s(v + h \cdot \mathsf{tg}^*) = s(\tilde{v} - b \cdot \mathsf{tg}^*) + s(\tilde{h} + b) \cdot \mathsf{tg}^* = s(\tilde{v} + \tilde{h} \cdot \mathsf{tg}^*),$$

  which is precisely how algorithm $\mathcal{B}$ constructs ct_3 in the reduction.
- Finally, consider the distribution of ct_4. We consider two possibilities depending on the distribution of z:
  - Suppose $z = abs$. In the reduction, algorithm $\mathcal{B}$ implicitly sets $\alpha = \tilde{\alpha} - ab$ so

    $$\mathsf{ct}_4 = \tilde{\alpha} s - z + m_\beta = \tilde{\alpha} s - abs + m_\beta = s\alpha + m_\beta,$$

    which is precisely the distribution of ct_4 in $\mathsf{Hyb}_0^{(\beta)}$.
  - Suppose $z \xleftarrow{R} \mathbb{Z}_p$. In this case, the distribution of ct_4 is uniform over $\mathbb{Z}_p$. This is the distribution of ct_4 in $\mathsf{Hyb}_1^{(\beta)}$.

We conclude that depending on the distribution of z, the challenge ciphertext in the reduction is distributed either according to the specification of $\mathsf{Hyb}_0^{(\beta)}$ or the specification of $\mathsf{Hyb}_1^{(\beta)}$. To complete the proof, it thus suffices to consider the key-computation queries. Suppose $\mathcal{A}$ makes a key-computation query on a set of identities $S \subseteq \mathbb{Z}_p$ and a batch label $\mathsf{tg} \in \mathbb{Z}_p$. As in the reduction, we consider two cases:

- Suppose $\mathsf{tg} \neq \mathsf{tg}^*$. In this case, algorithm $\mathcal{B}$ samples $\tilde{y} \xleftarrow{R} \mathbb{Z}_p^*$ and implicitly sets the randomness r to be

  $$r = u_1 = \tilde{r} + (\mathsf{tg} - \mathsf{tg}^*)^{-1}(1 - \tilde{y}\tilde{y}^* \cdot F_S(\mathsf{id}^*)) \cdot a,$$

where $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$. Since $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$, the distribution of r coincides with the distribution in $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$. Thus, it suffices to argue that the component u_2 is correctly constructed (with respect to algorithm $\mathcal{B}$'s choice of r and $\tilde{y}$). In $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$, the experiment would first compute the digest $\mathsf{dig} = [d]_2 = [F_S(\tau)]_2$ and then set

$$u_2 = \alpha + r(v + h \cdot \mathsf{tg}) + \tilde{y}w \cdot F_S(\tau). \tag{4.6}$$

In the reduction, algorithm $\mathcal{B}$ implicitly sets $\tau = \hat{\tau} + \mathsf{id}^*$. Thus, we can write

$$\begin{aligned} F_S(\tau) &= F_S(\mathsf{id}^*) + F_S(\tau) - F_S(\mathsf{id}^*) \\ &= F_S(\mathsf{id}^*) + F_S(\hat{\tau} + \mathsf{id}^*) - F_S(\mathsf{id}^*) \\ &= F_S(\mathsf{id}^*) + G_S(\hat{\tau}), \end{aligned}$$

where G_S is the polynomial from Eq. (4.4). By definition of the coefficients $\tilde{g}_i$ and using the fact that algorithm $\mathcal{B}$ implicitly defines $w = \tilde{y}^* ab$, we can write

$$\sum_{i \in [|S|]} \tilde{g}_i \tilde{y} \tilde{y}^* (ab\hat{\tau}^i) = \tilde{y}\tilde{y}^* abG_S(\hat{\tau}) = \tilde{y}w \cdot G_S(\hat{\tau}).$$

Now, in the reduction, algorithm $\mathcal{B}$ sets

$$u_2 = \tilde{\alpha} + \tilde{r}b(\mathsf{tg} - \mathsf{tg}^*) + u_1(\tilde{v} + \tilde{h} \cdot \mathsf{tg}) + \tilde{y}w \cdot G_S(\hat{\tau}). \tag{4.7}$$

Suppose now that we substitute the values of α, r, v, h, w from the reduction (see Eq. (4.5)) into Eq. (4.6). Then we have the following:

$$\begin{aligned} u_2 &= \alpha + r(v + h \cdot \mathsf{tg}) + \tilde{y}w \cdot F_S(\tau) \\ &= \tilde{\alpha} - ab + u_1(v + h \cdot \mathsf{tg}) + \tilde{y}w(F_S(\mathsf{id}^*) + G_S(\hat{\tau})) \\ &= \tilde{\alpha} + u_1(v + h \cdot \mathsf{tg}) + \tilde{y}w \cdot G_S(\hat{\tau}) - ab(1 - \tilde{y}\tilde{y}^* \cdot F_S(\mathsf{id}^*)). \end{aligned} \tag{4.8}$$

Consider now the value of $u_1(v + h \cdot \mathsf{tg})$:

$$\begin{aligned} u_1(v + h \cdot \mathsf{tg}) &= u_1(\tilde{v} - b \cdot \mathsf{tg}^* + (\tilde{h} + b) \cdot \mathsf{tg}) \\ &= u_1(\tilde{v} + \tilde{h} \cdot \mathsf{tg}) + u_1 b(\mathsf{tg} - \mathsf{tg}^*) \\ &= u_1(\tilde{v} + \tilde{h} \cdot \mathsf{tg}) + (\tilde{r} + (\mathsf{tg} - \mathsf{tg}^*)^{-1}(1 - \tilde{y}\tilde{y}^* \cdot F_S(\mathsf{id}^*)) \cdot a)b(\mathsf{tg} - \mathsf{tg}^*) \\ &= u_1(\tilde{v} + \tilde{h} \cdot \mathsf{tg}) + \tilde{r}b(\mathsf{tg} - \mathsf{tg}^*) + ab(1 - \tilde{y}\tilde{y}^* \cdot F_S(\mathsf{id}^*)). \end{aligned}$$

Observe that the highlighted term in green precisely cancels out the corresponding term in Eq. (4.8). Thus, substituting back in Eq. (4.8), we now have

$$\begin{aligned} u_2 &= \tilde{\alpha} + u_1(v + h \cdot \mathsf{tg}) + \tilde{y}wG_S(\hat{\tau}) - ab(1 - \tilde{y}\tilde{y}^* \cdot F_S(\mathsf{id}^*)) \\ &= \tilde{\alpha} + u_1(\tilde{v} + \tilde{h} \cdot \mathsf{tg}) + \tilde{r}b(\mathsf{tg} - \mathsf{tg}^*) + \tilde{y}w \cdot G_S(\hat{\tau}) \end{aligned}$$

This is precisely the expression in Eq. (4.7) so we conclude that algorithm $\mathcal{B}$ answers the key-computation query according to the specification of $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$.

- Suppose $\mathsf{tg} = \mathsf{tg}^*$. In this case, algorithm $\mathcal{B}$ sets $\tilde{y} = 1/(\tilde{y}^* F_S(\mathsf{id}^*))$ and samples $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$. By assumption, algorithm $\mathcal{A}$ makes at most one key-computation query on $\mathsf{tg} = \mathsf{tg}^*$. As shown above, the adversary's view can be described entirely as a function of the exponents (τ, α, w, v, h) from the public parameters and the exponent s from the challenge ciphertext. As argued above, the exponents in the public key perfectly hide $\tilde{y}^*$ and since s is independent of $\tilde{y}^*$, we conclude that that the view of adversary $\mathcal{A}$ before this point is independent of $\tilde{y}^*$. Since algorithm $\mathcal{B}$ samples $\tilde{y}^* \xleftarrow{R} \mathbb{Z}_p^*$, the distribution of $\tilde{y} = 1/(\tilde{y}^* F_S(\mathsf{id}^*))$ is thus uniform over $\mathbb{Z}_p^*$. Thus, the distribution of $(\tilde{r}, \tilde{y})$ is correctly distributed.

As in the previous case, it suffices now to show that the component u_2 is correctly computed (with respect to algorithm $\mathcal{B}$'s choice of r and $\tilde{y}$). As in the previous case, in $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$, the experiment would set

$$u_2 = \alpha + \tilde{r}(v + h \cdot \mathrm{tg}^*) + \tilde{y}w \cdot F_S(\tau). \tag{4.9}$$

Suppose now that we substitute the values of $\alpha, v, h, w, \tilde{y}$ from the reduction (see Eq. (4.5)) into Eq. (4.9). Then we have the following:

$$\begin{aligned} u_2 &= \alpha + \tilde{r}(v + h \cdot \mathrm{tg}^*) + \tilde{y}w \cdot F_S(\tau) \\ &= \tilde{\alpha} - ab + \tilde{r}(\tilde{v} - b \cdot \mathrm{tg}^* + (\tilde{h} + b) \cdot \mathrm{tg}^*) + \tilde{y}\tilde{y}^* ab(F_S(\mathrm{id}^*) + G_S(\hat{\tau})) \\ &= \tilde{\alpha} + \tilde{r}(\tilde{v} + \tilde{h} \cdot \mathrm{tg}^*) + \tilde{y}\tilde{y}^* ab \cdot G_S(\hat{\tau}) - ab(1 - \tilde{y}\tilde{y}^* F_S(\mathrm{id}^*)) \\ &= \tilde{\alpha} + \tilde{r}(\tilde{v} + \tilde{h} \cdot \mathrm{tg}^*) + \tilde{y}\tilde{y}^* ab \cdot G_S(\hat{\tau}), \end{aligned} \tag{4.10}$$

where the final equality uses the critical cancellation that $\tilde{y} = 1/(\tilde{y}^* F_S(\mathrm{id}^*))$ so $\tilde{y}\tilde{y}^* F_S(\mathrm{id}^*) = 1$. Now, in the reduction, algorithm $\mathcal{B}$ sets

$$u_2 = \tilde{\alpha} + \tilde{r}(\tilde{v} + \tilde{h} \cdot \mathrm{tg}^*) + \sum_{i \in [|S|]} \tilde{g}_i \tilde{y}\tilde{y}^* ab\hat{\tau}^i = \tilde{\alpha} + \tilde{r}(\tilde{v} + \tilde{h} \cdot \mathrm{tg}^*) + \tilde{y}\tilde{y}^* ab \cdot G_S(\hat{\tau}),$$

which precisely coincides with Eq. (4.10). We conclude that algorithm $\mathcal{B}$ answers the key-computation query according to the specification of $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$.

We conclude that algorithm $\mathcal{B}$ responds to the key-generation queries with the same procedure as in $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$. Thus, as argued above, if $z = abs$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_0^{(\beta)}$, whereas if $z \xleftarrow{R} \mathbb{Z}_p$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_1^{(\beta)}$. Thus, when $b \neq 0$, algorithm $\mathcal{B}$ breaks Assumption 4.1 with the same advantage ε. The claim follows. □

Corollary 4.5 (Batched Identity-Based Encryption)**.** *Let λ be a security parameter. Suppose Assumption 4.1 holds with respect to* GroupGen *for all polynomials $B = B(\lambda)$. Then, for every polynomial $B = B(\lambda)$, Construction 4.2 is a selectively-secure batched IBE scheme with the following efficiency properties:*

- **Public key size:** *For a batch size B, the public key contains 5 $\mathbb{G}_1$ elements, B $\mathbb{G}_2$ elements, and 1 $\mathbb{G}_\mathsf{T}$ element.*
- **Ciphertext size:** *Each ciphertext contains 3 $\mathbb{G}_1$ elements and 1 $\mathbb{G}_\mathsf{T}$ element.*
- **Digest size:** *A digest contains 1 $\mathbb{G}_2$ element.*
- **Decryption key size:** *A decryption key contains 2 $\mathbb{G}_2$ elements and 1 $\mathbb{Z}_p$ element.*

Adaptive security. As noted in Remark 3.4, we can lift the selectively-secure batched IBE scheme from Corollary 4.5 to an adaptively-secure scheme using complexity leveraging and relying on sub-exponential hardness of Assumption 4.1. In Appendix C, we describe how to extend Corollary 4.5 to obtain a variant that allows the adversary to *adaptively* choose the challenge identity id^* in the security game (see Construction C.3 and Remark C.15), but which is still selective in the challenge batch label tg^*. Security here only relies on polynomial-hardness of a q-type assumption in the plain model. In Remark C.16, we describe a plausible approach to achieve full adaptive security (essentially, by using Waters' technique [Wat05] to embed the batch label).

Supporting multiple key-generation queries. Like many previous batched IBE and batched decryption schemes that use a batch label [CGPP24, SAA24, CGPW25, AFP25], Construction 4.2 only ensures security against adversaries that make a single key-computation query on the challenge batch label. In Appendix C, we describe a simple technique that allows the adversary to request up to K decryption keys for each batch label for any a priori bounded K. The modification requires adding $2(K-1)$ group elements to the public key, $K-1$ group elements to the ciphertext, and $K-1$ field elements to the decryption key. In particular, we essentially replace the scalar $w \in \mathbb{Z}_p$ in the public key and the scalar $y \in \mathbb{Z}_p$ in the secret key with vectors $\mathbf{w} \in \mathbb{Z}_p^K$ and $\mathbf{y} \in \mathbb{Z}_p^K$.

5 Threshold Batched IBE with Silent Setup

In this section, we show how to integrate our batched IBE scheme from Section 4 (Construction 4.2) with the approach from [WW25a] for building threshold (non-batched) IBE with silent setup to obtain a threshold batched IBE scheme with silent setup. The recent work of [BCF+25] give a construction (obtained by integrating ideas from [BFOQ25] with the threshold encryption scheme with silent setup from [GKPW24]) that supports a polynomial-size identity space in the generic bilinear group model; ciphertexts in their scheme contain $O(\lambda/\log\lambda)$ group elements. Our construction preserves many of the features of our vanilla batched IBE scheme: it supports an exponential-size identity space, ciphertexts are just a constant number of group elements, and security can be based on a q-type assumption in the plain model.

Threshold batched IBE with silent setup. We start with the definition of threshold batched IBE with silent setup. Our definition is an adaptation of the concept of threshold batched encryption with silent setup from [BCF+25]. To simplify the exposition, we follow [WW25a] and work in the registered key model [RY07] where we only consider correctness and security for keys in the support of the honest key-generation algorithm (and moreover, in the case of security, the adversary must provide the randomness used to generate its keys). Previous works [RY07, WW25a] show that using simulation-sound non-interactive zero-knowledge (NIZK) proof of knowledge, we can generically lift any scheme with security in the registered-key model into one with security in the plain model. We now give the formal definition:

Definition 5.1 (Threshold Batched IBE with Silent Setup)**.** A threshold batched IBE scheme with silent setup $\Pi_{\mathsf{BatchSTIBE}}$ is a tuple of efficient algorithms $\Pi_{\mathsf{BatchSTIBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Preprocess}, \mathsf{Encrypt}, \mathsf{Digest}, \mathsf{CompKeyShare}, \mathsf{VerifyKeyShare}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^B, 1^N) \to \mathsf{pp}$: On input the security parameter λ, a bound on the batch size B, and a bound on the size of the decryption committee N, the setup algorithm outputs a set of public parameters pp. We assume that the public parameters (implicitly) specify the message space $\mathcal{M}$, identity space $\mathcal{I}$, and batch label space $\mathcal{T}$ for the encryption scheme.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk}, \mathsf{ht})$: On input the public parameters pp, the key-generation algorithm outputs a public key pk, a secret key sk, and an aggregation hint ht.
- $\mathsf{Preprocess}(\mathsf{pp}, (\mathsf{ht}_1, \ldots, \mathsf{ht}_L)) \to (\mathsf{ek}, \mathsf{ak})$: On input the public parameters pp and $L \le N$ aggregation hints $\mathsf{ht}_1, \ldots, \mathsf{ht}_L$, the preprocessing algorithm outputs an encryption key ek and an aggregation key ak. This algorithm is deterministic.
- $\mathsf{Encrypt}(\mathsf{ek}, m, \mathsf{id}, \mathsf{tg}, T) \to \mathsf{ct}$: On input the encryption key ek, a message $m \in \mathcal{M}$, an identity $\mathsf{id} \in \mathcal{I}$, a batch label $\mathsf{tg} \in \mathcal{T}$, and a threshold $T \le L$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Digest}(\mathsf{pp}, S) \to \mathsf{dig}$: On input the public parameters pp and a set of identities $S \subseteq \mathcal{I}$, the digest algorithm outputs a digest. This algorithm is deterministic.
- $\mathsf{CompKeyShare}(\mathsf{sk}, \mathsf{dig}, \mathsf{tg}) \to \sigma$: On input a secret key sk, a digest dig, and a batch label $\mathsf{tg} \in \mathcal{T}$, the key-share computation algorithm outputs a decryption key share σ.
- $\mathsf{VerifyKeyShare}(\mathsf{pk}, \mathsf{dig}, \mathsf{tg}, \sigma) \to 0/1$: On input a public key pk, a digest dig, a batch label $\mathsf{tg} \in \mathcal{T}$, and a decryption key share σ, the key-share verification algorithm outputs a bit indicating whether the decryption key share σ is valid under pk for dig and tg. This algorithm is deterministic.
- $\mathsf{Decrypt}(\mathsf{ak}, \{\sigma_\ell\}_{\ell \in U}, S, (\mathsf{id}, \mathsf{tg}), \mathsf{ct}) \to m$: On input the aggregation key ak, a collection of decryption key shares σ_ℓ for a set of users $\ell \in U \subseteq [L]$ where $|U| = T$, a set of identities $S \subseteq \mathcal{I}$, an identity $\mathsf{id} \in S$, a batch label $\mathsf{tg} \in \mathcal{T}$, and a ciphertext ct, the decryption algorithm outputs a message $m \in \mathcal{M}$. This algorithm is deterministic.

We require $\Pi_{\mathsf{BatchSTIBE}}$ satisfy the following properties:

- **Completeness:** For all $\lambda, B \in \mathbb{N}$, all $N \leq 2^\lambda$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda, 1^B, 1^N)$, all batch labels $\mathsf{tg} \in \mathcal{T}$, all sets $S \subseteq \mathcal{I}$ of size at most B (where $\mathcal{M}, \mathcal{I}, \mathcal{T}$ are the message, identity, and batch label spaces associated with pp, respectively), we have

$$\Pr\left[\mathsf{VerifyKeyShare}(\mathsf{pk}, \mathsf{dig}, \mathsf{tg}, \sigma) = 1 : \begin{array}{c}(\mathsf{pk}, \mathsf{sk}, \mathsf{ht}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}) \\ \mathsf{dig} = \mathsf{Digest}(\mathsf{pp}, S) \\ \sigma \leftarrow \mathsf{CompKeyShare}(\mathsf{sk}, \mathsf{dig}, \mathsf{tg})\end{array}\right] = 1.$$

- **Correctness:** For all $\lambda, B \in \mathbb{N}$, all $L \leq N \leq 2^\lambda$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda, 1^B, 1^N)$, all messages $m \in \mathcal{M}$, all identities $\mathsf{id} \in \mathcal{I}$, all batch labels $\mathsf{tg} \in \mathcal{T}$ (where $\mathcal{M}, \mathcal{I}, \mathcal{T}$ are the message, identity, and batch label spaces associated with pp, respectively), all collections of $(\mathsf{pk}_1, \mathsf{ht}_1), \ldots, (\mathsf{pk}_L, \mathsf{ht}_L)$ where $(\mathsf{pk}_i, \mathsf{ht}_i)$ is in the support of $\mathsf{KeyGen}(\mathsf{pp})$ for all $i \in [L]$, and setting $(\mathsf{ek}, \mathsf{ak}) = \mathsf{Preprocess}(\mathsf{pp}, (\mathsf{ht}_1, \ldots, \mathsf{ht}_L))$, all thresholds $T \leq L$, all ciphertexts ct in the support of $\mathsf{Encrypt}(\mathsf{ek}, m, \mathsf{id}, \mathsf{tg}, T)$, all sets $S \subseteq \mathcal{I}$ of size at most B where $\mathsf{id} \in S$, and setting $\mathsf{dig} = \mathsf{Digest}(\mathsf{pp}, S)$, all sets $U \subseteq [L]$ of size $|U| = T$ and all partial decryptions $\{\sigma_\ell\}_{\ell \in U}$ where $\mathsf{VerifyKeyShare}(\mathsf{pk}_\ell, \mathsf{dig}, \mathsf{tg}, \sigma_\ell) = 1$, we have

$$\mathsf{Decrypt}(\mathsf{ak}, \{\sigma_\ell\}_{\ell \in U}, S, (\mathsf{id}, \mathsf{tg}), \mathsf{ct}) = 1.$$

- **Static security:** For a security parameter λ, a batch size B, a bound on the size of the decryption committee N, a bit $\beta \in \{0, 1\}$, and an adversary $\mathcal{A}$, we define the threshold batched IBE with silent setup security game as follows:
  - Algorithm $\mathcal{A}$ commits to the challenge identity $\mathsf{id}^* \in \mathcal{I}$, the challenge batch label $\mathsf{tg}^* \in \mathcal{T}$, the size of the committee $L \leq N$, the threshold $T \leq L$, and the indices of the corrupted users $C \subseteq [L]$ where $|C| < T$.
  - The challenger starts by computing $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^B, 1^N)$ and $(\mathsf{pk}_\ell, \mathsf{sk}_\ell, \mathsf{ht}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for each $\ell \in [L] \setminus C$. It gives $(\mathsf{pp}, \{\mathsf{pk}_\ell, \mathsf{ht}_\ell\}_{\ell \in [L] \setminus C})$ to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ can now make any number of key-share-computation queries. On each query, algorithm $\mathcal{A}$ specifies an index $\ell \in [L] \setminus C$, a set of identities $S \subseteq \mathcal{I}$ where $|S| \leq B$, and a batch label $\mathsf{tg} \in \mathcal{T}$. The challenger replies with the decryption key share $\sigma_\ell \leftarrow \mathsf{ComputeKey}(\mathsf{sk}_\ell, \mathsf{Digest}(\mathsf{pp}, S), \mathsf{tg})$.
  - After $\mathcal{A}$ finishes making key-share computation queries, it specifies the key-generation randomness $\rho_\ell \in \{0, 1\}^*$ for each of the corrupted users $\ell \in C$. In addition, it outputs two messages $m_0, m_1 \in \mathcal{M}$.
  - For each $\ell \in C$, the challenger computes $(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{crs}; \rho_\ell)$. Next, it computes $(\mathsf{ek}, \mathsf{ak}) \leftarrow \mathsf{Preprocess}(\mathsf{crs}, \{(\mathsf{pk}_\ell, \mathsf{ht}_\ell)\}_{\ell \in [L]})$. Finally, the challenger replies to $\mathcal{A}$ with the challenge ciphertext $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{ek}, m_\beta, \mathsf{id}^*, \mathsf{tg}^*, T)$. Note that the challenger does not need to provide $(\mathsf{ek}, \mathsf{ak})$ to $\mathcal{A}$ because the Preprocess algorithm is deterministic so algorithm $\mathcal{A}$ can compute $(\mathsf{ek}, \mathsf{ak})$ itself.
  - Algorithm $\mathcal{A}$ can continue to make key-share-computation queries (subject to the same restrictions as described above). The challenger responds in the same manner.
  - At the end of the game, algorithm $\mathcal{A}$ outputs a bit $\beta' \in \{0, 1\}$, which is the output of the experiment.

  We say an adversary $\mathcal{A}$ is admissible if the following two conditions hold:
  - For each index $\ell \in [L] \setminus C$, algorithm $\mathcal{A}$ makes at most one key-share-computation query on the challenge batch label tg^*.
  - Algorithm $\mathcal{A}$ does not make a key-share-computation query on a pair (S, tg) where $\mathsf{tg} = \mathsf{tg}^*$ and $\mathsf{id}^* \in S$.[7]

  We say $\Pi_{\mathsf{BatchSTIBE}}$ is statically secure in the registered key model if for all polynomials $B = B(\lambda), N = N(\lambda)$ and all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\beta' = 1 : \beta = 0] - \Pr[\beta' = 1 : \beta = 1]| = \mathsf{negl}(\lambda) \tag{5.1}$$

  in the above security game. We say $\Pi_{\mathsf{BatchSTIBE}}$ is statically secure in the registered key model for a specific batch size $B = B(\lambda)$ and committee size $N = N(\lambda)$ if Eq. (5.1) holds for the specific functions B and N.

[7] An adversary that would like to make such a query to a user $\ell \in [L] \setminus C$ should instead corrupt user ℓ instead.

- **Succinctness:** There exists a universal polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda, B \in \mathbb{N}$, all $N \leq 2^\lambda$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda)$, all $(\mathsf{pk}_\ell, \mathsf{sk}_\ell, \mathsf{ht}_\ell)$ in the support of $\mathsf{KeyGen}(\mathsf{pp})$, all digests dig in the support of $\mathsf{Digest}(\mathsf{pp}, \cdot)$, and all batch labels $\mathsf{tg} \in \mathcal{T}$ (where $\mathcal{T}$ is the batch label space associated with pp), the following properties must hold:
  - The encryption key ek output by $\mathsf{Preprocess}(\mathsf{pp}, (\mathsf{ht}_1, \ldots, \mathsf{ht}_\ell))$ has size $\mathsf{poly}(\lambda)$.
  - The size of the digest dig is $\mathsf{poly}(\lambda)$ and the running time of the key-share-computation algorithm $\mathsf{CompKeyShare}(\mathsf{sk}, \mathsf{dig}, \cdot)$ is $\mathsf{poly}(\lambda)$.

Fixed-threshold batched IBE. To simplify our exposition, we define a relaxed version of threshold batched IBE with silent setup where the size of the decryption committee as well as the size of the threshold are fixed at Setup (rather than determined dynamically at aggregation and encryption time). While this may seem like a significant relaxation of the functionality, we can apply the simple padding and powers-of-two trick from [WW25a] to obtain a scheme that supports dynamic thresholds (see Remark 5.10). The [WW25a] transformation increases the size of the public parameters by a $\log N$ factor and does not affect the ciphertext size or the secret key size. Thus, for the main construction, we just focus on the comparably simpler fixed threshold setting. We define this more precisely below:

Definition 5.2 (Fixed-Threshold Batched IBE with Silent Setup)**.** A *fixed-threshold* batched IBE with silent setup is defined as in Definition 5.1 except with two simplifying assumptions:

- First, the threshold T is fixed during Setup rather than provided as a parameter to Encrypt.
- The size of the decryption committee is always set to $L = N$. In this case, we simply provide 1^L to Setup as the *exact* size of the decryption committee.

Remark 5.3 (Querying Decrypters on Different Sets for a Batch Label)**.** Our security definition for threshold batched IBE with silent setup (Definition 5.1) allows the adversary to request a decryption share for a *different* set S_ℓ from each decryption authority $\ell \in [L] \setminus C$ for the same batch label tg. The only restriction is that each decryption authority issue one key share for each batch label. In contrast, the security definition of threshold batched IBE from [AFP25] (see also Definition E.1) only ensures security against adversaries that specify the *same* set S to all of the decryption authorities when requesting a decryption key share for a particular batch label tg.

There is an important distinction between these two definitions. Under our definition, each decryption authority only needs to keep track of whether they individually have issued a decryption key share for each batch label. For instance, if the batch labels were a counter (or block number), the authority only needs to remember the current count. For threshold cryptography, it is natural to assume that decryption authorities do not have to coordinate or even be aware of each other.

On the other hand, with the [AFP25] definition, all of the decryption authorities in the system must coordinate and affirm that they are generating decryption key shares for the same set for each batch label tg. Otherwise, if one authority releases a key for a set S with respect to tg while another release a key for set $S' \neq S$ on the same batch label, then all bets are off. In fact, the [AFP25] scheme no longer provides semantic security if the adversary could request decryption key shares for two different sets S, S' under the same batch label tg to two different decryption authorities. The issue stems from the fact that all of the decryption authorities hold shares of a single master secret key for a (centralized) batched IBE that only ensures security if exactly one key is given out for each batch label.

In this section, we focus on the stronger notion of security where the only restriction we impose on the decryption authorities is they give out at most one decryption key share for each batch label (just as in the centralized scheme). As we show in Construction 5.5, we achieve this stronger security notion by having each decryption authority sample an independent share, and the share aggregation is done using the approach from [WW25a]. In fact, even without considering the silent setup property, the [WW25a] approach already provides a way to build a threshold batched IBE satisfying the stronger notion of security.

5.1 Constructing Fixed-Threshold Batched IBE with Silent Setup

In this section, we show how to construct a *fixed-threshold* batched IBE with silent setup from pairings. As mentioned before, this construction integrates our batched IBE scheme from Construction 4.2 with the threshold IBE scheme

with silent setup from [WW25a]. As in Section 4, we first introduce the q-type assumption we use in the security analysis and then give our construction. In Appendix B, we show this assumption holds in the standard generic bilinear group model [Sho97, BBG05].

Assumption 5.4 ((B, L)-Bilinear Diffie-Hellman Exponent Variant)**.** Let GroupGen be a prime-order bilinear group generator. For a security parameter λ and parameters $B, L \in \mathbb{N}$, and a bit $\beta \in \{0, 1\}$, we define the distribution $\mathcal{D}_{\lambda,B,L,\beta}$ as follows:

- Sample $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$. Sample exponents $a, b, c, s, \tau \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. Define

$$\mathsf{params} = \left(\begin{array}{c} 1^\lambda, \mathcal{G}, [a]_2, [b]_2, [bc^{L+1}]_1, [bc^{L+1}]_2, [ab]_1, [ab]_2, [s]_1, \\ [\tau]_1, \{[\tau^j]_2\}_{j\in[0,B]}, \{[c^\ell\tau^j]_1, [c^\ell\tau^j]_2\}_{\ell\in[2L],j\in[0,B]}, \\ \{[abc^\ell\tau^j]_2\}_{\ell\in[2L]\setminus\{L+1\},j\in[0,B]}, \{[abc^{L+1}\tau^j]_2\}_{j\in[B]}, \\ \{[c^\ell s]_2, [c^\ell s\tau]_2, [abc^\ell s\tau]_2\}_{\ell\in[L]} \end{array} \right),$$

- If $\beta = 0$, let $\xi = abc^{L+1}s$ and if $\beta = 1$, sample $\xi \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. Output $(\mathsf{params}, [\xi]_\mathsf{T})$.

We say Assumption 5.4 holds with respect to GroupGen and parameters $B = B(\lambda)$, $L = L(\lambda)$ if the distributions $\mathcal{D}_0 = \{\mathcal{D}_{\lambda,B(\lambda),L(\lambda),0}\}_{\lambda\in\mathbb{N}}$ and $\mathcal{D}_1 = \{\mathcal{D}_{\lambda,B(\lambda),L(\lambda),1}\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable.

Construction 5.5 (Fixed-Threshold Batched IBE with Silent Setup)**.** Let GroupGen be a prime-order bilinear group generator. We construct a fixed-threshold batched IBE scheme with silent setup $\Pi_{\mathsf{BatchSTIBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Preprocess}, \mathsf{Encrypt}, \mathsf{Digest}, \mathsf{CompKeyShare}, \mathsf{VerifyKeyShare}, \mathsf{Decrypt})$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^B, 1^L, T)$: On input the security parameter λ, a bound on the batch size B, the size of the decryption committee L, and the threshold $T \le L$, the setup algorithm proceeds as follows:
  - Sample $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and exponents $c, \tau, v, h, t \xleftarrow{\mathrm{R}} \mathbb{Z}_p$.
  - Let $\mathbf{M} \in \mathbb{Z}_p^{L\times T}$ be the share-generation matrix for a T-out-of-L threshold policy over $\mathbb{Z}_p$ (e.g., the Vandermonde matrix over $[L]$; see Section 3). Sample $\mathbf{t} \xleftarrow{\mathrm{R}} \mathbb{Z}_p^T$ where $t_1 = t$. For $\ell \in [L]$, let $\mathbf{m}_\ell^\mathsf{T}$ be the ℓ^{th} row of $\mathbf{M}$. Then compute

$$[z_0]_2 = \sum_{\ell\in[L]} [c^\ell \mathbf{m}_\ell^\mathsf{T}\mathbf{t}]_2$$
$$\forall \ell \in [L] : [v_{\ell,0}]_2 = \sum_{\substack{i\in[L]\\ i\neq\ell}} [c^{L+1-\ell+i}\mathbf{m}_i^\mathsf{T}\mathbf{t}]_2$$

  Then output the the public parameters

$$\mathsf{pp} = \left(\begin{array}{c} \mathcal{G}, [\tau]_1, \{[\tau^j]_2\}_{j\in[B]}, [v]_1, [v]_2, [h]_1, [h]_2, [c^{L+1}t]_\mathsf{T}, \\ [z_0]_2, \{[v_{\ell,0}]_2\}_{\ell\in[L]}, \{[c^\ell\tau^j]_1, [c^\ell\tau^j]_2\}_{\ell\in[2L],j\in[0,B]} \end{array} \right) \tag{5.2}$$

  The message space associated with pp is $\mathbb{G}_\mathsf{T}$, the identity space is $\mathbb{Z}_p$, and the tag space is $\mathbb{Z}_p$.

- $\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameters pp (parsed according to Eq. (5.2)), the key-generation algorithm samples exponents $\alpha, w' \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and sets $[c^{L+1}\alpha]_\mathsf{T} = [\alpha]_1 \cdot [c^{L+1}]_2$. Next, it computes

$$\mathsf{pk} = (\mathcal{G}, [v]_1, [h]_1, [w']_1, [c^{L+1}\alpha]_\mathsf{T})$$
$$\mathsf{ht} = \{\alpha \cdot [c^i]_2, w' \cdot [c^i\tau^j]_2\}_{i\in[2L]\setminus\{L+1\},j\in[0,B]}.$$

  Finally, it outputs the public key pk, the aggregation hint ht, and the secret key $\mathsf{sk} = (\mathcal{G}, \alpha, w', \alpha\cdot[c^{L+1}]_2, [v]_2, [h]_2)$.

- Preprocess(pp, $(\mathsf{ht}_1, \ldots, \mathsf{ht}_L)$): On input the public parameters pp (parsed according to Eq. (5.2)) and L aggregation hints $\mathsf{ht}_\ell = \left\{[c^i \alpha_\ell]_2, [c^i w'_\ell \tau^j]_2\right\}_{i \in [2L] \setminus \{L+1\}, j \in [0,B]}$, the preprocessing algorithm computes the aggregated public key components

$$[z]_2 = [z_0]_2 + \sum_{\ell \in [L]} [c^\ell \alpha_\ell]_2 \quad \text{and} \quad [w]_2 = \sum_{\ell \in [L]} [c^\ell w'_\ell]_2 \quad \text{and} \quad [w\tau]_2 = \sum_{\ell \in [L]} [c^\ell w'_\ell \tau]_2.$$

  Then, for each $\ell \in [L]$ and $j \in [0, B]$, it computes the aggregated cross terms

$$[v_\ell]_2 = [v_{\ell,0}]_2 + \sum_{\substack{i \in [L] \\ i \neq \ell}} [c^{L+1-\ell+i} \alpha_i]_2 \quad \text{and} \quad [d_\ell \tau^j]_2 = \sum_{\substack{i \in [L] \\ i \neq \ell}} [c^{L+1-\ell+i} w'_i \tau^j]_2.$$

  Then it sets

$$\begin{aligned} \mathsf{ek} &= (\mathcal{G}, [v]_1, [h]_1, [w]_2, [w\tau]_2, [z]_2, [c^{L+1} t]_\mathsf{T}) \\ \mathsf{ak} &= (\mathcal{G}, \{[c^\ell]_1, [v_\ell]_2, [c^\ell \tau^j]_1, [d_\ell \tau^j]_2\}_{\ell \in [L], j \in [0,B]}). \end{aligned} \tag{5.3}$$

  The preprocessing algorithm outputs the encryption key ek and the aggregation key ak.

- Encrypt(ek, $[m]_\mathsf{T}$, id, tg): On input the encryption key $\mathsf{ek} = (\mathcal{G}, [v]_1, [h]_1, [w]_2, [w\tau]_2, [z]_2, [c^{L+1} t]_\mathsf{T})$, a message $[m]_\mathsf{T} \in \mathbb{G}_\mathsf{T}$, an identity $\mathsf{id} \in \mathbb{Z}_p$, and a batch label $\mathsf{tg} \in \mathbb{Z}_p$, the encryption algorithm samples a random exponent $s \xleftarrow{\text{R}} \mathbb{Z}_p$. It then outputs the ciphertext

$$\mathsf{ct} = \Big([s]_1,\ s[w\tau]_2 - (s \cdot \mathsf{id})[w]_2,\ s([v]_1 + \mathsf{tg} \cdot [h]_1),\ s[z]_2,\ s[c^{L+1} t]_\mathsf{T} + [m]_\mathsf{T}\Big).$$

- Digest(pp, S): On input the public parameters pp (parsed according to Eq. (5.2)) and a set of identities $S \subseteq \mathbb{Z}_p$ where $|S| \leq B$, the digest algorithm defines the polynomial $F_S(x) = \prod_{\mathsf{id} \in S} (x - \mathsf{id})$ whose roots are the identities $\mathsf{id} \in S$. Write $F(x) = \sum_{j \in [0, |S|]} f_j x^j$. Output the digest

$$\mathsf{dig} = \sum_{j \in [0,|S|]} f_j \cdot [c^{L+1} \tau^j]_2 = [c^{L+1} \cdot F_S(\tau)]_2.$$

- CompKeyShare(sk, dig, tg): On input a secret key $\mathsf{sk} = (\mathcal{G}, \alpha, w', [c^{L+1}\alpha]_2, [v]_2, [h]_2)$, a digest $\mathsf{dig} = [d]_2$, and a batch label $\mathsf{tg} \in \mathbb{Z}_p$, the key-share-computation algorithm samples a random $r \xleftarrow{\text{R}} \mathbb{Z}_p$ and $y \xleftarrow{\text{R}} \mathbb{Z}_p^*$ and outputs the decryption key share

$$\sigma = \big(y,\ [r]_2,\ [c^{L+1}\alpha]_2 + r([v]_2 + \mathsf{tg} \cdot [h]_2) + y w' \cdot [d]_2\big).$$

- VerifyKeyShare(pk, dig, tg, σ): On input a public key $\mathsf{pk} = (\mathcal{G}, [v]_1, [h]_1, [w']_1, [c^{L+1}\alpha]_\mathsf{T})$, a digest $\mathsf{dig} = [d]_2$, a batch label $\mathsf{tg} \in \mathbb{Z}_p$, and a decryption key share $\sigma = \big(y, [\sigma_1]_2, [\sigma_2]_2\big)$, the key-share verification algorithm outputs 1 if the following relation holds (and 0 otherwise):

$$[c^{L+1}\alpha]_\mathsf{T} = [1]_1 \cdot [\sigma_2]_2 - ([v]_1 + \mathsf{tg} \cdot [h]_1) \cdot [\sigma_1]_2 - y \cdot [w']_1 \cdot [d]_2.$$

- Decrypt(ak, $\{\sigma_\ell\}_{\ell \in U}$, S, (id, tg), ct): On input the aggregation key ak (parsed according to Eq. (5.3)), a collection of decryption key shares $\sigma_\ell = (y_\ell, [\sigma_{\ell,1}]_2, [\sigma_{\ell,2}]_2)$ for a set of users $\ell \in U \subseteq [L]$ where $|U| = T$, a set of identities $S \subseteq \mathbb{Z}_p$, an identity $\mathsf{id} \in S$, a batch label $\mathsf{tg} \in \mathbb{Z}_p$, and a ciphertext $\mathsf{ct} = ([\mathsf{ct}_1]_1, [\mathsf{ct}_2]_2, [\mathsf{ct}_3]_1, [\mathsf{ct}_4]_2, [\mathsf{ct}_5]_\mathsf{T})$, the decryption algorithm proceeds as follows:

  - Let $\mathbf{M} \in \mathbb{Z}_p^{T \times L}$ be the share-generating matrix for the T-out-of-L threshold policy. Let $\boldsymbol{\omega} \in \mathbb{Z}_p^L$ be the interpolation vector where $\boldsymbol{\omega}^\mathsf{T} \mathbf{M} = \mathbf{e}_1^\mathsf{T}$ and $\omega_\ell = 0$ for all $\ell \notin U$.

– Let $K = |S|$. The decryption algorithm defines polynomials $F_S(x) = \prod_{\mathsf{id}' \in S}(x - \mathsf{id}') = \sum_{j\in[0,K]} f_j x^j$ and $F_{S\setminus\{\mathsf{id}\}}(x) = \prod_{\mathsf{id}' \in S\setminus\{\mathsf{id}\}}(x - \mathsf{id}') = \sum_{j\in[0,K-1]} f'_j x^j$. Then, for each $\ell \in U$, it computes the following:

$$[c^{L+1-\ell} \cdot F_{S\setminus\{\mathsf{id}\}}(\tau)]_1 := \sum_{j\in[0,K-1]} f'_j [c^{L+1-\ell}\tau^j]_1$$
$$[d_\ell \cdot F_S(\tau)]_2 := \sum_{j\in[0,K]} f_j [d_\ell \tau^j]_2.$$

For each $\ell \in U$, it computes

$$[\delta_\ell]_\mathsf{T} = ([\mathsf{ct}_1]_1 \cdot [\sigma_{\ell,2}]_2) - y_\ell \cdot \left(([c^{L+1-\ell} \cdot F_{S\setminus\{\mathsf{id}^*\}}(\tau)]_1 \cdot [\mathsf{ct}_2]_2) - ([\mathsf{ct}_1]_1 \cdot [d_\ell \cdot F_S(\tau)]_2)\right) - ([\mathsf{ct}_3]_1 \cdot [\sigma_{\ell,1}]_2).$$

– Finally, the decryption algorithm outputs

$$[\mathsf{ct}_5]_\mathsf{T} - \sum_{\ell\in U} (\omega_\ell \cdot [c^{L+1-\ell}]_1 \cdot [\mathsf{ct}_4]_2) + \sum_{\ell\in U} \omega_\ell \cdot [\delta_\ell]_\mathsf{T} + [\mathsf{ct}_1]_1 \cdot \sum_{\ell\in U} \omega_\ell \cdot [v_\ell]_2. \tag{5.4}$$

Theorem 5.6 (Completeness). *Construction 5.5 is complete.*

Proof. Take any $\lambda, B \in \mathbb{N}$ and $T \le L \le 2^\lambda$ and take $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^B, 1^L, 1^T)$, where

$$\mathsf{pp} = \left(\begin{array}{c} \mathcal{G}, [\tau]_1, \{[\tau^j]_2\}_{j\in[B]}, [v]_1, [v]_2, [h]_1, [h]_2, [c^{L+1}t]_\mathsf{T}, \\ [z_0]_2, \{[v_{\ell,0}]_2\}_{\ell\in[L]}, \{[c^\ell\tau^j]_1, [c^\ell\tau^j]_2\}_{\ell\in[2L],j\in[0,B]} \end{array} \right)$$

Take any batch label $\mathsf{tg} \in \mathbb{Z}_p$ and set $S \subseteq \mathbb{Z}_p$ of size at most B. Let

$$\begin{aligned} (\mathsf{pk}, \mathsf{sk}, \mathsf{ht}) &\leftarrow \mathsf{KeyGen}(\mathsf{pp}) \\ \mathsf{dig} &= \mathsf{Digest}(\mathsf{pp}, S) \\ \sigma &\leftarrow \mathsf{CompKeyShare}(\mathsf{sk}, \mathsf{dig}, \mathsf{tg}). \end{aligned}$$

Let $F_S(x) = \prod_{\mathsf{id}\in S}(x - \mathsf{id})$. Then we can write

$$\begin{aligned} \mathsf{pk} &= (\mathcal{G}, [v]_1, [h]_1, [w']_1, [c^{L+1}\alpha]_\mathsf{T}) \\ \mathsf{sk} &= (\mathcal{G}, \alpha, w', \alpha \cdot [c^{L+1}]_2, [v]_2, [h]_2) \\ \mathsf{dig} &= [c^{L+1} \cdot F_S(\tau)]_2 \\ \sigma &= (y_\ell, [r]_2, [c^{L+1}\alpha + r(v + h \cdot \mathsf{tg}^*) + y c^{L+1} w' \cdot F_S(\tau)]_2). \end{aligned}$$

Write $\mathsf{dig} = [d]_2$ and $\sigma = (y, [\sigma_1]_2, [\sigma_2]_2)$. Completeness holds if the following holds

$$[c^{L+1}\alpha]_\mathsf{T} = [1]_1 \cdot [\sigma_2]_2 - ([v]_1 + \mathsf{tg}^* \cdot [h]_1) \cdot [\sigma_1]_2 - y \cdot [w']_1 \cdot [d]_2$$

It is sufficient to check the exponents of both sides:

$$c^{L+1}\alpha = \sigma_2 - (v + \mathsf{tg}^* \cdot h) \cdot \sigma_1 - y \cdot w' \cdot d \tag{5.5}$$

By construction, the three terms on the right-hand side are as follows:

$$\begin{aligned} \sigma_2 &= c^{L+1}\alpha + r(v + h \cdot \mathsf{tg}^*) + y c^{L+1} w' \cdot F_S(\tau) \\ (v + \mathsf{tg}^* \cdot h) \cdot \sigma_1 &= (v + \mathsf{tg}^* \cdot h) \cdot r \\ y \cdot w' \cdot d &= y \cdot w' \cdot c^{L+1} \cdot F_S(\tau) \end{aligned}$$

where terms in the same color can be canceled out when we substitute them back to Eq. (5.5). This proves that Eq. (5.5) holds by construction and completeness follows. □

Theorem 5.7 (Correctness). *Construction 5.5 is correct.*

Proof. Take any $\lambda, B \in \mathbb{N}$ where $T \le L \le 2^\lambda$ and take $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^B, 1^L, 1^T)$, where

$$\mathsf{pp} = \left(\begin{array}{c} \mathcal{G}\,,\ [\tau]_1\,,\ \{[\tau^j]_2\}_{j\in[B]}\,,\ [v]_1\,,\ [v]_2\,,\ [h]_1\,,\ [h]_2\,,\ [c^{L+1}t]_\mathsf{T}\,, \\ [z_0]_2\,,\ \{[v_{\ell,0}]_2\}_{\ell\in[L]}\,,\ \{[c^\ell\tau^j]_1\,,\ [c^\ell\tau^j]_2\}_{\ell\in[2L],j\in[0,B]} \end{array} \right)$$

Take any message $[m]_\mathsf{T} \in \mathbb{G}_\mathsf{T}$, identity $\mathsf{id}^* \in \mathbb{Z}_p$, batch label $\mathsf{tg}^* \in \mathbb{Z}_p$, set $S^* \subseteq \mathbb{Z}_p$ of size at most B where $\mathsf{id}^* \in S^*$. Take any $T \le L$ and any collection of $(\mathsf{pk}_1, \mathsf{ht}_1), \ldots, (\mathsf{pk}_L, \mathsf{ht}_L)$ where $(\mathsf{pk}_i, \mathsf{ht}_i)$ is in the support of $\mathsf{KeyGen}(\mathsf{pp})$ for all $i \in [L]$. Let $\alpha_i, w'_i \in \mathbb{Z}_p$ be the secret exponents associated with pk_i. Let

$$\begin{aligned} (\mathsf{ek}, \mathsf{ak}) &= \mathsf{Preprocess}(\mathsf{pp}, (\mathsf{ht}_1, \ldots, \mathsf{ht}_L)) \\ \mathsf{ct} &\leftarrow \mathsf{Encrypt}(\mathsf{ek}, [m]_\mathsf{T}, \mathsf{id}^*, \mathsf{tg}^*) \\ \mathsf{dig} &= \mathsf{Digest}(\mathsf{pp}, S^*) \end{aligned}$$

Take any set $U \subseteq [L]$ of size $|U| = T$ and a set of decryption shares $\{\sigma_\ell\}_{\ell\in U}$ where $\mathsf{VerifyKeyShare}(\mathsf{pk}_\ell, \mathsf{dig}, \mathsf{tg}, \sigma_\ell) = 1$. For a set $S \subseteq \mathbb{Z}_p$, let $F_S(x) = \prod_{\mathsf{id}\in S}(x - \mathsf{id})$. Then we can write

$$\begin{aligned} \mathsf{ek} &= (\mathcal{G}, [v]_1, [h]_1, [w]_2, [w\tau]_2, [z]_2, [c^{L+1}t]_\mathsf{T}) \\ \mathsf{ak} &= (\mathcal{G}, \{[c^\ell]_1, [v_\ell]_2, [c^\ell\tau^j]_1, [d_\ell\tau^j]_2\}_{\ell\in[L],j\in[0,B]}) \\ \mathsf{ct} &= ([s]_1, [sw(\tau - \mathsf{id}^*)]_2, [s(v + h\cdot\mathsf{tg}^*)]_1, [sz]_2, [c^{L+1}st]_\mathsf{T} + [m]_\mathsf{T}) \\ \mathsf{dig} &= [c^{L+1}\cdot F_{S^*}(\tau)]_2. \end{aligned}$$

Moreover, since the decryption shares $\sigma_\ell = (y_\ell, [\sigma_{\ell,1}]_2, [\sigma_{\ell,2}]_2)$ are valid, we have

$$[c^{L+1}\alpha_\ell]_\mathsf{T} = [1]_1 \cdot [\sigma_{\ell,2}]_2 - ([v]_1 + \mathsf{tg}^*\cdot[h]_1)\cdot[\sigma_{\ell,1}]_2 - y_\ell\cdot[w'_\ell]_1\cdot[c^{L+1}\cdot F_{S^*}(\tau)]_2. \tag{5.6}$$

By construction of Preprocess, we have that

$$w = \sum_{i\in[L]} c^i w'_i \quad \text{and} \quad d_\ell\tau^j = \sum_{\substack{i\in[L]\\ i\ne\ell}} c^{L+1-\ell+i} w'_i \tau^j.$$

Consider the value of $\mathsf{Decrypt}(\mathsf{ak}, \{\sigma_\ell\}_{\ell\in U}, S^*, (\mathsf{id}^*, \mathsf{tg}^*), \mathsf{ct})$. Let $\mathbf{M}$ be the share-generating matrix for the T-out-of-L threshold policy and let $\boldsymbol{\omega} \in \mathbb{Z}_p^L$ be the interpolation vector where $\boldsymbol{\omega}^\mathsf{T}\mathbf{M} = \mathbf{e}_1^\mathsf{T}$ and $\omega_\ell = 0$ for all $\ell \notin U$. Let $K = |S^*|$ and let $F_{S^*}(x) = \sum_{j\in[0,K]} f_j x^j$ and $F_{S^*\setminus\{\mathsf{id}^*\}}(x) = \sum_{j\in[0,K-1]} f'_j x^j$. For each $\ell \in U$, we have

$$\begin{aligned} [c^{L+1-\ell}\cdot F_{S^*\setminus\{\mathsf{id}^*\}}(\tau)]_1 &= \sum_{j\in[0,K-1]} f'_j [c^{L+1-\ell}\tau^j]_1 \\ [d_\ell\cdot F_{S^*}(\tau)]_2 &= \sum_{j\in[0,K]} f_j [d_\ell\tau^j]_2. \end{aligned}$$

This means

$$\begin{aligned} [c^{L+1-\ell}\cdot F_{S^*\setminus\{\mathsf{id}^*\}}(\tau)]_1 \cdot [sw(\tau-\mathsf{id}^*)]_2 &= \sum_{i\in[L]} [sc^{L+1-\ell+i} w'_i F_{S^*}(\tau)]_\mathsf{T} \\ &= [sc^{L+1}w'_\ell F_{S^*}(\tau)]_\mathsf{T} + [s]_1 \cdot \sum_{\substack{i\in[L]\\ i\ne\ell}} [c^{L+1-\ell+i} w'_i F_{S^*}(\tau)]_2 \\ &= [sc^{L+1}w'_\ell F_{S^*}(\tau)]_\mathsf{T} + [s]_1\cdot[d_\ell\cdot F_{S^*}(\tau)]_2. \end{aligned}$$

This means that

$$\begin{aligned} &\left([c^{L+1-\ell}\cdot F_{S^*\setminus\{\mathsf{id}^*\}}(\tau)]_1 \cdot [\mathsf{ct}_2]_2\right) - ([\mathsf{ct}_1]_1\cdot[d_\ell\cdot F_{S^*}(\tau)]_2) \\ &\quad = [c^{L+1-\ell}\cdot F_{S^*\setminus\{\mathsf{id}^*\}}(\tau)]_1\cdot[sw(\tau-\mathsf{id}^*)]_2 - [s]_1\cdot[d_\ell\cdot F_{S^*}(\tau)]_2 \\ &\quad = [sc^{L+1}w'_\ell F_{S^*}(\tau)]_\mathsf{T}. \end{aligned} \tag{5.7}$$

Next, the decryption algorithm computes for each $\ell \in U$:

$$\begin{aligned}
[\delta_\ell]_T &= ([\mathrm{ct}_1]_1 \cdot [\sigma_{\ell,2}]_2) - y_\ell \cdot \left(([c^{L+1-\ell} \cdot F_{S^* \setminus \{\mathrm{id}^*\}}(\tau)]_1 \cdot [\mathrm{ct}_2]_2) - ([\mathrm{ct}_1]_1 \cdot [d_\ell \cdot F_{S^*}(\tau)]_2)\right) - ([\mathrm{ct}_3]_1 \cdot [\sigma_{\ell,1}]_2) \\
&= ([s]_1 \cdot [\sigma_{\ell,2}]_2) - y_\ell \cdot [sc^{L+1} w'_\ell F_{S^*}(\tau)]_T - ([s(v + h \cdot \mathrm{tg}^*)]_1 \cdot [\sigma_{\ell,1}]_2) \\
&= s \cdot \left([1]_1 \cdot [\sigma_{\ell,2}]_2 - ([v]_1 + \mathrm{tg}^* \cdot [h]_1) \cdot [\sigma_{\ell,1}]_2 - y_\ell \cdot [w'_\ell]_1 \cdot [c^{L+1} \cdot F_{S^*}(\tau)]_2\right) \\
&= s \cdot [c^{L+1} \alpha_\ell]_T = [sc^{L+1} \alpha_\ell]_T,
\end{aligned}$$

where the second step uses Eq. (5.7) and the fourth step uses Eq. (5.6). We now consider the components in Eq. (5.4). First, by construction of Preprocess, we have

$$\begin{aligned}
z &= z_0 + \sum_{\ell \in [L]} c^\ell \alpha_\ell = \sum_{\ell \in [L]} c^\ell (\mathbf{m}_\ell^\intercal \mathbf{t} + \alpha_\ell), \\
v_\ell &= v_{\ell,0} + \sum_{\substack{i \in [L] \\ i \neq \ell}} c^{L+1-\ell+i} \alpha_i = \sum_{\substack{i \in [L] \\ i \neq \ell}} c^{L+1-\ell+i} (\mathbf{m}_i^\intercal \mathbf{t} + \alpha_i).
\end{aligned}$$

This means

$$\begin{aligned}
\sum_{\ell \in U} \omega_\ell \cdot [c^{L+1-\ell}]_1 \cdot [\mathrm{ct}_4]_2 &= \sum_{\ell \in U} \omega_\ell \cdot [c^{L+1-\ell}]_1 \cdot [sz]_2 \\
&= \sum_{\ell \in U} \sum_{i \in [L]} [\omega_\ell s c^{L+1-\ell+i} (\mathbf{m}_i^\intercal \mathbf{t} + \alpha_i)]_T \\
&= \sum_{\ell \in U} [sc^{L+1} \omega_\ell \mathbf{m}_\ell^\intercal \mathbf{t}]_T + \sum_{\ell \in U} \omega_\ell [sc^{L+1} \alpha_\ell]_T + \sum_{\ell \in U} \omega_\ell \cdot [s]_1 \cdot [v_\ell]_2 \\
&= [stc^{L+1}]_T + \sum_{\ell \in U} [\omega_\ell (\delta_\ell + s v_\ell)]_T,
\end{aligned}$$

using the fact that $\sum_{\ell \in [L]} \omega_\ell \mathbf{m}_\ell^\intercal \mathbf{t} = t$. Eq. (5.4) now becomes

$$\begin{aligned}
&[\mathrm{ct}_5]_T - \sum_{\ell \in U} \left(\omega_\ell \cdot [c^{N+1-\ell}]_1 \cdot [\mathrm{ct}_4]_2\right) + \sum_{\ell \in U} \omega_\ell \cdot [\delta_\ell]_T + [\mathrm{ct}_1]_1 \cdot \sum_{\ell \in U} \omega_\ell \cdot [v_\ell]_2 \\
&\quad = [m]_T + [stc^{L+1}]_T - [stc^{L+1}]_T - \sum_{\ell \in U} [\omega_\ell (\delta_\ell + s v_\ell)]_T + \sum_{\ell \in U} [\omega_\ell (\delta_\ell + s v_\ell)]_T \\
&\quad = [m]_T,
\end{aligned}$$

and correctness holds. □

Theorem 5.8 (Static Security). *Take any polynomial $B = B(\lambda)$ and $L = L(\lambda)$. Suppose Assumption 5.4 holds with parameters B and L. Then, Construction 5.5 is statically secure in the registered key model with batch size B and supporting committees of size L.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the static security experiment for Construction 5.5 with batch size B and committees of size L. We define a simple sequence of hybrid experiments, each indexed by a bit $\beta \in \{0, 1\}$:

- $\mathrm{Hyb}_0^{(\beta)}$: This is the static security game with bit β.
- $\mathrm{Hyb}_1^{(\beta)}$: Same as $\mathrm{Hyb}_0^{(\beta)}$, except when constructing the challenge ciphertext

$$\mathrm{ct} = ([\mathrm{ct}_1]_1, [\mathrm{ct}_2]_2, [\mathrm{ct}_3]_1, [\mathrm{ct}_4]_2, [\mathrm{ct}_5]_T),$$

  the challenger samples $\mathrm{ct}_5 \xleftarrow{R} \mathbb{Z}_p$. In this experiment, the challenge ciphertext is independent of the bit β.

Let $\mathsf{Hyb}_1^{(0)}(\mathcal{A})$ and $\mathsf{Hyb}_1^{(1)}(\mathcal{A})$ denote the output distribution of an execution of experiment $\mathsf{Hyb}_1^{(0)}$ and $\mathsf{Hyb}_1^{(1)}$ with adversary $\mathcal{A}$, respectively. By construction, experiments $\mathsf{Hyb}_1^{(0)}$ and $\mathsf{Hyb}_1^{(1)}$ are identical so it suffices to show that the outputs of $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$ are computationally indistinguishable for each $\beta \in \{0,1\}$. Suppose there exists $\beta \in \{0,1\}$ and a non-negligible ε such that

$$\left|\Pr[\mathsf{Hyb}_0^{(\beta)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(\beta)}(\mathcal{A}) = 1]\right| \geq \varepsilon.$$

We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks Assumption 5.4 with parameters (B, L) and the same advantage ε:

1. At the beginning of the game, algorithm $\mathcal{B}$ receives a challenge $(\mathsf{params}, [\xi]_T)$ where

$$\mathsf{params} = \left(\begin{array}{c} 1^\lambda, \mathcal{G}, [a]_2, [b]_2, [bc^{L+1}]_1, [bc^{L+1}]_2, [ab]_1, [ab]_2, [s]_1, \\ [\hat{\tau}]_1, \{[\hat{\tau}^j]_2\}_{j\in[0,B]}, \{[c^\ell \hat{\tau}^j]_1, [c^\ell \hat{\tau}^j]_2\}_{\ell\in[2L],j\in[0,B]}, \\ \{[abc^\ell \hat{\tau}^j]_2\}_{\ell\in[2L]\setminus\{L+1\},j\in[0,B]}, \{[abc^{L+1}\hat{\tau}^j]_2\}_{j\in[B]}, \\ \{[c^\ell s]_2, [c^\ell s\hat{\tau}]_2, [abc^\ell s\hat{\tau}]_2\}_{\ell\in[L]} \end{array} \right),$$

   and $\xi = abc^{L+1}s$ or $\xi \xleftarrow{R} \mathbb{Z}_p$.

2. Algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ commits to the challenge identity $\mathsf{id}^* \in \mathbb{Z}_p$, the challenge batch label $\mathsf{tg}^* \in \mathbb{Z}_p$, and the indices of the corrupted users $C \subseteq [L]$.

3. Let $\mathbf{M}$ be the share-generating matrix for a T-out-of-L threshold policy. Since $|C| < T$, the set C does not satisfy the threshold policy. Thus, there exists a vector $\tilde{\mathbf{w}} \in \mathbb{Z}_p^L$ such that for all indices $i \in C$, $\mathbf{m}_i^\top \tilde{\mathbf{w}} = 0$, where $\mathbf{m}_i^\top$ is the i^{th} row of $\mathbf{M}$ and moreover, $\tilde{w}_1 = 1$.

4. Algorithm $\mathcal{B}$ now constructs the public parameters pp as follows. As in the proof of Theorem 4.4, we will use a "tilde" (e.g., $\tilde{\alpha}, \tilde{v}$) to denote an exponent that is chosen by (or otherwise known to) the reduction algorithm.

   - Algorithm $\mathcal{B}$ implicit sets $\tau = \hat{\tau} + \mathsf{id}^*$. Specifically, for each $j \in [0, B]$ and $\ell \in [2L]$, algorithm $\mathcal{B}$ computes

$$\begin{aligned} [\tau^j]_2 &= \sum_{k\in[0,j]} \binom{j}{k} \cdot (\mathsf{id}^*)^{j-k} \cdot [\hat{\tau}^k]_2 = [(\hat{\tau} + \mathsf{id}^*)^j]_2 \\ [c^\ell \tau^j]_1 &= \sum_{k\in[0,j]} \binom{j}{k} \cdot (\mathsf{id}^*)^{j-k} \cdot [c^\ell \hat{\tau}^k]_1 = [c^\ell(\hat{\tau} + \mathsf{id}^*)^j]_1 \\ [c^\ell \tau^j]_2 &= \sum_{k\in[0,j]} \binom{j}{k} \cdot (\mathsf{id}^*)^{j-k} \cdot [c^\ell \hat{\tau}^k]_2 = [c^\ell(\hat{\tau} + \mathsf{id}^*)^j]_2 \end{aligned} \tag{5.8}$$

     Algorithm $\mathcal{B}$ sets $[\tau]_1 = [\hat{\tau}]_1 + [\mathsf{id}^*]_1$.

   - Algorithm $\mathcal{B}$ samples a vector $\tilde{\mathbf{t}} \xleftarrow{R} \mathbb{Z}_p^N$ and implicitly sets $\mathbf{t} = \tilde{\mathbf{t}} + ab \cdot \tilde{\mathbf{w}}$. In this case, $t = t_1 = \tilde{t}_1 + ab$. Algorithm $\mathcal{B}$ now computes

$$[c^{L+1}t]_T = [c^{L+1}]_1 \cdot [\tilde{t}_1]_2 + [c^{L+1}]_1 \cdot [ab]_2.$$

     Next, algorithm $\mathcal{B}$ sets $[z_0]_2$ and $[v_{\ell,0}]_2$ for $\ell \in [L]$ as

$$\begin{aligned} [z_0]_2 &= \sum_{\ell\in[L]} \left(\mathbf{m}_\ell^\top \tilde{\mathbf{t}} \cdot [c^\ell]_2 + \mathbf{m}_\ell^\top \tilde{\mathbf{w}} \cdot [abc^\ell]_2\right) = \sum_{\ell\in[L]} [c^\ell \mathbf{m}_\ell^\top \mathbf{t}]_2 \\ [v_{\ell,0}]_2 &= \sum_{\substack{i\in[L] \\ i\neq\ell}} \left(\mathbf{m}_i^\top \tilde{\mathbf{t}} \cdot [c^{L+1-\ell+i}]_2 + \mathbf{m}_i^\top \tilde{\mathbf{w}} \cdot [abc^{L+1-\ell+i}]_2\right) = \sum_{\substack{i\in[L] \\ i\neq\ell}} [c^{L+1-\ell+i}\mathbf{m}_i^\top \mathbf{t}]_2. \end{aligned}$$

- Algorithm $\mathcal{B}$ samples $\tilde{v}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$ and sets

$$\begin{aligned}
[v]_1 &= [\tilde{v}]_1 - \mathrm{tg}^* \cdot [bc^{L+1}]_1 = [\tilde{v} - bc^{L+1} \cdot \mathrm{tg}^*]_1 \\
[v]_2 &= [\tilde{v}]_2 - \mathrm{tg}^* \cdot [bc^{L+1}]_2 = [\tilde{v} - bc^{L+1} \cdot \mathrm{tg}^*]_2 \\
[h]_1 &= [\tilde{h}]_1 + [bc^{L+1}]_1 = [\tilde{h} + bc^{L+1}]_1 \\
[h]_2 &= [\tilde{h}]_2 + [bc^{L+1}]_2 = [\tilde{h} + bc^{L+1}]_2.
\end{aligned}$$

Algorithm $\mathcal{B}$ now constructs the public parameters as

$$\mathsf{pp} = \left(\begin{array}{c} \mathcal{G},\ [\tau]_1,\ \{[\tau^j]_2\}_{j\in[B]},\ [v]_1,\ [v]_2,\ [h]_1,\ [h]_2,\ [c^{L+1}t]_T, \\ [z_0]_2,\ \{[v_{\ell,0}]_2\}_{\ell\in[L]},\ \{[c^\ell\tau^j]_1,\ [c^\ell\tau^j]_2\}_{\ell\in[2L],j\in[0,B]} \end{array} \right)$$

5. Next, to simulate the public keys for the honest users, algorithm $\mathcal{B}$ starts by sampling $\tilde{\alpha}_\ell, \tilde{w}'_\ell \xleftarrow{R} \mathbb{Z}_p$ for each $\ell \in [L] \setminus C$. Next, for each $\ell \in [L] \setminus C$, if $\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} = 0$, algorithm $\mathcal{B}$ sets $\tilde{y}^*_\ell = 0$. Otherwise, it samples $\tilde{y}^*_\ell \xleftarrow{R} \mathbb{Z}_p^*$. Then, algorithm $\mathcal{B}$ implicitly sets the exponents

$$\begin{aligned}
\alpha_\ell &= \tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} \\
w'_\ell &= \tilde{w}'_\ell + ab\tilde{y}^*_\ell.
\end{aligned}$$

Algorithm $\mathcal{B}$ then sets

$$\begin{aligned}
[c^{L+1}\alpha_\ell]_T &= [c^{L+1}]_1 \cdot [\tilde{\alpha}_\ell]_2 - \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} \cdot [c^{L+1}]_1 \cdot [ab]_2 \\
[w'_\ell]_1 &= [\tilde{w}'_\ell]_1 + \tilde{y}^*_\ell \cdot [ab]_1.
\end{aligned}$$

Finally, it defines the public key to be

$$\mathsf{pk}_\ell = (\mathcal{G}, [v]_1, [h]_1, [w']_1, [c^{L+1}\alpha]_T).$$

To construct the aggregation hints, algorithm $\mathcal{B}$ first computes for each $i \in [2L] \setminus \{L+1\}$ and $j \in [0, B]$,

$$[abc^i\tau^j]_2 = \sum_{k\in[0,j]} \binom{j}{k} \cdot (\mathsf{id}^*)^{j-k} \cdot [abc^i\hat{\tau}^k]_2 = [abc^i(\hat{\tau} + \mathsf{id}^*)^j]_2.$$

Then, for each $i \in [2L] \setminus \{L+1\}$ and $j \in [0, B]$, algorithm $\mathcal{B}$ computes the cross-terms

$$\begin{aligned}
[c^i\alpha_\ell]_2 &= \tilde{\alpha}_\ell \cdot [c^i]_2 - \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} \cdot [abc^i]_2 \\
[c^i w'_\ell \tau^j]_2 &= \tilde{w}'_\ell \cdot [c^i\tau^j]_2 + \tilde{y}^*_\ell \cdot [abc^i\tau^j]_2,
\end{aligned}$$

where $[c^i\tau^j]_2$ was computed in Eq. (5.8). It then sets $\mathsf{ht}_\ell = \{[c^i\alpha_\ell]_2, [c^i w'_\ell \tau^j]_2\}_{i\in[2L]\setminus\{L+1\}, j\in[0,B]}$. Algorithm $\mathcal{B}$ gives pp and $(\mathsf{pk}_\ell, \mathsf{ht}_\ell)$ for each $\ell \in [L] \setminus C$ to $\mathcal{A}$.

6. Whenever $\mathcal{A}$ makes a key-computation query on an index $\ell \in [L] \setminus C$, a set of identities $S \subseteq \mathbb{Z}_p$ where $|S| \le B$, and a batch label $\mathrm{tg} \in \mathbb{Z}_p$, algorithm $\mathcal{B}$ defines the following two polynomials over $\mathbb{Z}_p$:

$$\begin{aligned}
F_S(x) &= \prod_{\mathsf{id}\in S} (x - \mathsf{id}) \\
G_S(x) &= F_S(x + \mathsf{id}^*) - F_S(\mathsf{id}^*).
\end{aligned} \tag{5.9}$$

Write $F_S(x) = \sum_{i=[0,|S|]} \tilde{f}_i x^i$ and $G_S(x) = \sum_{i\in[|S|]} \tilde{g}_i x^i$. (By the same argument as in the proof of Theorem 4.4, the constant term of G_S is 0.) Algorithm $\mathcal{B}$ now proceeds as follows:

- If $\mathsf{tg} \neq \mathsf{tg}^*$, algorithm $\mathcal{B}$ samples $\tilde{y} \xleftarrow{R} \mathbb{Z}_p^*$ and $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$. Then, algorithm $\mathcal{B}$ computes the following:

$$\begin{aligned}
[u_1]_2 &= [\tilde{r}]_2 + (\mathsf{tg} - \mathsf{tg}^*)^{-1}(\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} - \tilde{y}\tilde{y}_\ell^* \cdot F_S(\mathsf{id}^*)) \cdot [a]_2 \\
[u_2]_2 &= \tilde{\alpha}_\ell \cdot [c^{L+1}]_2 + \tilde{r} \cdot (\mathsf{tg} - \mathsf{tg}^*) \cdot [bc^{L+1}]_2 + (\tilde{v} + \tilde{h} \cdot \mathsf{tg}) \cdot [u_1]_2 \\
&\quad + \tilde{y}\tilde{w}'_\ell \cdot F_S(\mathsf{id}^*) \cdot [c^{L+1}]_2 + \sum_{i \in [|S|]} \tilde{g}_i \cdot (\tilde{y}\tilde{w}'_\ell \cdot [c^{L+1}\hat{\tau}^i]_2 + \tilde{y}\tilde{y}_\ell^* \cdot [abc^{L+1}\hat{\tau}^i]_2).
\end{aligned}$$

- If $\mathsf{tg} = \mathsf{tg}^*$, algorithm $\mathcal{B}$ first sets $\tilde{y} = \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} / (\tilde{y}_\ell^* F_S(\mathsf{id}^*))$ if $\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} \neq 0$ and $\tilde{y} \xleftarrow{R} \mathbb{Z}_p^*$ if $\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} = 0$. Note that $\tilde{y}$ in the former case is well defined since $\tilde{y}_\ell^* \in \mathbb{Z}_p^*$, and when $\mathsf{tg} = \mathsf{tg}^*$, it must be the case that $\mathsf{id}^* \notin S$ so $F_S(\mathsf{id}^*) \neq 0$ by definition of F_S. Next, algorithm $\mathcal{B}$ samples $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$ and sets

$$\begin{aligned}
[u_1]_2 &= [\tilde{r}]_2 \\
[u_2]_2 &= \tilde{\alpha}_\ell \cdot [c^{L+1}]_2 + [\tilde{r}(\tilde{v} + \tilde{h} \cdot \mathsf{tg}^*)]_2 \\
&\quad + \tilde{y}\tilde{w}'_\ell \cdot F_S(\mathsf{id}^*) \cdot [c^{L+1}]_2 + \sum_{i \in [|S|]} \tilde{g}_i \cdot (\tilde{y}\tilde{w}'_\ell \cdot [c^{L+1}\hat{\tau}^i]_2 + \tilde{y}\tilde{y}_\ell^* [abc^{L+1}\hat{\tau}^i]_2).
\end{aligned}$$

  In both cases, algorithm $\mathcal{B}$ responds to $\mathcal{A}$ with the secret key $\mathsf{sk} = (\tilde{y}, [u_1]_2, [u_2]_2)$.

7. After $\mathcal{A}$ finished making key-computation queries, it specifies the key-generation randomness $\rho_\ell \in \{0,1\}^*$ for the corrupted users $\ell \in C$ along with two messages $[m_0]_\mathsf{T}, [m_1]_\mathsf{T} \in \mathbb{G}_\mathsf{T}$.

8. For each $\ell \in C$, algorithm $\mathcal{B}$ computes $(\mathsf{vk}_\ell, \mathsf{ht}_\ell, \mathsf{sk}_\ell) \leftarrow \mathsf{KeyGen}(\mathsf{crs}; \rho_\ell)$. Let $\tilde{\alpha}_\ell, \tilde{w}'_\ell \in \mathbb{Z}_p$ be the exponents associated with sk_ℓ. Then, algorithm $\mathcal{B}$ constructs the challenge ciphertext as follows:

$$\begin{aligned}
[\mathsf{ct}_1]_1 &= [s]_1 \\
[\mathsf{ct}_2]_2 &= \sum_{\ell \in [L]} \tilde{w}'_\ell \cdot [c^\ell s\hat{\tau}]_2 + \sum_{\ell \in [L] \setminus C} \tilde{y}_\ell^* \cdot [abc^\ell s\hat{\tau}]_2 \\
[\mathsf{ct}_3]_1 &= (\tilde{v} + \tilde{h} \cdot \mathsf{tg}^*) \cdot [s]_1 \\
[\mathsf{ct}_4]_2 &= \sum_{\ell \in [L]} (\mathbf{m}_\ell^\intercal \tilde{\mathbf{t}} + \tilde{\alpha}_\ell) \cdot [c^\ell s]_2 \\
[\mathsf{ct}_5]_\mathsf{T} &= \tilde{t}_1 \cdot [s]_1 \cdot [c^{L+1}]_2 + [\xi]_\mathsf{T} + [m_\beta]_\mathsf{T}.
\end{aligned}$$

  Algorithm $\mathcal{B}$ responds with the ciphertext $\mathsf{ct} = ([\mathsf{ct}_1]_1, [\mathsf{ct}_2]_2, [\mathsf{ct}_3]_1, [\mathsf{ct}_4]_2, [\mathsf{ct}_5]_\mathsf{T})$.

9. Algorithm $\mathcal{A}$ can continue to make key-computation queries. Algorithm $\mathcal{B}$ answers them using the same procedure as above.

10. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $\beta' \in \{0,1\}$, which algorithm $\mathcal{B}$ also outputs.

To complete the proof, we show that depending on the distribution of the challenge element ξ, algorithm $\mathcal{A}$ perfectly simulates either an execution of $\mathsf{Hyb}_0^{(\beta)}$ or $\mathsf{Hyb}_1^{(\beta)}$. We first consider the distribution of the public parameters:

- As described above, algorithm $\mathcal{B}$ constructs the public parameters by implicitly setting

$$\begin{aligned}
\tau &= \hat{\tau} + \mathsf{id}^* \\
v &= \tilde{v} - bc^{L+1} \cdot \mathsf{tg}^* \\
h &= \tilde{h} + bc^{L+1} \\
t &= \tilde{t}_1 + ab.
\end{aligned}$$

  and taking $c \in \mathbb{Z}_p$ to be the same value from the challenge. In addition, algorithm $\mathcal{B}$ samples $\tilde{v}, \tilde{h}, \tilde{t}_1 \xleftarrow{R} \mathbb{Z}_p$. Since the challenger samples $\hat{\tau}, c \xleftarrow{R} \mathbb{Z}_p$, the exponents τ, v, h, t, c match the distribution in $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$.

- Next, the reduction implicitly sets $\mathbf{t} = \tilde{\mathbf{t}} + ab \cdot \tilde{\mathbf{w}}$ where $\tilde{\mathbf{t}} \xleftarrow{R} \mathbb{Z}_p$ and $\tilde{t}_1 = t$. Since it samples $\tilde{\mathbf{t}} \xleftarrow{R} \mathbb{Z}_p^L$, this matches the distribution in $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$. For this choice of $\mathbf{t}$, algorithm $\mathcal{B}$ constructs z_0 and $v_{\ell,0}$ exactly as in $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$.

We conclude that the public parameters pp constructed by $\mathcal{B}$ are distributed identically to the public parameters in $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$. Next, consider the distribution of the public keys and the aggregation hints for the honest users $\ell \in [L] \setminus C$.

- As described, algorithm $\mathcal{B}$ simulates an execution of $\mathsf{KeyGen}(\mathsf{pp})$ where the underlying exponents α_ℓ, w'_ℓ are sampled as

$$\alpha_\ell = \tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\top \tilde{\mathbf{w}}$$
$$w'_\ell = \tilde{w}'_\ell + ab\tilde{y}^*_\ell,$$

  where $\tilde{\alpha}_\ell, \tilde{w}'_\ell \xleftarrow{R} \mathbb{Z}_p$. Thus, the distribution of α and w'_ℓ are also uniform random over $\mathbb{Z}_p$, exactly as in $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$. Moreover, we also note that w'_ℓ information-theoretically hides the value of $\tilde{y}^*_\ell$. This property will become important when we analyze the key-generation queries.

- By construction, the components of the public key pk_ℓ and the aggregation hint ht_ℓ are constructed according to the exact same relations as in the real scheme (with respect to the above choice of α_ℓ, w'_ℓ).

We conclude that all of the honest users' public keys and aggregation hints are distributed exactly as in the real scheme. Next, consider the distribution of the challenge ciphertext. We claim that algorithm $\mathcal{B}$ generates the challenge ciphertext according to the specification of $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$ where the encryption randomness $s \xleftarrow{R} \mathbb{Z}_p$ is the corresponding exponent sampled by the challenger. We consider each component individually:

- By construction, algorithm $\mathcal{B}$ sets $\mathsf{ct}_1 = s$, which matches the distribution in $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$.

- Consider ct_2. Recall first that for the honest users $\ell \in [L] \setminus C$, algorithm $\mathcal{B}$ sets $w'_\ell = \tilde{w}'_\ell + \tilde{y}^*_\ell ab$. For the keys for users $\ell \in C$ chosen adversarially, algorithm $\mathcal{B}$ defines $\tilde{w}'_\ell = w'_\ell$. In the reduction, algorithm $\mathcal{B}$ computes

$$\begin{aligned}
\mathsf{ct}_2 &= \sum_{\ell \in [L]} c^\ell s\hat{\tau}\tilde{w}'_\ell + \sum_{\ell \in [L]\setminus C} \tilde{y}^*_\ell abc^\ell s\hat{\tau} \\
&= \sum_{\ell \in [L]\setminus C} c^\ell s\hat{\tau}(\tilde{w}'_\ell + \tilde{y}^*_\ell ab) + \sum_{\ell \in C} c^\ell s\hat{\tau}\tilde{w}'_\ell \\
&= \sum_{\ell \in [L]\setminus C} c^\ell s\hat{\tau}w'_\ell + \sum_{\ell \in C} c^\ell s\hat{\tau}w'_\ell = s\hat{\tau} \sum_{\ell \in [L]} c^\ell w'_\ell.
\end{aligned}$$

  In the reduction, algorithm $\mathcal{B}$ implicitly defines $\tau = \hat{\tau} + \mathsf{id}^*$. This means

$$\mathsf{ct}_2 = s\hat{\tau} \sum_{\ell \in [L]} c^\ell w'_\ell = s(\tau - \mathsf{id}^*) \sum_{\ell \in [L]} c^\ell w'_\ell,$$

  which matches the distribution in $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$. Recall that in these experiments, the Preprocess algorithm would compute $[w]_2 = \sum_{\ell \in [L]} [c^\ell w'_\ell]_2$.

- Consider ct_3. In $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$, this component is set to $\mathsf{ct}_3 = s(v + h \cdot \mathsf{tg}^*)$. Substituting in algorithm $\mathcal{B}$'s choice of v and h, we have

$$\mathsf{ct}_3 = s(v + h \cdot \mathsf{tg}^*) = s(\tilde{v} - bc^{L+1} \cdot \mathsf{tg}^* + (\tilde{h} + bc^{L+1}) \cdot \mathsf{tg}^*) = s(\tilde{v} + \tilde{h} \cdot \mathsf{tg}^*),$$

  which is exactly how $\mathcal{B}$ constructs ct_3 in the reduction.

- Consider ct_4. In $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$, the experiment would set $\mathsf{ct}_4 = sz$ where

$$z = z_0 + \sum_{\ell \in [L]} c^\ell \alpha_\ell = \sum_{\ell \in [L]} c^\ell (\mathbf{m}_\ell^\top \mathbf{t} + \alpha_\ell).$$

  In the reduction, we have that $\mathbf{t} = \tilde{\mathbf{t}} + ab\tilde{\mathbf{w}}$. By construction $\mathbf{m}_\ell^\top \tilde{\mathbf{w}} = 0$ for all $\ell \in C$. In addition, the reduction algorithm defines $\alpha_\ell = \tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\top \tilde{\mathbf{w}}$ for all $\ell \in [L] \setminus C$ and sets $\tilde{\alpha}_\ell = \alpha_\ell$ for all $\ell \in C$. For this choice of variables, we can write

$$\begin{aligned} z &= \sum_{\ell \in [L]} c^\ell (\mathbf{m}_\ell^\top \mathbf{t} + \alpha_\ell) \\ &= \sum_{\ell \in C} c^\ell (\mathbf{m}_\ell^\top (\tilde{\mathbf{t}} + ab\tilde{\mathbf{w}}) + \tilde{\alpha}_\ell) + \sum_{\ell \in [L] \setminus C} c^\ell (\mathbf{m}_\ell^\top (\tilde{\mathbf{t}} + ab\tilde{\mathbf{w}}) + \tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\top \tilde{\mathbf{w}}) \\ &= \sum_{\ell \in [L]} c^\ell (\mathbf{m}_\ell^\top \tilde{\mathbf{t}} + \tilde{\alpha}_\ell). \end{aligned}$$

  In this case,

$$\mathsf{ct}_4 = sz = \sum_{\ell \in [L]} sc^\ell (\mathbf{m}_\ell^\top \tilde{\mathbf{t}} + \tilde{\alpha}_\ell),$$

  which is precisely how algorithm $\mathcal{B}$ constructs ct_4.

- Finally, consider the distribution of ct_5. We consider two possibilities depending on the distribution of ξ:
  - Suppose $\xi = abc^{L+1}s$. In the reduction, algorithm $\mathcal{B}$ implicitly sets $t = \tilde{t}_1 + ab$. In this case,

$$\mathsf{ct}_5 = s\tilde{t}_1 c^{L+1} + \xi + m_\beta = sc^{L+1}(\tilde{t}_1 + ab) + m_\beta = sc^{L+1}t + m_\beta,$$

    which is precisely the distribution of ct_5 in $\mathsf{Hyb}_0^{(\beta)}$.
  - If $\xi \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, then the distribution of ct_5 is also uniform over $\mathbb{Z}_p$. This coincides with the distribution in $\mathsf{Hyb}_1^{(\beta)}$.

Finally, consider the key-computation queries on an index $\ell \in [L] \setminus C$, a set of identities $S \subseteq \mathbb{Z}_p$ and a batch label $\mathsf{tg} \in \mathbb{Z}_p$. As in the reduction, we consider two cases:

- Suppose $\mathsf{tg} \neq \mathsf{tg}^*$. In this case, algorithm $\mathcal{B}$ samples $\tilde{y} \xleftarrow{\mathrm{R}} \mathbb{Z}_p^*$ and implicitly sets the randomness r to be

$$r = u_1 = \tilde{r} + (\mathsf{tg} - \mathsf{tg}^*)^{-1} (\mathbf{m}_\ell^\top \tilde{\mathbf{w}} - \tilde{y}\tilde{y}_\ell^* \cdot F_S(\mathsf{id}^*)) \cdot a,$$

  where $\tilde{r} \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. This matches the distribution of r in the real scheme. Thus, it suffices to argue that the component u_2 is correctly constructed (with respect to algorithm $\mathcal{B}$'s choice of r and $\tilde{y}$). In $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$, the experiment would compute the digest $\mathsf{dig} = [d]_2 = [c^{L+1} \cdot F_S(\tau)]_2$ and then set the variable

$$u_2 = c^{L+1}\alpha_\ell + r(v + h \cdot \mathsf{tg}) + c^{L+1}\tilde{y}w'_\ell \cdot F_S(\tau). \tag{5.10}$$

  In the reduction, algorithm $\mathcal{B}$ implicitly sets $\tau = \hat{\tau} + \mathsf{id}^*$. Thus, we can write

$$\begin{aligned} F_S(\tau) &= F_S(\mathsf{id}^*) + F_S(\tau) - F_S(\mathsf{id}^*) \\ &= F_S(\mathsf{id}^*) + F_S(\hat{\tau} + \mathsf{id}^*) - F_S(\mathsf{id}^*) \\ &= F_S(\mathsf{id}^*) + G_S(\hat{\tau}), \end{aligned}$$

  where G_S is the polynomial from Eq. (5.9). By definition of the coefficients $\tilde{g}_i$,

$$\sum_{i \in [|S|]} \tilde{g}_i \cdot (c^{L+1}\tilde{y}\tilde{w}'_\ell \hat{\tau}^i + abc^{L+1}\tilde{y}\tilde{y}_\ell^* \hat{\tau}^i) = c^{L+1}\tilde{y} \cdot (\tilde{w}'_\ell + ab\tilde{y}_\ell^*) \cdot G_S(\hat{\tau}) = c^{L+1}\tilde{y}w'_\ell \cdot G_S(\hat{\tau}).$$

Now, in the reduction, algorithm $\mathcal{B}$ sets

$$\begin{aligned} u_2 &= c^{L+1}\tilde{\alpha}_\ell + \tilde{r}(\mathrm{tg}-\mathrm{tg}^*)bc^{L+1} + (\tilde{v}+\tilde{h}\cdot\mathrm{tg})u_1 + c^{L+1}\tilde{y}\tilde{w}'_\ell \cdot F_S(\mathrm{id}^*) \\ &\quad + \sum_{i\in[|S|]} \tilde{g}_i \cdot (c^{L+1}\tilde{y}\tilde{w}'_\ell\hat{\tau}^i + abc^{L+1}\tilde{y}\tilde{y}^*_\ell\hat{\tau}^i) \\ &= c^{L+1}\tilde{\alpha}_\ell + \tilde{r}(\mathrm{tg}-\mathrm{tg}^*)bc^{L+1} + (\tilde{v}+\tilde{h}\cdot\mathrm{tg})u_1 + c^{L+1}\tilde{y}\tilde{w}'_\ell \cdot F_S(\mathrm{id}^*) + c^{L+1}\tilde{y}w'_\ell \cdot G_S(\hat{\tau}). \end{aligned} \tag{5.11}$$

Suppose now that we substitute the values of $\alpha_\ell, r, v, h, w'_\ell$ into Eq. (5.10). Then we have the following:

$$\begin{aligned} u_2 &= c^{L+1}\alpha_\ell + r(v+h\cdot\mathrm{tg}) + c^{L+1}\tilde{y}w'_\ell \cdot F_S(\tau) \\ &= c^{L+1}\alpha_\ell + u_1(v+h\cdot\mathrm{tg}) + c^{L+1}\tilde{y}w'_\ell \cdot (F_S(\mathrm{id}^*) + G_S(\hat{\tau})) \\ &= c^{L+1}(\tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\intercal\tilde{\mathbf{w}}) + u_1(v+h\cdot\mathrm{tg}) + c^{L+1}\tilde{y}(\tilde{w}'_\ell + ab\tilde{y}^*_\ell)\cdot F_S(\mathrm{id}^*) + c^{L+1}\tilde{y}w'_\ell \cdot G_S(\hat{\tau}). \end{aligned} \tag{5.12}$$

We have highlighted the terms that depend on abc^{L+1} in green since these terms will be cancelled out. Specifically, consider now the value of $u_1(v+h\cdot\mathrm{tg})$:

$$\begin{aligned} u_1(v+h\cdot\mathrm{tg}) &= u_1(\tilde{v} - bc^{L+1}\cdot\mathrm{tg}^* + (\tilde{h}+bc^{L+1})\cdot\mathrm{tg}) \\ &= u_1(\tilde{v}+\tilde{h}\cdot\mathrm{tg}) + u_1\cdot bc^{L+1}(\mathrm{tg}-\mathrm{tg}^*) \\ &= u_1(\tilde{v}+\tilde{h}\cdot\mathrm{tg}) + (\tilde{r}+(\mathrm{tg}-\mathrm{tg}^*)^{-1}(\mathbf{m}_\ell^\intercal\tilde{\mathbf{w}} - \tilde{y}\tilde{y}^*_\ell\cdot F_S(\mathrm{id}^*))\cdot a)\cdot bc^{L+1}(\mathrm{tg}-\mathrm{tg}^*) \\ &= u_1(\tilde{v}+\tilde{h}\cdot\mathrm{tg}) + \tilde{r}bc^{L+1}(\mathrm{tg}-\mathrm{tg}^*) + abc^{L+1}(\mathbf{m}_\ell^\intercal\tilde{\mathbf{w}} - \tilde{y}\tilde{y}^*_\ell\cdot F_S(\mathrm{id}^*)). \end{aligned}$$

Observe that the highlighted terms in green precisely cancels out the corresponding terms that depend on abc^{L+1} in Eq. (5.12). Thus, substituting back into Eq. (5.12), we now have

$$\begin{aligned} u_2 &= c^{L+1}(\tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\intercal\tilde{\mathbf{w}}) + u_1(v+h\cdot\mathrm{tg}) + c^{L+1}\tilde{y}(\tilde{w}'_\ell + ab\tilde{y}^*_\ell)\cdot F_S(\mathrm{id}^*) + c^{L+1}\tilde{y}w'_\ell \cdot G_S(\hat{\tau}) \\ &= c^{L+1}\tilde{\alpha}_\ell + u_1(\tilde{v}+\tilde{h}\cdot\mathrm{tg}) + \tilde{r}bc^{L+1}(\mathrm{tg}-\mathrm{tg}^*) + c^{L+1}\tilde{y}\tilde{w}'_\ell \cdot F_S(\mathrm{id}^*) + c^{L+1}\tilde{y}w'_\ell \cdot G_S(\hat{\tau}). \end{aligned}$$

This is precisely the expression in Eq. (5.11), so we conclude that algorithm $\mathcal{B}$ correctly answers the key-computation query according to the specification of $\mathrm{Hyb}_0^{(\beta)}$ and $\mathrm{Hyb}_1^{(\beta)}$.

- Suppose $\mathrm{tg} = \mathrm{tg}^*$. In this case, algorithm $\mathcal{B}$ sets $\tilde{y} = \mathbf{m}_\ell^\intercal\tilde{\mathbf{w}}/(\tilde{y}^*_\ell F_S(\mathrm{id}^*))$ if $\mathbf{m}_\ell^\intercal\tilde{\mathbf{w}} \neq 0$ and $\tilde{y} \xleftarrow{\mathrm{R}} \mathbb{Z}_p^*$ if $\mathbf{m}_\ell^\intercal\tilde{\mathbf{w}} = 0$. It also samples $\tilde{r} \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. By assumption, algorithm $\mathcal{A}$ makes at *most* one key-computation query to user ℓ on batch label tg^*. Moreover, as argued previously, the adversary's view in the reduction can be described as a function of w'_ℓ, which perfectly hides $\tilde{y}^*_\ell$. In the case where $\mathcal{B}$ samples $\tilde{y}^*_\ell \xleftarrow{\mathrm{R}} \mathbb{Z}_p^*$, the distribution of $\tilde{y}$ is uniform over $\mathbb{Z}_p^*$ (and independent of all other quantities in the adversary's view) in both cases. In the other case, algorithm $\mathcal{B}$ simply sample $\tilde{y} \xleftarrow{\mathrm{R}} \mathbb{Z}_p^*$.

  As in the previous case, it suffices now to show that the component u_2 is correctly computed (with respect to algorithm $\mathcal{B}$'s choice of r and $\tilde{y}$). As in the previous case, in $\mathrm{Hyb}_0^{(\beta)}$ and $\mathrm{Hyb}_1^{(\beta)}$, the experiment would set

$$u_2 = c^{L+1}\alpha_\ell + r(v+h\cdot\mathrm{tg}^*) + c^{L+1}\tilde{y}w'_\ell \cdot (F_S(\mathrm{id}^*) + G_S(\hat{\tau})). \tag{5.13}$$

  Now, in the reduction, algorithm $\mathcal{B}$ sets

$$\begin{aligned} u_2 &= c^{L+1}\tilde{\alpha}_\ell + \tilde{r}(\tilde{v}+\tilde{h}\cdot\mathrm{tg}^*) + c^{L+1}\tilde{y}\tilde{w}'_\ell \cdot F_S(\mathrm{id}^*) + \sum_{i\in[|S|]} \tilde{g}_i \cdot (\tilde{y}\tilde{w}'_\ell \cdot c^{L+1}\hat{\tau}^i + \tilde{y}\tilde{y}^*_\ell abc^{L+1}\hat{\tau}^i) \\ &= c^{L+1}\tilde{\alpha}_\ell + \tilde{r}(\tilde{v}+\tilde{h}\cdot\mathrm{tg}^*) + c^{L+1}\tilde{y}\tilde{w}'_\ell \cdot F_S(\mathrm{id}^*) + c^{L+1}\tilde{y}(\tilde{w}'_\ell + \tilde{y}^*_\ell ab)\cdot G_S(\hat{\tau}) \\ &= c^{L+1}\tilde{\alpha}_\ell + \tilde{r}(\tilde{v}+\tilde{h}\cdot\mathrm{tg}^*) + c^{L+1}\tilde{y}\tilde{w}'_\ell \cdot F_S(\mathrm{id}^*) + c^{L+1}\tilde{y}w'_\ell \cdot G_S(\hat{\tau}). \end{aligned} \tag{5.14}$$

Suppose now that we substitute the values of $\alpha_\ell, r, v, h, w'_\ell$ into Eq. (5.13). Then we have the following:

$$\begin{aligned} u_2 &= c^{L+1}\alpha_\ell + r(v + h \cdot \mathrm{tg}) + c^{L+1}\tilde{y}w'_\ell \cdot (F_S(\mathrm{id}^*) + G_S(\hat{\tau})) \\ &= c^{L+1}\alpha_\ell + \tilde{r}(\tilde{v} - bc^{L+1} \cdot \mathrm{tg}^* + (\tilde{h} + bc^{L+1}) \cdot \mathrm{tg}^*) + c^{L+1}\tilde{y}w'_\ell \cdot (F_S(\mathrm{id}^*) + G_S(\hat{\tau})) \\ &= c^{L+1}(\tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\intercal\tilde{\mathbf{w}}) + \tilde{r}(\tilde{v} + \tilde{h} \cdot \mathrm{tg}^*) + c^{L+1}\tilde{y}(\tilde{w}'_\ell + ab\tilde{y}^*_\ell) \cdot F_S(\mathrm{id}^*) + c^{L+1}\tilde{y}w'_\ell \cdot G_S(\hat{\tau}), \end{aligned} \tag{5.15}$$

where we have again highlighted the terms that depend on abc^{L+1}. When $\mathbf{m}_\ell^\intercal\tilde{\mathbf{w}} = 0$, then algorithm $\mathcal{B}$ also sets $\tilde{y}^*_\ell = 0$. In this case

$$-abc^{L+1}\mathbf{m}_\ell^\intercal\tilde{\mathbf{w}} + abc^{L+1}\tilde{y}\tilde{y}^*_\ell F_S(\mathrm{id}^*) = 0.$$

If $\mathbf{m}_\ell^\intercal\tilde{\mathbf{w}} \neq 0$, then algorithm $\mathcal{B}$ sets $\tilde{y} = \mathbf{m}_\ell^\intercal\tilde{\mathbf{w}}/(\tilde{y}^*_\ell F_S(\mathrm{id}^*))$. We can then write

$$-abc^{L+1}\mathbf{m}_\ell^\intercal\tilde{\mathbf{w}} + abc^{L+1}\tilde{y}\tilde{y}^*_\ell F_S(\mathrm{id}^*) = -abc^{L+1}\mathbf{m}_\ell^\intercal\tilde{\mathbf{w}} + abc^{L+1}\mathbf{m}_\ell^\intercal\tilde{\mathbf{w}} = 0.$$

Substituting back in Eq. (5.15), we have

$$\begin{aligned} u_2 &= c^{L+1}(\tilde{\alpha}_\ell - ab\mathbf{m}_\ell^\intercal\tilde{\mathbf{w}}) + \tilde{r}(\tilde{v} + \tilde{h} \cdot \mathrm{tg}^*) + c^{L+1}\tilde{y}(\tilde{w}'_\ell + ab\tilde{y}^*_\ell) \cdot F_S(\mathrm{id}^*) + c^{L+1}\tilde{y}w'_\ell \cdot G_S(\hat{\tau}) \\ &= c^{L+1}\tilde{\alpha}_\ell + \tilde{r}(\tilde{v} + \tilde{h} \cdot \mathrm{tg}^*) + c^{L+1}\tilde{y}\tilde{w}'_\ell \cdot F_S(\mathrm{id}^*) + c^{L+1}\tilde{y}w'_\ell \cdot G_S(\hat{\tau}), \end{aligned}$$

which precisely coincides with how algorithm $\mathcal{B}$ constructs u_2 in Eq. (5.14).

We conclude that algorithm $\mathcal{B}$ responds to the key-generation queries with the same procedure as in $\mathsf{Hyb}_0^{(\beta)}$ and $\mathsf{Hyb}_1^{(\beta)}$. Thus, as argued above, if $\xi = abc^{L+1}s$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_0^{(\beta)}$, whereas if $\xi \xleftarrow{R} \mathbb{Z}_p$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_1^{(\beta)}$. Thus, algorithm $\mathcal{B}$ breaks Assumption 5.4 with the same advantage ε and the claim follows. □

Corollary 5.9 (Threshold Batched Identity-Based Encryption with Silent Setup)**.** *Let λ be a security parameter. Suppose Assumption 5.4 holds with respect to* GroupGen *for all polynomials $B = B(\lambda)$ and $N = N(\lambda)$. Then, for every polynomial $B = B(\lambda)$ and $L = L(\lambda)$, Construction 5.5 is a statically-secure fixed-threshold batched IBE scheme with silent setup and the following efficiency properties:*

- **Public parameter size:** *For a batch size B and a decryption committee size L, the public parameters contain $O(LB)$ group elements.*
- **Ciphertext size:** *Each ciphertext contains 2 $\mathbb{G}_1$ elements, 2 $\mathbb{G}_2$ elements and 1 $\mathbb{G}_\mathsf{T}$ element.*
- **Aggregated encryption key size:** *The aggregated encryption key for a group of L users contains 2 $\mathbb{G}_1$ elements, 3 $\mathbb{G}_2$ elements, and 1 $\mathbb{G}_\mathsf{T}$ element.*
- **Digest size:** *A digest contains 1 $\mathbb{G}_2$ element.*
- **Decryption key size:** *A decryption key share contains 2 $\mathbb{G}_2$ elements and 1 $\mathbb{Z}_p$ element.*

Remark 5.10 (Supporting Dynamic Thresholds)**.** Construction 5.5 gives a scheme that supports a *fixed threshold* where the threshold is determined at setup time. We could also consider a more general setting where the threshold can be determined dynamically at *encryption* time. The work of [WW25a] provide a general template for building a scheme that supports dynamic thresholds from one that supports fixed thresholds. We sketch this approach here.

Let N be a bound on the size of the decryption committee. The [WW25a] approach instantiates the fixed threshold scheme with $2N - 1$ users and a fixed threshold of N. The first N users (or slots) are associated with potential decryption committee keys while the remaining $N - 1$ users are dummy users. The public parameters include a random *public* key for each dummy user while the associated secret key is discarded.

Suppose we want to support a decryption committee with L users. Let $\mathsf{pk}_1, \ldots, \mathsf{pk}_L$ denote their public keys. In Construction 5.5, the Preprocess algorithm now needs to compute the aggregated components $([z]_2, [w]_2, [w\tau]_2)$. The idea is that the Preprocess algorithm will only aggregate in the keys for the first N users, defined as follows:

- The public keys for the first L slots are $\mathsf{pk}_1, \ldots, \mathsf{pk}_L$.
- The public keys for the remaining $N - L$ slots are set to the all-zero key (e.g., $\alpha_\ell = w'_\ell = 0$). By construction, anyone can compute decryption key shares with respect to these keys.

Thus far, $([z]_2, [w]_2, [w\tau]_2)$ only contain information for the first N slots (for a scheme that supports $2N - 1$ users). The public keys for the remaining $N - 1$ slots are now determined at *encryption* time based on the threshold.

Specifically, suppose we want to encrypt to a threshold $T \le L \le N$. Since we have a fixed threshold scheme, any decrypter needs to accumulate N decryption shares in order to decrypt. A decrypter that has T decryption shares from the decryption committee could obtain an additional $N - L$ decryption shares (from the users with the zero keys) for a grand total of $N + T - L$ shares. At this point, they are still short by $L - T$ decryption shares. The idea in the [WW25a] approach is the encrypter will associate a set of $L - T$ users in slots $N + 1, \ldots, 2N - 1$ with the all-zeroes key and the remaining users with their dummy keys. Essentially then, any decrypter can get an additional $L - T$ decryption shares *for free*. This brings their total share count to $T + (N - L) + (L - T) = N$, thus allowing them to decrypt. In other words, the encryption algorithm will determine how many additional shares to give out for free based on the encryption threshold, and then encrypt to the associated collection of public keys.

Implementing this step naïvely will lead to long encryption keys (since we have to include dummy keys for up to $N - 1$ users in order to support arbitrary thresholds). The final idea in [WW25b] is to "pre-aggregate" the dummy keys together in blocks, where each block contains a different power-of-two pre-aggregated users. This allows us to support dynamic thresholds with only $\log N$ overhead in the size of the CRS and the size of the encryption key. The ciphertext size and the secret key size are unaffected. Static security of this adaptation would follow under Assumption 5.4 with parameters B and $2N - 1$.

Acknowledgments

We thank Amit Agarwal, Dan Boneh, and Abhishek Jain for drawing our attention to the batch decryption problem. This work was done in part while H.W. and D.W. were visiting the Simons Institute for the Theory of Computing, supported in part by a grant from the UC Noyce Initiative. J.G. is supported in part by National Natural Science Foundation of China (62372175) and Shanghai Pilot Program for Basic Research (TQ20240212).

References

[ADM+24] Gennaro Avitabile, Nico Döttling, Bernardo Magri, Christos Sakkas, and Stella Wohnig. Signature-based witness encryption with compact ciphertext. In *ASIACRYPT*, 2024.

[AFP25] Amit Agarwal, Rex Fernando, and Benny Pinkas. Efficiently-thresholdizable batched identity based encryption, with applications. In *CRYPTO*, 2025.

[BB04] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *CRYPTO*, 2004.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, 2005.

[BCF+25] Jan Bormet, Arka Rai Choudhuri, Sebastian Faust, Sanjam Garg, Hussien Othman, Guru-Vamsi Policharla, Ziyan Qu, and Mingyuan Wang. BEAST-MEV: Batched threshold encryption with silent setup for MEV prevention. *IACR Cryptol. ePrint Arch.*, 2025.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, 2001.

[BFOQ25] Jan Bormet, Sebastian Faust, Hussien Othman, and Ziyan Qu. BEAT-MEV: epochless approach to batched threshold encryption for MEV prevention. In *USENIX Security Symposium*, 2025.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO*, 2005.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *ASIACRYPT*, 2001.

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

[BO22] Joseph Bebel and Dev Ojha. Ferveo: Threshold decryption for mempool privacy in BFT networks. *IACR Cryptol. ePrint Arch.*, 2022.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, 2014.

[CGPP24] Arka Rai Choudhuri, Sanjam Garg, Julien Piet, and Guru-Vamsi Policharla. Mempool privacy via batched threshold encryption: Attacks and defenses. In *USENIX Security Symposium*, 2024.

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

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, 2003.

[CHW25] Jeffrey Champion, Yao-Ching Hsieh, and David J. Wu. Registered ABE and adaptively-secure broadcast encryption from succinct LWE. In *CRYPTO*, 2025.

[Coc01] Clifford C. Cocks. An identity based encryption scheme based on quadratic residues. In *Cryptography and Coding*, 2001.

[CW24] Jeffrey Champion and David J. Wu. Distributed broadcast encryption from lattices. In *TCC*, 2024.

[DGK+20] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *IEEE S&P*, 2020.

[DJWW25] Lalita Devadas, Abhishek Jain, Brent Waters, and David J. Wu. Succinct witness encryption for batch languages and applications. In *ASIACRYPT*, 2025.

[DKL+23] Nico Döttling, Dimitris Kolonelos, Russell W. F. Lai, Chuanwei Lin, Giulio Malavolta, and Ahmadreza Rahimi. Efficient laconic cryptography from learning with errors. In *EUROCRYPT*, 2023.

[DPY24] Pratish Datta, Tapas Pal, and Shota Yamada. Registered FE beyond predicates: (attribute-based) linear functions and more. In *ASIACRYPT*, 2024.

[EFF85] Paul Erdős, Peter Frankl, and Zoltán Füredi. Families of finite sets in which no set is covered by the union of r others. *Israel J. Math*, 51(1-2), 1985.

[EHK+13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge L. Villar. An algebraic framework for Diffie-Hellman assumptions. In *CRYPTO*, 2013.

[FFM+23] Danilo Francati, Daniele Friolo, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Daniele Venturi. Registered (inner-product) functional encryption. In *ASIACRYPT*, 2023.

[FKdP23] Dario Fiore, Dimitris Kolonelos, and Paola de Perthuis. Cuckoo commitments: Registration-based encryption and key-value map commitments for large spaces. In *ASIACRYPT*, 2023.

[FPTX25] Rex Fernando, Guru-Vamsi Policharla, Andrei Tonkikh, and Zhuolun Xiang. TrX: Encrypted mempools in high performance BFT protocols. 2025.

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered ABE, flexible broadcast, and more. In *CRYPTO*, 2023.

[GHM+19] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In *PKC*, 2019.

[GHMR18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. In *TCC*, 2018.

[GKMR23] Noemi Glaeser, Dimitris Kolonelos, Giulio Malavolta, and Ahmadreza Rahimi. Efficient registration-based encryption. In *ACM CCS*, 2023.

[GKPW24] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold encryption with silent setup. In *CRYPTO*, 2024.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, 2006.

[HLWW23] Susan Hohenberger, George Lu, Brent Waters, and David J. Wu. Registered attribute-based encryption. In *EUROCRYPT*, 2023.

[KLJD23] Alireza Kavousi, Duc Viet Le, Philipp Jovanovic, and George Danezis. BlindPerm: Efficient MEV mitigation with an encrypted mempool and permutation. *IACR Cryptol. ePrint Arch.*, 2023.

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. In *ASIACRYPT*, 2023.

[KS64] William H. Kautz and Richard C. Singleton. Nonrandom binary superimposed codes. *IEEE Trans. Inf. Theory*, 10(4), 1964.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, 2010.

[LWW25] George Lu, Brent Waters, and David J. Wu. Multi-authority registered attribute-based encryption. In *EUROCRYPT*, 2025.

[RSY21] Leonid Reyzin, Adam D. Smith, and Sophia Yakoubov. Turning HATE into LOVE: compact homomorphic ad hoc threshold encryption for scalable MPC. In *Cyber Security Cryptography and Machine Learning*, 2021.

[RY07] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In *EUROCRYPT*, 2007.

[SAA24] Sora Suegami, Shinsaku Ashizawa, and Shinsaku Ashizawa. Constant-cost batched partial decryption in threshold encryption. *IACR Cryptol. ePrint Arch.*, 2024.

[Sch80] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4), 1980.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11), 1979.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, 1984.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, 1997.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, 2005.

[TCZ+20] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan-Gueta, and Srinivas Devadas. Towards scalable threshold cryptosystems. In *IEEE S&P*, 2020.

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, 2005.

[WQZD10] Qianhong Wu, Bo Qin, Lei Zhang, and Josep Domingo-Ferrer. Ad hoc broadcast encryption. In *ACM CCS*, 2010.

[WW25a] Brent Waters and David J. Wu. Silent threshold cryptography from pairings: Expressive policies in the plain model. *IACR Cryptol. ePrint Arch.*, 2025.

[WW25b] Hoeteck Wee and David J. Wu. Unbounded distributed broadcast encryption and registered ABE from succinct LWE. In *CRYPTO*, 2025.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM*, 1979.

[ZZGQ23] Ziqi Zhu, Kai Zhang, Junqing Gong, and Haifeng Qian. Registered ABE via predicate encodings. In *ASIACRYPT*, 2023.

A The Batched IBE Scheme from [AFP25]

To facilitate a comparison with our approach, we provide a brief description of the batched IBE scheme from [AFP25] scheme using our notation. For ease of comparison, we also interchange the roles of $\mathbb{G}_1$ and $\mathbb{G}_2$ from their construction. Then, the master public key mpk, the ciphertext $\mathsf{ct}_{\mathsf{id},\mathsf{tg}}$ (for identity id and batch label tg), and the secret key $\mathsf{sk}_{S,\mathsf{tg}}$ (for set $S \subset \mathbb{Z}_p$ and batch label tg) have the following structure:

$$\begin{aligned}
\mathsf{mpk} &= ([\tau]_2, \ldots, [\tau^B]_2, [\alpha]_\mathsf{T}) \\
\mathsf{ct}_{\mathsf{id},\mathsf{tg}} &= ([s]_1, [s_0 + s\alpha]_1, [s_0(\tau - \mathsf{id})]_1, [s\alpha \cdot H(\mathsf{tg})]_\mathsf{T} + [m]_\mathsf{T}) \quad \text{where} \quad s, s_0 \xleftarrow{\mathrm{R}} \mathbb{Z}_p \\
\mathsf{sk}_{S,\mathsf{tg}} &= [\alpha \cdot (H(\mathsf{tg}) + F_S(\tau))]_2 \quad \text{where} \quad F_S(x) = \prod_{\mathsf{id} \in S} (x - \mathsf{id})
\end{aligned}$$

Observe that if $\mathsf{id} \in S$, then

$$[s\alpha \cdot H(\mathsf{tg})]_\mathsf{T} = \overbrace{[s]_1}^{\mathsf{ct}} \cdot (\overbrace{[\alpha \cdot (H(\mathsf{tg}) + F_S(\tau))]_2}^{\mathsf{sk}}) - \overbrace{[s_0 + s\alpha]_1}^{\mathsf{ct}} \cdot \overbrace{F_S(\tau)}^{\mathsf{pp}} + \overbrace{[s_0(\tau - \mathsf{id})]_1}^{\mathsf{ct}} \cdot \overbrace{F_{S \setminus \{\mathsf{id}\}}(\tau)}^{\mathsf{pp}}$$

Running times. Table 2 compares the computational cost between our batched IBE schemes and that of [AFP25]. We refer to Table 1 for a comparison of the parameter sizes.

Scheme	Encrypt	Decrypt
[AFP25]	$5T_E + T_P + T_H$	$BT_E + 3T_P$
Corollary 4.5	$6T_E$	$BT_E + 3T_P$
Corollary D.6	$4T_E + T_P + T_H$	$BT_E + 2T_P$

Table 2: Comparison of our batched IBE schemes with the [AFP25] scheme in terms of the computational costs for encryption and decryption. We use T_E, T_H, T_P to refer to the cost of an exponentiation, a hash operation, and a pairing operation, respectively. In all cases, we report the numbers of exponentiations, hash operations, and pairing operations, and do not consider any optimization.

B Generic Hardness of Bilinear Diffie-Hellman Variants

In this section, we show that the q-type assumptions we use in this work (Assumptions 4.1 and 5.4) hold unconditionally in the generic bilinear group model [Sho97, BBG05]. In the generic bilinear group model, we model a generic asymmetric bilinear group of prime order p with label space $\mathcal{L}$ as three random injective functions $\varphi_1, \varphi_2, \varphi_\mathsf{T} : \mathbb{Z}_p \to \mathcal{L}$. An algorithm in the generic asymmetric bilinear group model has access to the following two oracles:

- **Evaluation oracle:** On input two labels $\ell_1, \ell_2 \in \mathcal{L}$ and a group index $i \in \{1, 2, \mathsf{T}\}$, the evaluation oracle first checks that ℓ_1, ℓ_2 are in the image of φ_i. If not, the evaluation oracle outputs $\perp$. Otherwise, it returns $\varphi_i(\varphi_i^{-1}(\ell_1) + \varphi_i^{-1}(\ell_2))$.
- **Pairing oracle:** On input two labels $\ell_1, \ell_2 \in \mathcal{L}$, the pairing oracle first checks that ℓ_1 is in the image of φ_1 and ℓ_2 is in the image of φ_2. If not, the pairing oracle outputs $\perp$. Otherwise, it returns $\varphi_{\mathsf{T}}(\varphi_1^{-1}(\ell_1) \cdot \varphi_2^{-1}(\ell_2))$.

Boneh, Boyen, and Goh [BBG05] described a set of sufficient conditions for a cryptographic assumption to hold unconditionally in the generic bilinear group model. Below, we present a specialized version (that only involves elements in the base groups) that suffices for analyzing the assumptions we use in this work. Our presentation below is adapted from that of [WW25a, Appendix A].

Definition B.1 (Independence of Polynomials)**.** Let $\mathcal{P} = \{P_i\}_{i \in [k]}$ be a collection of n-variate polynomials $P_i \in \mathbb{Z}_p[X_1, \ldots, X_n]$. We say a polynomial $f \in \mathbb{Z}_p[X_1, \ldots, X_n]$ is dependent with respect to $\mathcal{P}$ if there exists coefficients $\alpha_0, \ldots, \alpha_k$ such that

$$f(X_1, \ldots, x_n) = \alpha_0 + \sum_{i \in [k]} \alpha_k P_k(X_1, \ldots, X_n).$$

Conversely, we say that f is independent with respect to $\mathcal{P}$ if f is not dependent with respect to $\mathcal{P}$.

Theorem B.2 (Generic Hardness in Prime-Order Groups [BBG05, Theorem A.2, adapted])**.** *Let p be a prime and $\mathcal{P} = \{P_i\}_{i \in [k]}$ and $\mathcal{Q} = \{Q_j\}_{j \in [m]}$ be two collections of n-variate polynomials $P_i, Q_j \in \mathbb{Z}_p[X_1, \ldots, X_n]$ and where $P_1 = Q_1 = 1$. Let $T \in \mathbb{Z}_p[X_1, \ldots, X_n]$ be a polynomial. For an adversary $\mathcal{A}$ and a bit $b \in \{0, 1\}$, we define the following distinguishing experiment in the generic asymmetric bilinear group of order p:*

- *At the beginning of the game, the challenger samples $x_1, \ldots, x_n \xleftarrow{R} \mathbb{Z}_p$. For each $i \in [k]$, it computes $\ell_i = \varphi_1(P_i(x_1, \ldots, x_n))$ and for each $j \in [m]$, it computes $\ell'_j = \varphi_2(Q_j(x_1, \ldots, x_n))$.*
- *If $b = 0$, the challenger computes $\tau = \varphi_{\mathsf{T}}(T(x_1, \ldots, x_n))$. If $b = 1$, the challenger samples $r \xleftarrow{R} \mathbb{Z}_p$ and sets $\tau = \varphi_{\mathsf{T}}(r)$.*

The challenger gives $(\ell_1, \ldots, \ell_k, \ell'_1, \ldots, \ell'_m, \tau)$ to $\mathcal{A}$. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$ which is the output of the experiment. Let $\mathcal{PQ} = \{P_i Q_j : i \in [k], j \in [m]\}$. Let d be a bound on the total degree of the polynomials in $\mathcal{PQ} \cup \{T\}$. If T is independent of $\mathcal{PQ}$, then for all adversaries $\mathcal{A}$ making at most q queries to the generic asymmetric bilinear group oracle, it holds that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \frac{(q + k + m + 1)^2 d}{p}.$$

Generic hardness of Assumption 4.1. First, we use Theorem B.2 to prove that Assumption 4.1 holds in the generic asymmetric bilinear group model.

Theorem B.3 (Generic Hardness of Assumption 4.1)**.** *Let $N \in \mathbb{N}$ and let $\mathcal{A}$ be any adversary for Assumption 4.1 with parameter N. If $\mathcal{A}$ makes at most q generic group oracle queries, then the advantage of $\mathcal{A}$ is at most $O(q^2 N^3)/p$ in the generic asymmetric bilinear group model. In particular, whenever $p > \lambda^{\omega(1)}$, the advantage of $\mathcal{A}$ is negligible for all polynomials $N, q = \mathrm{poly}(\lambda)$.*

Proof. We start by defining the sets of polynomials $\mathcal{P}, \mathcal{Q}$ and the challenge polynomial T associated with the challenge terms in Assumption 4.1. By construction, each polynomial is over the formal variables a, b, s, τ. Then, the polynomials are defined as follows:

$$\begin{aligned}
\mathcal{P} &= \{1, b, s, \tau, ab, ab\tau, abs\tau\} \\
\mathcal{Q} &= \{1, a, b, \tau, \ldots, \tau^N, ab\tau, \ldots, ab\tau^N\} \\
T &= abs.
\end{aligned}$$

To appeal to Theorem B.2, we need to show that T is independent with respect to the product $\mathcal{PQ}$. Since $\mathcal{P}$ and $\mathcal{Q}$ consist of monomials and T is also a monomial, it suffices to argue that $T \notin \mathcal{PQ}$. This follows by inspection. Namely, suppose we write $T = PQ$ where $P \in \mathcal{P}$. Then, the following holds:

- If $P \in \{1, b, s, ab\}$ and $PQ = T = abs$, then $Q \in \{abs, sa, ab, s\}$. By definition of $\mathcal{Q}$, this means $Q \notin \mathcal{Q}$.
- If $P \in \{\tau, ab\tau, abs\tau\}$ and $PQ = T = abs$, then Q must be a multiple of τ^{-1}, which means $Q \notin \mathcal{Q}$.

We conclude that T is independent with respect to $\mathcal{PQ}$. Finally, to invoke Theorem B.2, we compute the maximum degree d among polynomials in $\mathcal{PQ} \cup \{T\}$. By inspection, the maximum degree $d_{\mathcal{P}}$ of polynomials in $\mathcal{P}$ is $d_{\mathcal{P}} = 4$ and the maximum degree $d_{\mathcal{Q}}$ of polynomials in $\mathcal{Q}$ is $d_{\mathcal{Q}} = O(N)$. The degree of T is 3. Thus $d = O(N)$. Finally, $|\mathcal{P}| = O(1)$ and $|\mathcal{Q}| = O(N)$. The claim now follows by Theorem B.2. □

Generic hardness of Assumption 5.4. Similarly, we can also use Theorem B.2 to prove that Assumption 5.4 holds in the generic asymmetric bilinear group model.

Theorem B.4 (Generic Hardness of Assumption 5.4)**.** *Let $L, B \in \mathbb{N}$ and let $\mathcal{A}$ be any adversary for Assumption 5.4 with parameter L and B. If $\mathcal{A}$ makes at most q generic group oracle queries, then the advantage of $\mathcal{A}$ is at most $O(q^2L^3B^3)/p$ in the generic asymmetric bilinear group model. In particular, whenever $p > \lambda^{\omega(1)}$, the advantage of $\mathcal{A}$ is negligible for all polynomials $L, B, q = \mathrm{poly}(\lambda)$.*

Proof. We start by defining the sets of polynomials $\mathcal{P}, \mathcal{Q}$ and the challenge polynomial T associated with the challenge terms in Assumption 5.4. By construction, each polynomial is over the formal variables a, b, c, s, τ. Then, the polynomials are defined as follows:

$$\mathcal{P} = \left\{1, bc^{L+1}, ab, s, \tau, \{c^\ell \tau^j\}_{\ell \in [2L], j \in [0,B]}\right\}$$
$$\mathcal{Q} = \left\{1, a, b, bc^{L+1}, ab, \{c^\ell \tau^j\}_{\ell \in [0,2L], j \in [0,B]}, \{abc^\ell \tau^j\}_{\ell \in [2L] \setminus \{L+1\}, j \in [0,B]}, \{abc^{L+1}\tau^j\}_{j \in [B]}, \{c^\ell s, c^\ell s\tau, abc^\ell s\tau\}_{\ell \in [L]}\right\}$$
$$T = abc^{L+1}s.$$

To appeal to Theorem B.2, we need to show that T is independent with respect to the product $\mathcal{PQ}$. Since $\mathcal{P}$ and $\mathcal{Q}$ consist of monomials and T is also a monomial, it suffices to argue that $T \notin \mathcal{PQ}$. This follows by inspection. Namely, suppose we write $T = PQ$ where $P \in \mathcal{P}$. Then, the following holds:

- If $P \in \{1, bc^{L+1}, ab, s\}$ and $PQ = T = abc^{L+1}s$, then $Q \in \{abc^{L+1}s, as, c^{L+1}s, abc^{L+1}\}$. By definition of $\mathcal{Q}$, this means $Q \notin \mathcal{Q}$.
- If $P \in \{c^\ell\}_{\ell \in [2L]}$ and $PQ = T = abc^{L+1}s$, then Q must be a multiple of abs. This rules out all monomials in $\mathcal{Q}$ except those in $\{abc^\ell s\tau\}_{\ell \in [L]}$. However, all of these monomials introduce τ and this means $Q \notin \mathcal{Q}$.
- If $P \in \{\tau\} \cup \{c^\ell \tau^j\}_{\ell \in [2L], j \in [B]}$ and $PQ = T = abc^{L+1}s$, then Q must be a multiple of τ^{-j} for some $j \in [B]$. This means $Q \notin \mathcal{Q}$.

We conclude that T is independent with respect to $\mathcal{PQ}$. Finally, to invoke Theorem B.2, we compute the maximum degree d among polynomials in $\mathcal{PQ} \cup \{T\}$. By inspection, the maximum degree $d_{\mathcal{P}}$ of polynomials in $\mathcal{P}$ is $d_{\mathcal{P}} = O(L+B)$ and the maximum degree $d_{\mathcal{Q}}$ of polynomials in $\mathcal{Q}$ is $d_{\mathcal{Q}} = O(L+B)$. The degree of T is $L+4$. Thus $d = O(L+B)$. Finally, $|\mathcal{P}| = O(LB)$ and $|\mathcal{Q}| = O(LB)$. The claim now follows by Theorem B.2. □

C Adaptively-Secure Batched IBE in the Plain Model

Two of the limitations of our batched IBE scheme from Section 4 (Construction 4.2) are (1) the scheme is only proven to be selectively secure; and (2) the security game restricts the adversary to making a single key query for the challenge batch label. In this section, we show a straightforward generalization of Construction 4.2 that addresses these limitations:

- **Adaptive choice of identity.** Theorem 4.4 proves selective security of Construction 4.2 where the adversary has to commit to both the challenge identity id^* as well as the challenge batch label tg^* at the beginning of the security game. Here, we show a generalization that allows the adversary to *adaptively* choose the identity, but remains selective in the choice of batch label. As we discuss in Remark C.16, supporting an adaptive choice of

batch label is plausible using Waters' bit-by-bit approach of embedding the batch label [Wat05] in place of the Boneh-Boyen embedding [BB04] used in Construction 4.2. For ease of exposition, we just focus on handling the adaptive choice of identity rather than the batch label as the latter can plausibly be handled via standard techniques. We refer to this notion as "identity-adaptive" security.

- **Multiple key-generation queries.** Second, we show a simple approach that allows the adversary to request up to K keys for the challenge batch label tg^* for some a priori bounded K. Our approach adds K field elements to the secret keys and K group elements to the ciphertexts.

To achieve identity-adaptive security, we first introduce a more general functionality where users are associated with a *set* $I \subset \mathbb{Z}_p$ of identities (instead of a single identity $\mathsf{id} \in \mathbb{Z}_p$). Secret keys are still associated with a set $S \subset \mathbb{Z}_p$ as before. Decryption is possible whenever the user's set of identities I is a subset of S. We refer to this notion as a tag-based attribute-based encryption (ABE) scheme for subset policies. The key property is that when the sets of identities I are drawn from a *polynomial-size* universe, then we can prove adaptive security just by having the reduction guess an element $\mathsf{id} \in I$. Then, in Remark C.15, we describe how to combine the tag-based ABE scheme for subset policies with a cover-free set system [KS64, EFF85]. This yields a batched IBE scheme with identity-adaptive security. We start by introducing the formal notion of a tag-based attribute-based encryption scheme. As mentioned above, we also consider an extension where the adversary is allowed to request up to K keys for the challenge tag (the analog of the batch label in the batched IBE scheme).

Definition C.1 (Tag-Based Attribute-Based Encryption for Subset Policies)**.** A tag-based attribute-based encryption (ABE) scheme for subset policies Π_{TagABE} is a tuple of efficient algorithms $\Pi_{\mathsf{TagABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Digest}, \mathsf{ComputeKey}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: On input the security parameter $\lambda \in \mathbb{N}$, the setup algorithm outputs a set of public parameters pp. We assume that the public parameters (implicitly) specifies the message space $\mathcal{M}$, the identity space $\mathcal{I}$, and the tag space $\mathcal{T}$ for the encryption scheme.
- $\mathsf{KeyGen}(\mathsf{pp}, 1^N, 1^B, 1^K) \to (\mathsf{mpk}, \mathsf{msk})$: On input the public parameters pp, a bound on the number of identities N associated with each user, a bound on the maximum set size B, and a collusion bound K, the key-generation algorithm outputs a master public key mpk and a master secret key msk. We assume that mpk and msk also include an implicit description of the message space $\mathcal{M}$, the tag space $\mathcal{T}$, the identity space $\mathcal{I}$, the bound on the number of identities N associated with each user, the maximum set size B, and the collusion bound K.
- $\mathsf{Encrypt}(\mathsf{mpk}, m, I, \mathsf{tg}) \to \mathsf{ct}$: On input the master public key mpk, a message $m \in \mathcal{M}$, a set of identities $I \subseteq \mathcal{I}$, and a tag $\mathsf{tg} \in \mathcal{T}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Digest}(\mathsf{mpk}, S) \to \mathsf{dig}$: On input the master public key mpk and a set of identities $S \subseteq \mathcal{I}$, the digest algorithm outputs a digest dig. This algorithm is deterministic.
- $\mathsf{ComputeKey}(\mathsf{msk}, \mathsf{dig}, \mathsf{tg}) \to \mathsf{sk}$: On input the master secret key msk, a digest dig, and a tag tg, the key-computation algorithm outputs a secret key sk associated with dig and tg.
- $\mathsf{Decrypt}(\mathsf{mpk}, \mathsf{sk}, S, (I, \mathsf{tg}), \mathsf{ct}) \to m$: On input the master public key mpk, a secret key sk, a set of identities $S \subseteq \mathcal{I}$, a pair (I, tg), and a ciphertext ct, the decryption algorithm outputs a message $m \in \mathcal{M}$ (or possibly a special symbol $\perp$ to indicate decryption failed). This algorithm is deterministic.

We require Π_{TagABE} satisfy the following properties:

- **Correctness:** For all $\lambda, B, K \in \mathbb{N}$, all $N \le B$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda)$, all messages $m \in \mathcal{M}$, sets of identities $I \subset \mathcal{I}$ of size at most N, and tags $\mathsf{tg} \in \mathcal{T}$ (where $\mathcal{M}, \mathcal{I}, \mathcal{T}$ are the message, identity, and tag spaces associated with pp, respectively), all sets $S \subseteq \mathcal{I}$ of size at most B where $I \subseteq S$, we have

$$\Pr\left[\mathsf{Decrypt}(\mathsf{mpk}, \mathsf{sk}, S, (I, \mathsf{tg}), \mathsf{ct}) = m : \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, 1^N, 1^B, 1^K) \\ \mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, m, I, \mathsf{tg}) \\ \mathsf{dig} = \mathsf{Digest}(\mathsf{mpk}, S) \\ \mathsf{sk} \leftarrow \mathsf{ComputeKey}(\mathsf{msk}, \mathsf{dig}, \mathsf{tg}) \end{array}\right] = 1.$$

- **Adaptive security:** For a security parameter λ, a bound N on the number of identities associated with each user, a bound B on the maximum set size, a collusion bound K, a bit $\beta \in \{0,1\}$, and an adversary $\mathcal{A}$, we define the adaptive security game as follows:

  - The challenger starts by computing $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, 1^N, 1^B, 1^K)$. It gives $(1^\lambda, 1^N, 1^B, 1^K, \mathsf{pp}, \mathsf{mpk})$ to $\mathcal{A}$. Let $\mathcal{M}, \mathcal{I}, \mathcal{T}$ be the message space, identity space, and tag space associated with pp, respectively.
  - Algorithm $\mathcal{A}$ can now make key-computation queries. On each query, algorithm $\mathcal{A}$ specifies a set $S \subseteq \mathcal{I}$ of size at most $|S| \le B$, and a tag $\mathsf{tg} \in \mathcal{T}$. The challenger replies with a secret key $\mathsf{sk} \leftarrow \mathsf{ComputeKey}(\mathsf{msk}, \mathsf{Digest}(\mathsf{mpk}, S), \mathsf{tg})$.
  - After $\mathcal{A}$ is finished making key-computation queries, it outputs two messages $m_0, m_1 \in \mathcal{M}$ and a challenge pair (I^*, tg^*) where $\emptyset \neq I^* \subseteq \mathcal{I}$ and $|I^*| \le N$. The challenger responds with a challenge ciphertext $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, m_\beta, I^*, \mathsf{tg}^*)$.
  - Algorithm $\mathcal{A}$ can continue to make key-computation queries. The challenger answers the queries exactly as before.
  - At the end of the game, algorithm $\mathcal{A}$ outputs a bit $\beta' \in \{0,1\}$, which is the output of the experiment.

  We say an adversary $\mathcal{A}$ is admissible if the following two conditions hold:

  - Algorithm $\mathcal{A}$ makes at most K key-computation queries on the challenge tag tg^*.
  - Let $S_1, \ldots, S_K \subseteq \mathcal{I}$ be the sets associated with $\mathcal{A}$'s K key-computation queries on tg^*. It holds that $I^* \notin \bigcup_{j \in [K]} S_j$. Note that this is a stronger admissibility requirement than standard ABE for subset policies. We refer to Remark C.2 for additional discussion.

  We say Π_{TagABE} is secure if for all polynomials $N = N(\lambda)$, $B = B(\lambda)$, and $K = K(\lambda)$, where $N \le B$, and all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\beta' = 1 : \beta = 0] - \Pr[\beta' = 1 : \beta = 1]| = \mathsf{negl}(\lambda) \tag{C.1}$$

  in the above security game. We say Π_{TagABE} is secure for parameters $(N, B, K) = (N(\lambda), B(\lambda), K(\lambda))$, if the above holds for the specific functions N, B, K. We say Π_{TagABE} satisfies tag-selective security if Eq. (C.1) holds against all efficient adversaries that must declare the challenge tag tg^* at the beginning of security game (but is allowed to choose the challenge set I^* *adaptively*).

- **Succinctness:** There exists a universal polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda, B, K \in \mathbb{N}$, all $N \le B$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda)$, all $(\mathsf{mpk}, \mathsf{msk})$ in the support of $\mathsf{KeyGen}(\mathsf{pp}, 1^N, 1^B, 1^K)$, all digests dig in the support of $\mathsf{Digest}(\mathsf{mpk}, \cdot)$, all tags $\mathsf{tg} \in \mathcal{T}$ (where $\mathcal{T}$ is the tag space associated with pp), and all ciphertexts ct in the support of $\mathsf{Encrypt}(\mathsf{mpk}, \cdot, \cdot, \cdot)$, the running time of $\mathsf{ComputeKey}(\mathsf{msk}, \mathsf{dig}, \mathsf{tg})$, the size of the ciphertext ct, and the size of the digest dig is $\mathsf{poly}(\lambda, K)$, and in particular, independent of B and N.

Remark C.2 (Comparison to ABE)**.** Definition C.1 can be viewed as a special case of key-policy ABE [SW05, GPSW06] for the family of "tagged subset" policies. Specifically, we can associate each policy with a pair (S, tg) where $S \subseteq \mathcal{I}$ is a set of identities and tg is a tag. Similarly, the attribute associated with each ciphertext is a pair $(I, \mathsf{tg}_{\mathsf{ct}})$. We say the policy is satisfied if

$$\mathsf{tg}_{\mathsf{ct}} = \mathsf{tg} \quad \text{and} \quad I \subseteq S.$$

Note that we can equivalently view the scheme as a ciphertext-policy ABE scheme for "tagged superset" policies. The additional requirement Definition C.1 imposes is that the secret key associated with the policy (S, tg) and the ciphertext associated with the attribute $(I, \mathsf{tg}_{\mathsf{ct}})$ are *both* short. Standard key-policy ABE typically allows the size of the secret key to grow with the size of the policy description and the ciphertext to grow with the length of the attribute. For our applications to batched IBE and batch decryption, it will be important to support *succinct* ciphertexts *and* secret keys.

On the flip side, the security requirement in Definition C.1 is significantly more restrictive than that for standard ABE. Namely, we only require security against adversaries whose challenge set I^* is not contained in the *union* of the

sets $S_1, \ldots, S_K$ appearing in the key-generation queries associated with tag tg^*. In normal ABE, the restriction would be that $I^* \notin S_i$ for all $i \in [K]$ (i.e., the challenge attribute does not satisfy the policy associated with any *individual* key). As we discuss in Remark C.15, this weaker notion is already sufficient to construct a batched IBE scheme with security against an adversary that can adaptively choose the challenge identity.

Tag-based ABE construction for subset policies. We now describe how to extend Construction 4.2 to obtain a tag-based ABE scheme for subset policies. Simultaneously, we also incorporate the generalization to support giving out K decryption keys for each tag. The idea here is to replace the component $w \in \mathbb{Z}_p$ in Construction 4.2 with a *vector* $\mathbf{w} \in \mathbb{Z}_p^K$. We refer to Section 2.1 for a high-level description of our approach.

Construction C.3 (Tag-Based Attribute-Based Encryption for Subset Policies)**.** Take any polynomial $M = M(\lambda)$ where $M(\lambda) \le 2^\lambda$ for all $\lambda \in \mathbb{N}$. Let GroupGen be a prime-order bilinear group generator. We construct a tag-based ABE scheme for subset policies $\Pi_{\mathsf{TagABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Digest}, \mathsf{ComputeKey}, \mathsf{Decrypt})$ as follows:

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter λ, the setup algorithm samples $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and outputs the public parameters $\mathsf{pp} = \mathcal{G}$. The message space associated with pp is $\mathcal{M} = \mathbb{G}_\mathsf{T}$, the identity space is $\mathcal{I} = [M(\lambda)]$, and the tag space is $\mathcal{T} = \mathbb{Z}_p$.

- $\mathsf{KeyGen}(\mathsf{pp}, 1^N, 1^B, 1^K)$: On input the public parameters $\mathsf{pp} = \mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e)$, a bound on the number of identities N associated with each user, a bound on the maximum set size B, and a collusion bound K, the key-generation algorithm samples exponents $\tau, v, h, \alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and $\mathbf{w} \xleftarrow{\mathrm{R}} \mathbb{Z}_p^K$. It outputs the master public key

$$\mathsf{mpk} = \left(\mathcal{G}, [\tau]_1, [\tau]_2, [\tau^2]_2, \ldots, [\tau^B]_2, [\mathbf{w}]_1, [\mathbf{w}\tau]_1, \ldots, [\mathbf{w}\tau^N]_1, [v]_1, [h]_1, [\alpha]_\mathsf{T}\right) \tag{C.2}$$

  and the master secret key $\mathsf{msk} = (\mathbf{w}, v, h, \alpha)$.

- $\mathsf{Encrypt}(\mathsf{mpk}, [m]_\mathsf{T}, I, \mathsf{tg})$: On input the master public key mpk (parsed according to Eq. (C.2)), a message $[m]_\mathsf{T} \in \mathbb{G}_\mathsf{T}$, a set $I \subseteq [M]$ of size at most N, and a tag $\mathsf{tg} \in \mathbb{Z}_p$, the encryption algorithm samples $s \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. It then constructs the polynomial $F_I(x) = \prod_{\mathsf{id} \in I}(x - \mathsf{id})$ whose roots are the elements $\mathsf{id} \in I$. Write $F_I(x) = \sum_{i \in [0,|I|]} f_i x^i$. Then, compute

$$[\mathbf{ct}_2]_1 = \sum_{i \in [0,|I|]} f_i s [\mathbf{w}\tau^i]_1 = [s\mathbf{w} \cdot F_I(\tau)]_1.$$

  Then output the ciphertext

$$\begin{aligned}\mathsf{ct} &= \left([s]_1,\ [\mathbf{ct}_2]_1,\ s([v]_1 + \mathsf{tg} \cdot [h]_1),\ s[\alpha]_\mathsf{T} + [m]_\mathsf{T}\right) \\ &= \left([s]_1,\ [s\mathbf{w} \cdot F_I(\tau)]_1,\ [s(v + h \cdot \mathsf{tg})]_1,\ [s\alpha]_\mathsf{T} + [m]_\mathsf{T}\right).\end{aligned}$$

- $\mathsf{Digest}(\mathsf{mpk}, S)$: On input the master public key mpk (parsed according to Eq. (C.2)) and a set of identities $S \subseteq [M]$ where $|S| \le B$, the digest algorithm defines the polynomial $F_S(x) = \prod_{\mathsf{id} \in S}(x - \mathsf{id})$ whose roots are the elements $\mathsf{id} \in S$. Write $F_S(x) = \sum_{i \in [0,|S|]} f_i x^i$. Output the digest

$$\mathsf{dig} = \sum_{i \in [0,|S|]} f_i \cdot [\tau^i]_2 = [F_S(\tau)]_2.$$

- $\mathsf{ComputeKey}(\mathsf{msk}, \mathsf{dig}, \mathsf{tg})$: On input the master secret key $\mathsf{msk} = (\mathbf{w}, v, h, \alpha)$, a digest $\mathsf{dig} = [d]_2$, and a tag $\mathsf{tg} \in \mathbb{Z}_p$, the key-computation algorithm samples random $r \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and $\mathbf{y} \xleftarrow{\mathrm{R}} \mathbb{Z}_p^K$. Then, it outputs the secret key

$$\mathsf{sk} = \left(\mathbf{y},\ [r]_2,\ [\alpha + r(v + h \cdot \mathsf{tg})]_2 + \mathbf{y}^\mathsf{T}\mathbf{w} \cdot [d]_2\right).$$

- $\mathsf{Decrypt}(\mathsf{mpk}, \mathsf{sk}, S, (I, \mathsf{tg}), \mathsf{ct})$: On input the master public key mpk (parsed according to Eq. (C.2)), a secret key $\mathsf{sk} = (\mathbf{y}, [u_1]_2, [u_2]_2)$, two sets of identities $S, I \subseteq [M]$, a tag $\mathsf{tg} \in \mathbb{Z}_p$, and the ciphertext $\mathsf{ct} = ([\mathsf{ct}_1]_1, [\mathbf{ct}_2]_1, [\mathsf{ct}_3]_1, [\mathsf{ct}_4]_\mathsf{T})$, the decryption algorithm proceeds as follows:

– If $I \not\subseteq S$, output $\perp$. Otherwise, define the polynomial

$$F_{S\setminus I}(x) = \prod_{\mathsf{id}\in S\setminus I} (x - \mathsf{id}).$$

Compute $[F_{S\setminus I}(\tau)]_2 = \sum_{i\in[0,|S\setminus I|]} f_i[\tau^i]_2$, where $F_{S\setminus I}(x) = \sum_{i\in[0,|S\setminus I|]} f_i x^i$.

– Then it computes and outputs

$$[\mathsf{ct}_4]_\mathsf{T} - \big(([\mathsf{ct}_1]_1 \cdot [u_2]_2) - (\mathbf{y}^\mathsf{T} \cdot [\mathbf{ct}_2]_1 \cdot [F_{S\setminus I}(\tau)]_2) - ([\mathsf{ct}_3]_1 \cdot [u_1]_2)\big). \tag{C.3}$$

Theorem C.4 (Correctness)**.** *Construction C.3 is correct.*

Proof. Take any $\lambda, B, K \in \mathbb{N}$, $N \le B$ and any $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e)$ in the support of $\mathsf{GroupGen}(1^\lambda)$. Take any $[m]_\mathsf{T} \in \mathbb{G}_\mathsf{T}$, any tag $\mathsf{tg} \in \mathbb{Z}_p$, any set $I, S \subseteq [M]$ where $|I| \le N$ and $|S| \le B$ and $I \subseteq S$. Sample

$$\begin{aligned}(\mathsf{mpk}, \mathsf{msk}) &\leftarrow \mathsf{KeyGen}(\mathsf{pp}, 1^B, 1^N, 1^K)\\ \mathsf{ct} &\leftarrow \mathsf{Encrypt}(\mathsf{mpk}, [m]_\mathsf{T}, I, \mathsf{tg})\\ \mathsf{dig} &= \mathsf{Digest}(\mathsf{mpk}, S)\\ \mathsf{sk} &\leftarrow \mathsf{ComputeKey}(\mathsf{msk}, \mathsf{dig}, \mathsf{tg})\end{aligned}$$

By construction, this means

$$\begin{aligned}\mathsf{mpk} &= \big(\mathcal{G}, [\tau]_1, [\tau]_2, [\tau^2]_2, \ldots, [\tau^B]_2, [\mathbf{w}]_1, [\mathbf{w}\tau]_1, \ldots, [\mathbf{w}\tau^N]_1, [v]_1, [h]_1, [\alpha]_\mathsf{T}\big)\\ \mathsf{ct} &= \big([s]_1, [s\mathbf{w} \cdot F_I(\tau)]_1, [s(v + h \cdot \mathsf{tg})]_1, [s\alpha]_\mathsf{T} + [m]_\mathsf{T}\big)\\ \mathsf{sk} &= \big(\mathbf{y}, [r]_2\,,\ [\alpha + r(v + h \cdot \mathsf{tg}) + \mathbf{y}^\mathsf{T}\mathbf{w} \cdot F_S(\tau)]_2\big),\end{aligned}$$

where $F_I(x) = \prod_{\mathsf{id}\in I}(x - \mathsf{id})$ and $F_S(x) = \prod_{\mathsf{id}\in S}(x - \mathsf{id})$. Consider now $\mathsf{Decrypt}(\mathsf{mpk}, \mathsf{sk}, S, (I, \mathsf{tg}), \mathsf{ct})$. If we write $\mathsf{sk} = (\mathbf{y}, [u_1]_2, [u_2]_2)$, $\mathsf{ct} = ([\mathsf{ct}_1]_1, [\mathbf{ct}_2]_1, [\mathsf{ct}_3]_1, [\mathsf{ct}_4]_\mathsf{T})$ and $F_{S\setminus I}(x) = \prod_{\mathsf{id}\in S\setminus I}(x - \mathsf{id})$, then the decryption algorithm computes

$$\begin{aligned}\mathsf{ct}_1 \cdot u_2 &= \alpha s + s\mathbf{y}^\mathsf{T}\mathbf{w} \cdot F_S(\tau) + rs(v + h \cdot \mathsf{tg})\\ \mathbf{y}^\mathsf{T} \cdot \mathbf{ct}_2 \cdot F_{S\setminus I}(\tau) &= s\mathbf{y}^\mathsf{T}\mathbf{w} \cdot \prod_{\mathsf{id}\in I}(\tau - \mathsf{id}) \cdot \prod_{\mathsf{id}\in S\setminus I}(\tau - \mathsf{id})\\ &= s\mathbf{y}^\mathsf{T}\mathbf{w} \cdot \prod_{\mathsf{id}\in S}(\tau - \mathsf{id}) = s\mathbf{y}^\mathsf{T}\mathbf{w} \cdot F_S(\tau)\\ \mathsf{ct}_3 \cdot u_1 &= rs(v + h \cdot \mathsf{tg}).\end{aligned}$$

Here the second expression uses the fact that $I \subseteq S$ which means that $I \cap (S \setminus I) = \varnothing$ and $I \cup (S \setminus I) = S$. This means

$$\begin{aligned}\mathsf{ct}_1 \cdot u_2 - \mathbf{y}^\mathsf{T} \cdot \mathbf{ct}_2 \cdot F_{S\setminus I}(\tau) - \mathsf{ct}_3 \cdot u_1 &= \alpha s + s\mathbf{y}^\mathsf{T}\mathbf{w} \cdot F_S(\tau) + rs(v + h \cdot \mathsf{tg}) - s\mathbf{y}^\mathsf{T}\mathbf{w} \cdot F_S(\tau) - rs(v + h \cdot \mathsf{tg})\\ &= \alpha s.\end{aligned}$$

The decryption relation (Eq. (4.3)) now yields:

$$[\mathsf{ct}_4 - (\mathsf{ct}_1 \cdot u_2 - \mathbf{y}^\mathsf{T} \cdot \mathbf{ct}_2 \cdot F_{S\setminus I}(\tau) - \mathsf{ct}_3 \cdot u_1)]_\mathsf{T} = [s\alpha + m - \alpha s]_\mathsf{T} = [m]_\mathsf{T}$$

and correctness holds. □

Security of Construction C.3. Security of Construction C.3 relies on a variant of Assumption 4.1 which we state below. We show this assumption holds in the generic asymmetric bilinear group model in Theorem C.17.

Assumption C.5 (N-Bilinear Diffie-Hellman Exponent Variant)**.** Let GroupGen be a prime-order bilinear group generator. For a security parameter λ, a parameter $N \in \mathbb{N}$, and a bit $\beta \in \{0, 1\}$, we define the distribution $\mathcal{D}_{\lambda,N,\beta}$ as follows:

- Sample $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$. Sample exponents $a, b, s, \tau \xleftarrow{R} \mathbb{Z}_p$. Define

$$\mathsf{params} = \left(\begin{array}{c} 1^\lambda, \mathcal{G}, [b]_1, [s]_1, [ab]_1, \{[\tau^i]_1, [ab\tau^i]_1, [s\tau^i]_1, [abs\tau^i]_1\}_{i\in[N]} \\ [a]_2, [b]_2, \{[\tau^i]_2, [ab\tau^i]_2\}_{i\in[N]} \end{array} \right).$$

- If $\beta = 0$, let $z = abs \in \mathbb{Z}_p$ and if $\beta = 1$, sample $z \xleftarrow{R} \mathbb{Z}_p$. Output $(\mathsf{params}, [z]_T)$.

We say Assumption C.5 holds with respect to GroupGen and parameter $N = N(\lambda)$ if the distributions $\mathcal{D}_0 = \{\mathcal{D}_{\lambda,N(\lambda),0}\}_{\lambda\in\mathbb{N}}$ and $\mathcal{D}_1 = \{\mathcal{D}_{\lambda,N(\lambda),1}\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable.

Theorem C.6 (Tag-Selective Security)**.** *Take any polynomial $B = B(\lambda)$ and suppose Assumption C.5 with parameter B holds with respect to* GroupGen*. Then, for all polynomials $N = N(\lambda)$ and $K = K(\lambda)$ where $N \le B$, Construction C.3 satisfies tag-selective security with respect to parameters (N, B, K).*

Proof. The proof follows a very similar structure as the proof of Theorem 4.4. Specifically, let $\mathcal{A}$ be an efficient and admissible adversary for the selective security experiment for Construction C.3 with parameters (N, B, K). For ease of notation, we assume without loss of generality that $\mathcal{A}$ always makes K key-computation queries on the challenge tag tg^*. We define a sequence of hybrid experiments, each indexed by a bit $\beta \in \{0, 1\}$:

- $\mathsf{Hyb}_0^{(\beta)}$: This is the tag-selective security game with bit β. Specifically, in this experiment, the adversary starts by committing to the challenge tag tg^*.
- $\mathsf{Hyb}_1^{(\beta)}$: Same as $\mathsf{Hyb}_0^{(\beta)}$, except at the beginning of the experiment, the challenger samples an identity $\mathsf{id}^* \xleftarrow{R} [M]$. At the end of the experiment, the challenger defines the following quantities:
  - Let $S_1, \ldots, S_K \subseteq [M]$ be the sets associated with the adversary's key-computation queries on tag tg^*.
  - Let $\emptyset \neq I^* \subseteq [M]$ be the adversary's challenge set.

  Since the adversary is admissible, it must be the case that $I^* \setminus \bigcup_{j\in[K]} S_j \neq \emptyset$. Now, the challenger outputs 0 if id^* is *not* the minimum element in the set $I^* \setminus \bigcup_{j\in[K]} S_j$. Otherwise, the output is computed exactly as in $\mathsf{Hyb}_0^{(\beta)}$.
- $\mathsf{Hyb}_2^{(\beta)}$: Same as $\mathsf{Hyb}_1^{(\beta)}$, except at the end of the experiment, the the challenger defines the following quantities:
  - Let $\mathbf{y}_1, \ldots, \mathbf{y}_K \in \mathbb{Z}_p^K$ be the vectors the challenger samples when responding to key-computation queries with tag tg^*. For each $i \in [K]$, write $\mathbf{y}_i = [y_{i,1}, \ldots, y_{i,K}]$.
  - For each $i \in [K]$, let $\bar{\mathbf{y}}_i^\intercal = [y_{i,1}, \ldots, y_{i,K-1}] \in \mathbb{Z}_p^{K-1}$.

  At the end of the experiment, the challenger outputs 0 if the vectors $\bar{\mathbf{y}}_1, \ldots, \bar{\mathbf{y}}_{K-1} \in \mathbb{Z}_p^{K-1}$ are not linearly independent.
- $\mathsf{Hyb}_3^{(\beta)}$: Same as $\mathsf{Hyb}_2^{(\beta)}$, except when responding to the K^{th} key-computation queries with tag tg^*, the challenger now samples $y_{K,K} \xleftarrow{R} \mathbb{Z}_p^*$.
- $\mathsf{Hyb}_4^{(\beta)}$: Same as $\mathsf{Hyb}_3^{(\beta)}$ except the challenger changes how it samples $\mathbf{y}_i = (y_{i,1}, \ldots, y_{i,K})$ when responding to key-computation queries on the challenge tag tg^*. At the beginning of the experiment, the challenger samples $\mathbf{c} \xleftarrow{R} \mathbb{Z}_p^K$ where $\mathbf{c} = [c_1, \ldots, c_K]$. The challenger halts with output 0 if $c_K = 0$. Then, for each $i \in [K]$, let $S_i \subseteq [N]$ be the i^{th} key-computation query on tag tg^*. The challenger now changes how it computes the final component $y_{i,K}$ for each $\mathbf{y}_i$:
  - At the beginning of the experiment, the challenger samples $u_1, \ldots, u_{K-1} \xleftarrow{R} \mathbb{Z}_p$.
  - When responding to the i^{th} key-computation query on tag tg^* and set S_i, the challenger sets $y_{i,K} = -c_K^{-1}(u_i - 1/F_{S_i}(\mathsf{id}^*))$, where $F_{S_i}(x) = \prod_{\mathsf{id}\in S_i}(x - \mathsf{id})$. If $\mathsf{id}^* \in S_i$, then the challenger halts with output 0 exactly as in $\mathsf{Hyb}_3^{(\beta)}$. If $\mathsf{id}^* \notin S_i$, then $F_{S_i}(\mathsf{id}^*) \neq 0$ by construction.

– When responding to the K^{th} key-computation query on tag tg^* and set S_K, the challenger samples $\gamma_1, \ldots, \gamma_{K-1} \xleftarrow{R} \mathbb{Z}_p$ and sets $\bar{\mathbf{y}}_K = \sum_{j \in [K-1]} \gamma_j \bar{\mathbf{y}}_j$. Next, it computes $u_K = \sum_{j \in [K-1]} \gamma_j u_j$. If $u_K = 1/F_{S_K}(\mathsf{id}^*)$, then the challenger halts with output 0. Otherwise, it sets $y_{K,K} = -c_K^{-1}(u_K - 1/F_{S_K}(\mathsf{id}^*))$.

- $\mathsf{Hyb}_5^{(\beta)}$: Same as $\mathsf{Hyb}_4^{(\beta)}$, except the challenger sets $u_i = \bar{\mathbf{y}}_i^\top \bar{\mathbf{c}}$ for all $i \in [K-1]$, where $\bar{\mathbf{c}} = (c_1, \ldots, c_{K-1}) \in \mathbb{Z}_p^{K-1}$.
- $\mathsf{Hyb}_6^{(\beta)}$: Same as $\mathsf{Hyb}_5^{(\beta)}$, except for all $i \in [K]$, the challenger samples $\mathbf{y}_i \xleftarrow{R} \mathbb{Z}_p^K$ such that $\mathbf{y}_i^\top \mathbf{c} = 1/F_{S_i}(\mathsf{id}^*)$. Note that the challenger in this experiment still halts with output 0 if $y_{K,K} = 0$.
- $\mathsf{Hyb}_7^{(\beta)}$: Same as $\mathsf{Hyb}_6^{(\beta)}$, except when constructing the challenge ciphertext $\mathsf{ct} = ([\mathsf{ct}_1]_1, [\mathbf{ct}_2]_1, [\mathsf{ct}_3]_1, [\mathsf{ct}_4]_T)$, the challenger samples $\mathsf{ct}_4 \xleftarrow{R} \mathbb{Z}_p$. In this experiment, the adversary's view is independent of the message.

Let $\mathsf{Hyb}_i^{(\beta)}(\mathcal{A})$ denote the output of an execution of $\mathsf{Hyb}_i^{(\beta)}$ with adversary $\mathcal{A}$. We now analyze each adjacent pair of distributions.

Lemma C.7. *For all $\beta \in \{0,1\}$, $\Pr\left[\mathsf{Hyb}_0^{(\beta)}(\mathcal{A}) = 1\right] = M \cdot \Pr\left[\mathsf{Hyb}_1^{(\beta)}(\mathcal{A}) = 1\right]$.*

Proof. By construction, the output in $\mathsf{Hyb}_1^{(\beta)}$ is 1 if and only if the output in $\mathsf{Hyb}_0^{(\beta)}$ is 1, and moreover, id^* is the smallest element in the set $I^* \setminus \bigcup_{j \in [K]} S_j \subseteq [M]$. Since the challenger samples $\mathsf{id}^* \xleftarrow{R} [M]$ and id^* is independent of the view of the adversary, the probability that id^* is the smallest element of the set $I^* \setminus \bigcup_{j \in [K]} S_j \subseteq [M]$ is *exactly* $1/M$. The claim follows. □

Lemma C.8. *For all $\beta \in \{0,1\}$, $\left|\Pr\left[\mathsf{Hyb}_1^{(\beta)}(\mathcal{A}) = 1\right] - \Pr\left[\mathsf{Hyb}_2^{(\beta)}(\mathcal{A}) = 1\right]\right| \leq (K-1)/p$.*

Proof. The only difference between these two experiments is $\mathsf{Hyb}_2^{(\beta)}$ always outputs 0 if $\bar{\mathbf{y}}_1, \ldots, \bar{\mathbf{y}}_{K-1}$ are not linearly independent when $\bar{\mathbf{y}}_1, \ldots, \bar{\mathbf{y}}_{K-1} \xleftarrow{R} \mathbb{Z}_p^{K-1}$. This happens with probability $(K-1)/p$. This is because the determinant of the matrix $[\bar{\mathbf{y}}_1 \mid \cdots \mid \bar{\mathbf{y}}_{K-1}]$ is a multivariate polynomial of degree $K-1$, so it is zero with probability $(K-1)/p$ over the random choice of $\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_{K-1}$ by the Schwartz-Zippel lemma [Sch80, Zip79]. □

Lemma C.9. *For $\beta \in \{0,1\}$, $\left|\Pr\left[\mathsf{Hyb}_2^{(\beta)}(\mathcal{A}) = 1\right] - \Pr\left[\mathsf{Hyb}_3^{(\beta)}(\mathcal{A}) = 1\right]\right| \leq 1/p$.*

Proof. The only difference between these two distributions is the distribution of $y_{K,K}$ In $\mathsf{Hyb}_2^{(\beta)}$, the challenger samples $y_{K,K} \xleftarrow{R} \mathbb{Z}_p$ whereas in $\mathsf{Hyb}_3^{(\beta)}$, the challenger samples $y_{K,K} \xleftarrow{R} \mathbb{Z}_p^*$. The statistical distance between the uniform distribution over $\mathbb{Z}_p$ and $\mathbb{Z}_p^*$ is $1/p$. □

Lemma C.10. *For $\beta \in \{0,1\}$, $\left|\Pr\left[\mathsf{Hyb}_3^{(\beta)}(\mathcal{A}) = 1\right] - \Pr\left[\mathsf{Hyb}_4^{(\beta)}(\mathcal{A}) = 1\right]\right| \leq 3/p$.*

Proof. It suffices to consider the case where $\bar{\mathbf{y}}_1, \ldots, \bar{\mathbf{y}}_{K-1}$ are linearly independent. Otherwise, the challenger outputs 0 in both distributions. First, consider the distribution of $\bar{\mathbf{y}}_K \in \mathbb{Z}_p^{K-1}$ in $\mathsf{Hyb}_4^{(\beta)}$. Since $\bar{\mathbf{y}}_1, \ldots, \bar{\mathbf{y}}_{K-1}$ are linearly independent over $\mathbb{Z}_p^{K-1}$, they form a basis for $\mathbb{Z}_p^{K-1}$. Since the challenger samples $\gamma_1, \ldots, \gamma_{K-1} \xleftarrow{R} \mathbb{Z}_p$, the distribution of $\bar{\mathbf{y}}_K$ is uniform over $\mathbb{Z}_p^{K-1}$. Next, consider the distribution of $y_{i,K}$ in $\mathsf{Hyb}_4^{(\beta)}$ when $c_K \neq 0$ and $u_1 \neq 0$.

- Consider $i \in [K-1]$. In this case, the challenger samples $u_i \xleftarrow{R} \mathbb{Z}_p$ so the distribution of $u_i - 1/F_{S_i}(\mathsf{id}^*)$ is also uniform over $\mathbb{Z}_p$. Since $c_K \neq 0$, scaling by $-c_K^{-1}$ does not affect the distribution. Thus, the distribution of $y_{i,K}$ is uniform over $\mathbb{Z}_p$ and moreover, *independent* of the value of c_K.
- Consider $y_{K,K}$. If $u_1 \neq 0$, then the probability that $u_K = \sum_{j \in [K-1]} \gamma_j u_j = 1/F_{S_K}(\mathsf{id}^*)$ is precisely $1/p$ (over the random choice of $\gamma_1 \xleftarrow{R} \mathbb{Z}_p$). If $u_K \neq 1/F_{S_K}(\mathsf{id}^*)$, then the distribution of $-c_K^{-1}(u_K - 1/F_{S_K}(\mathsf{id}^*))$ is uniform over $\mathbb{Z}_p^*$ over the randomness of $c_K \xleftarrow{R} \mathbb{Z}_p^*$.

We conclude that the distribution of $(y_{1,K}, \ldots, y_{K,K})$ in $\mathsf{Hyb}_4^{(\beta)}$ is distributed exactly as in $\mathsf{Hyb}_3^{(\beta)}$ unless $c_K = 0$, $u_1 = 0$, or $u_K = 1/F_{S_K}(\mathsf{id}^*)$. Over the randomness of c_K, u_1, γ_1 and taking a union bound, these events happen with probability at most $3/p$, and the claim follows. □

Lemma C.11. *For $\beta \in \{0,1\}$, $\Pr\left[\mathsf{Hyb}_4^{(\beta)}(\mathcal{A}) = 1\right] = \Pr\left[\mathsf{Hyb}_5^{(\beta)}(\mathcal{A}) = 1\right]$.*

Proof. It suffices to consider the case where $\bar{\mathbf{y}}_1, \ldots, \bar{\mathbf{y}}_K$ are linearly independent. Otherwise, the challenger outputs 0 in both distributions. However, if $\bar{\mathbf{y}}_1, \ldots, \bar{\mathbf{y}}_{K-1}$ are linearly independent, then the distribution of $[\bar{\mathbf{y}}_1 \mid \cdots \mid \bar{\mathbf{y}}_{K-1}]^\mathsf{T}\bar{\mathbf{c}}$ with $\bar{\mathbf{c}} \xleftarrow{\mathrm{R}} \mathbb{Z}_p^{K-1}$ is uniform over $\mathbb{Z}_p^{K-1}$ (since the matrix $[\bar{\mathbf{y}}_1 \mid \cdots \mid \bar{\mathbf{y}}_{K-1}]^\mathsf{T}$ is invertible). Thus, these distributions are identical. □

Lemma C.12. *For $\beta \in \{0,1\}$, $\Pr\left[\mathsf{Hyb}_5^{(\beta)}(\mathcal{A}) = 1\right] = \Pr\left[\mathsf{Hyb}_6^{(\beta)}(\mathcal{A}) = 1\right]$.*

Proof. It suffices to consider the case where $c_K \neq 0$ and $\bar{\mathbf{y}}_1, \ldots, \bar{\mathbf{y}}_{K-1}$ are linearly independent. Otherwise, both experiments output 0. By definition, the distribution of each $\mathbf{y}_i$ in $\mathsf{Hyb}_6^{(\beta)}$ is equivalent to the following sampling procedure:

- Sample $\bar{\mathbf{y}}_i \xleftarrow{\mathrm{R}} \mathbb{Z}_p^{K-1}$. Output $\mathbf{y}_i^\mathsf{T} = [\bar{\mathbf{y}}_i^\mathsf{T} \mid y_{i,K}]$ where $y_{i,K} \in \mathbb{Z}_p$ is chosen so that $\mathbf{y}^\mathsf{T}\mathbf{c} = 1/F_{S_i}(\mathsf{id}^*)$.

By construction, this means $y_{i,K} = -c_K^{-1}(\bar{\mathbf{y}}_i^\mathsf{T}\bar{\mathbf{c}} - 1/F_{S_i}(\mathsf{id}^*))$. This is precisely how the challenger constructs $\mathbf{y}_i$ for $i \in [K-1]$ in $\mathsf{Hyb}_5^{(\beta)}$. It suffices to consider the distribution of $\mathbf{y}_K$ in $\mathsf{Hyb}_5^{(\beta)}$:

- In $\mathsf{Hyb}_5^{(\beta)}$, the challenger sets $\bar{\mathbf{y}}_K = \sum_{j\in[K-1]} \gamma_j \bar{\mathbf{y}}_j$, where $\gamma_1, \ldots, \gamma_{K-1} \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. Since $\bar{\mathbf{y}}_1, \ldots, \bar{\mathbf{y}}_{K-1}$ are linearly independent, this means $\bar{\mathbf{y}}_K$ is uniform over $\mathbb{Z}_p^{K-1}$.
- Consider now the value of $y_{K,K}$. The challenger in $\mathsf{Hyb}_5^{(\beta)}$ sets $y_{K,K} = -c_K^{-1}(u_K - 1/F_{S_K}(\mathsf{id}^*))$ where

$$u_K = \sum_{j\in[K-1]} \gamma_j u_j = \sum_{j\in[K-1]} \gamma_j \bar{\mathbf{y}}_j^\mathsf{T}\bar{\mathbf{c}} = \left(\sum_{j\in[K-1]} \gamma_j \bar{\mathbf{y}}_j^\mathsf{T}\right)\bar{\mathbf{c}} = \bar{\mathbf{y}}_K^\mathsf{T}\bar{\mathbf{c}}.$$

- We conclude that in $\mathsf{Hyb}_5^{(\beta)}$, the distribution of $\bar{\mathbf{y}}_K$ is uniform over $\mathbb{Z}_p^{K-1}$ and $y_{K,K} = -c_K^{-1}(\bar{\mathbf{y}}_K^\mathsf{T}\bar{\mathbf{c}} - 1/F_{S_K}(\mathsf{id}^*))$, which is precisely the distribution in $\mathsf{Hyb}_6^{(\beta)}$. Note that both experiments output 0 if $y_{K,K} = 0$. □

Lemma C.13. *Suppose Assumption C.5 with parameter B holds with respect to* GroupGen. *Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_6^{(\beta)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_7^{(\beta)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.*

Proof. Suppose there exists a non-negligible ε such that

$$\left|\Pr[\mathsf{Hyb}_6^{(\beta)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_7^{(\beta)}(\mathcal{A}) = 1]\right| \geq \varepsilon.$$

We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks Assumption C.5 with parameter B and the same advantage ε:

1. At the beginning of the game, algorithm $\mathcal{B}$ receives a challenge $(\mathsf{params}, [z]_\mathsf{T})$ where

$$\mathsf{params} = \left(\begin{array}{c} 1^\lambda, \mathcal{G}, [b]_1, [s]_1, [ab]_1, \{[\hat{\tau}^i]_1, [ab\hat{\tau}^i]_1, [s\hat{\tau}^i]_1, [abs\hat{\tau}^i]_1\}_{i\in[B]} \\ [a]_2, [b]_2, \{[\hat{\tau}^i]_2, [ab\hat{\tau}^i]_2\}_{i\in[B]} \end{array}\right).$$

   and either $z = abs$ or $z \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. We use $\hat{\tau}$ to denote the powers-of-τ that appear in the assumption since the reduction algorithm below will program id^* into the simulated powers-of-τ.

2. Algorithm $\mathcal{B}$ sets $\mathsf{pp} = \mathcal{G}$ and gives $\mathcal{G}$ to $\mathcal{A}$. Algorithm $\mathcal{A}$ now commits to a tag $\mathsf{tg}^* \in \mathbb{Z}_p$.

3. Algorithm $\mathcal{B}$ samples an identity $\mathsf{id}^* \xleftarrow{\mathrm{R}} [M]$ and constructs the public key as follows. In the following description, we will use a "tilde" (e.g., $\tilde{\alpha}, \tilde{v}$) to denote an exponent that is chosen by (or otherwise known to) the reduction algorithm.

- Algorithm $\mathcal{B}$ implicitly sets $\tau = \hat{\tau} + \mathsf{id}^*$. For each $i \in [B]$, algorithm $\mathcal{B}$ computes

$$[\tau^i]_2 = \sum_{j \in [0,i]} \binom{i}{j} \cdot (\mathsf{id}^*)^{i-j} \cdot [\hat{\tau}^j]_2 = [(\hat{\tau} + \mathsf{id}^*)^i]_2.$$

  Similarly, it sets $[\tau]_1 = [\hat{\tau}]_1 + [\mathsf{id}^*]_1 = [\hat{\tau} + \mathsf{id}^*]_1$.
- Algorithm $\mathcal{B}$ samples $\tilde{\alpha} \xleftarrow{R} \mathbb{Z}_p$ and implicitly sets $\alpha = \tilde{\alpha} - ab$:

$$[\alpha]_T = [\tilde{\alpha}]_T - [ab]_1 \cdot [1]_2 = [\tilde{\alpha} - ab]_T.$$

- Algorithm $\mathcal{B}$ samples $\tilde{\mathbf{w}} \xleftarrow{R} \mathbb{Z}_p^K$ and $\tilde{\mathbf{c}} \xleftarrow{R} \mathbb{Z}_p^K$. If $\tilde{c}_K = 0$, then algorithm $\mathcal{B}$ outputs 0. Otherwise, it implicitly sets $\mathbf{w} = \tilde{\mathbf{w}} + ab\tilde{\mathbf{c}}$ and computes

$$[\mathbf{w}]_1 = [\tilde{\mathbf{w}}]_1 + \tilde{\mathbf{c}} \cdot [ab]_1 = [\tilde{\mathbf{w}} + ab\tilde{\mathbf{c}}]_1.$$

  Then, for each $i \in [N]$, algorithm $\mathcal{B}$ computes

$$[\mathbf{w}\tau^i]_1 = \sum_{k \in [0,i]} \binom{i}{k} \cdot (\mathsf{id}^*)^{i-k} \cdot (\tilde{\mathbf{w}} \cdot [\hat{\tau}^k]_1 + \tilde{\mathbf{c}} \cdot [ab\hat{\tau}^k]_1) = [\mathbf{w}(\hat{\tau} + \mathsf{id}^*)^i]_1.$$

  Recall here that $N \le B$, so the terms $[\hat{\tau}^k]_1$ and $[ab\hat{\tau}^i]_1$ are all available in params.
- Algorithm $\mathcal{B}$ samples $\tilde{v}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$ and implicitly sets $v = \tilde{v} - b \cdot \mathsf{tg}^*$ and $h = \tilde{h} + b$. Concretely, algorithm $\mathcal{B}$ defines

$$[v]_1 = [\tilde{v}]_1 - \mathsf{tg}^* \cdot [b]_1 = [\tilde{v} - b \cdot \mathsf{tg}^*]_1$$
$$[h]_1 = [\tilde{h}]_1 + [b]_1 = [\tilde{h} + b]_1.$$

Algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with the master public key

$$\mathsf{mpk} = \big(\mathcal{G}, [\tau]_1, [\tau]_2, [\tau^2]_2, \ldots, [\tau^B]_2, [\mathbf{w}]_1, [\mathbf{w}\tau]_1, \ldots, [\mathbf{w}\tau^N]_1, [v]_1, [h]_1, [\alpha]_T\big).$$

4. When algorithm $\mathcal{A}$ makes a key-computation query on a set of identities $S \subseteq [M]$ where $|S| \le B$ and a tag $\mathsf{tg} \in \mathbb{Z}_p$, algorithm $\mathcal{B}$ defines the following two polynomials over $\mathbb{Z}_p$:

$$\begin{aligned} F_S(x) &= \prod_{\mathsf{id} \in S} (x - \mathsf{id}) \\ G_S(x) &= F_S(x + \mathsf{id}^*) - F_S(\mathsf{id}^*). \end{aligned} \tag{C.4}$$

Write $F_S(x) = \sum_{i=[0,|S|]} \tilde{f}_i x^i$. Next, observe that the constant term of $G_S(x)$ is $G_S(0) = F_S(\mathsf{id}^*) - F_S(\mathsf{id}^*) = 0$. This means $G_S(x) = \sum_{i \in [|S|]} \tilde{g}_i x^i$. Algorithm $\mathcal{B}$ now proceeds as follows:

- If $\mathsf{tg} \ne \mathsf{tg}^*$, algorithm $\mathcal{B}$ samples $\tilde{\mathbf{y}} \xleftarrow{R} \mathbb{Z}_p^K$ and $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$. Then, algorithm $\mathcal{B}$ sets

$$\begin{aligned} [u_1]_2 &= [\tilde{r}]_2 + (\mathsf{tg} - \mathsf{tg}^*)^{-1}\big(1 - \tilde{\mathbf{y}}^\top \tilde{\mathbf{c}} \cdot F_S(\mathsf{id}^*)\big) \cdot [a]_2 \\ [u_2]_2 &= [\tilde{\alpha}]_2 + \tilde{r} \cdot (\mathsf{tg} - \mathsf{tg}^*) \cdot [b]_2 + (\tilde{v} + \tilde{h} \cdot \mathsf{tg}) \cdot [u_1]_2 \\ &\quad + \sum_{i \in [|S|]} \tilde{\mathbf{y}}^\top \tilde{\mathbf{w}} \cdot \tilde{g}_i \cdot [\hat{\tau}^i]_2 + \sum_{i \in [|S|]} \tilde{\mathbf{y}}^\top \tilde{\mathbf{c}} \cdot \tilde{g}_i \cdot [ab\hat{\tau}^i]_2 + [\tilde{\mathbf{y}}^\top \tilde{\mathbf{w}} \cdot F_S(\mathsf{id}^*)]_2. \end{aligned}$$

- If $\mathsf{tg} = \mathsf{tg}^*$, algorithm $\mathcal{B}$ first checks if $\mathsf{id}^* \in S$. If so, it outputs 0. Otherwise, it samples $\tilde{\mathbf{y}} \xleftarrow{R} \mathbb{Z}_p^K$ conditioned on $\tilde{\mathbf{y}}^\top \mathbf{c} = 1/F_S(\mathsf{id}^*)$. Next, algorithm $\mathcal{B}$ samples $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$ and sets It then computes

$$\begin{aligned} [u_1]_2 &= [\tilde{r}]_2 \\ [u_2]_2 &= [\tilde{\alpha} + \tilde{r}(\tilde{v} + \tilde{h} \cdot \mathsf{tg}^*)]_2 + \sum_{i \in [|S|]} \tilde{\mathbf{y}}^\top \tilde{\mathbf{w}} \cdot \tilde{g}_i \cdot [\hat{\tau}^i]_2 + \sum_{i \in [|S|]} \tilde{\mathbf{y}}^\top \tilde{\mathbf{c}} \cdot \tilde{g}_i \cdot [ab\hat{\tau}^i]_2 + [\tilde{\mathbf{y}}^\top \tilde{\mathbf{w}} \cdot F_S(\mathsf{id}^*)]_2 \end{aligned}$$

  If this is the K^{th} query that algorithm $\mathcal{A}$ makes with tag tg^*, algorithm $\mathcal{B}$ also checks that $\tilde{y}_K \ne 0$. Otherwise, it halts with outputs 0.

In both cases, algorithm $\mathcal{B}$ responds to $\mathcal{A}$ with the secret key $\mathsf{sk} = (\tilde{\mathbf{y}}, [u_1]_2, [u_2]_2)$.

5. In the challenge phase, algorithm $\mathcal{A}$ outputs two messages $[m_0]_T$ and $[m_1]_T$ and a challenge set $I^* \subseteq [M]$. If $\mathsf{id}^* \notin I^*$, then algorithm $\mathcal{B}$ halts with output 0. Algorithm $\mathcal{B}$ defines the following two polynomials over $\mathbb{Z}_p$:

$$\begin{aligned} F_{I^*}(x) &= \prod_{\mathsf{id} \in I^*} (x - \mathsf{id}) \\ G_{I^*}(x) &= F_{I^*}(x + \mathsf{id}^*) - F_{I^*}(\mathsf{id}^*). \end{aligned} \tag{C.5}$$

Observe that the constant term of G_{I^*} is $G_{I^*}(0) = F_{I^*}(\mathsf{id}^*) - F_{I^*}(\mathsf{id}^*) = 0$. Write $G_{I^*}(x) = \sum_{i \in [|I^*|]} \tilde{g}_i x^i$. Algorithm $\mathcal{B}$ computes the following:

$$\begin{aligned} [\mathsf{ct}_1]_1 &= [s]_1 \\ [\mathbf{ct}_2]_1 &= \sum_{i \in [|I^*|]} (\tilde{\mathbf{w}} \cdot \tilde{g}_i \cdot [s\hat{\tau}^i]_1 + \tilde{\mathbf{c}} \cdot \tilde{g}_i \cdot [abs\hat{\tau}^i]_1) = [s(\tilde{\mathbf{w}} + ab\tilde{\mathbf{c}}) \cdot G_{I^*}(\hat{\tau})]_1 \\ [\mathsf{ct}_3]_1 &= (\tilde{v} + \tilde{h} \cdot \mathsf{tg}^*) \cdot [s]_1 \\ [\mathsf{ct}_4]_T &= \tilde{\alpha} \cdot [s]_1 \cdot [1]_2 - [z]_T + [m_\beta]_T \end{aligned}$$

Algorithm $\mathcal{B}$ responds with the ciphertext $\mathsf{ct} = ([\mathsf{ct}_1]_1, [\mathbf{ct}_2]_1, [\mathsf{ct}_3]_1, [\mathsf{ct}_4]_T)$.

6. Algorithm $\mathcal{A}$ can continue to make key-computation queries. Algorithm $\mathcal{B}$ responds as described above.

7. At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $\beta' \in \{0, 1\}$. Algorithm $\mathcal{B}$ then checks that id^* is the minimum element in the set $I^* \setminus \bigcup_{j \in [K]} S_j$ and outputs 0 if not. Otherwise, algorithm $\mathcal{B}$ outputs β'.

We show that depending on the distribution of the challenge element z, algorithm $\mathcal{A}$ either perfectly simulates an execution of $\mathsf{Hyb}_6^{(\beta)}$ or $\mathsf{Hyb}_7^{(\beta)}$. We first consider the distribution of the public parameters. By construction, algorithm $\mathcal{B}$ constructs the public parameters by implicitly setting

$$\begin{aligned} \tau &= \hat{\tau} + \mathsf{id}^* & \alpha &= \tilde{\alpha} - ab & v &= \tilde{v} - b \cdot \mathsf{tg}^* \\ & & \mathbf{w} &= \tilde{\mathbf{w}} + ab\tilde{\mathbf{c}} & h &= \tilde{h} + b \end{aligned} \tag{C.6}$$

Since the challenger samples $\hat{\tau} \xleftarrow{R} \mathbb{Z}_p$ and algorithm $\mathcal{B}$ samples $\tilde{\alpha}, \tilde{v}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$ and $\tilde{\mathbf{w}} \xleftarrow{R} \mathbb{Z}_p^K$, the distribution of $\tau, v, h, \mathbf{w}, \alpha$ are all distributed according to the distribution in $\mathsf{Hyb}_6^{(\beta)}$ and $\mathsf{Hyb}_7^{(\beta)}$. Moreover, the public parameters perfectly hides $\tilde{\mathbf{c}}$. Next, consider the components of the challenge ciphertext. We claim that algorithm $\mathcal{B}$ generates the challenge ciphertext according to the specification of $\mathsf{Hyb}_6^{(\beta)}$ or $\mathsf{Hyb}_7^{(\beta)}$ where the encryption randomness $s \xleftarrow{R} \mathbb{Z}_p$ is the corresponding exponent sampled by the challenger. We consider each component separately:

- By construction, algorithm $\mathcal{B}$ sets $\mathsf{ct}_1 = s$ which matches the distribution in $\mathsf{Hyb}_6^{(\beta)}$ and $\mathsf{Hyb}_7^{(\beta)}$.
- Consider $\mathbf{ct}_2$. In the reduction, algorithm $\mathcal{B}$ implicitly sets $\tau = \hat{\tau} + \mathsf{id}^*$ and $\mathbf{w} = \tilde{\mathbf{w}} + ab\tilde{\mathbf{c}}$. Now, in $\mathsf{Hyb}_6^{(\beta)}$ and $\mathsf{Hyb}_7^{(\beta)}$, the challenger would set $\mathbf{ct}_2 = s\mathbf{w} \cdot F_{I^*}(\tau)$ where we can write

$$\begin{aligned} F_{I^*}(\tau) &= F_{I^*}(\mathsf{id}^*) + F_{I^*}(\tau) - F_{I^*}(\mathsf{id}^*) \\ &= F_{I^*}(\mathsf{id}^*) + F_{I^*}(\hat{\tau} + \mathsf{id}^*) - F_{I^*}(\mathsf{id}^*) \\ &= F_{I^*}(\mathsf{id}^*) + G_{I^*}(\hat{\tau}), \end{aligned}$$

where G_{I^*} is the polynomial from Eq. (C.5). If $\mathsf{id}^* \notin I^*$, then algorithm $\mathcal{B}$ outputs 0, exactly as in $\mathsf{Hyb}_6^{(\beta)}$ and $\mathsf{Hyb}_7^{(\beta)}$. Otherwise, if $\mathsf{id}^* \in I^*$, then $F_{I^*}(\mathsf{id}^*) = 0$ and so $F_{I^*}(\tau) = G_{I^*}(\hat{\tau})$. Substituting in algorithm $\mathcal{B}$'s choice

of $\mathbf{w}$ and the fact that $F_{I^*}(\tau) = G_{I^*}(\hat{\tau}) = \sum_{i \in [|I^*|]} \tilde{g}_i \hat{\tau}^i$, we have

$$\begin{aligned}\mathbf{ct}_2 &= s\mathbf{w} \cdot F_{I^*}(\tau) \\ &= s(\tilde{\mathbf{w}} + \tilde{\mathbf{c}}ab) \cdot G_{I^*}(\hat{\tau}) \\ &= \sum_{i \in [|I^*|]} (\tilde{\mathbf{w}} \cdot \tilde{g}_i \cdot s\hat{\tau}^i + \tilde{\mathbf{c}} \cdot \tilde{g}_i \cdot abs\hat{\tau}^i).\end{aligned}$$

This coincides with how $\mathcal{B}$ constructs $\mathbf{ct}_2$ in the reduction.

- Consider ct_3. In $\mathsf{Hyb}_6^{(\beta)}$ and $\mathsf{Hyb}_7^{(\beta)}$, the challenger would set $\mathrm{ct}_3 = s(v + h \cdot \mathrm{tg}^*)$. Substituting in algorithm $\mathcal{B}$'s choice of v and h, we have

$$\mathrm{ct}_3 = s(v + h \cdot \mathrm{tg}^*) = s(\tilde{v} - b \cdot \mathrm{tg}^*) + s(\tilde{h} + b) \cdot \mathrm{tg}^* = s(\tilde{v} + \tilde{h} \cdot \mathrm{tg}^*),$$

  which is precisely how algorithm $\mathcal{B}$ constructs ct_3 in the reduction.

- Finally, consider the distribution of ct_4. We consider two possibilities depending on the distribution of z:
  - Suppose $z = abs$. In the reduction, algorithm $\mathcal{B}$ implicitly sets $\alpha = \tilde{\alpha} - ab$ so

$$\mathrm{ct}_4 = \tilde{\alpha}s - z + m_\beta = \tilde{\alpha}s - abs + m_\beta = s\alpha + m_\beta,$$

    which is precisely the distribution of ct_4 in $\mathsf{Hyb}_6^{(\beta)}$.
  - Suppose $z \xleftarrow{R} \mathbb{Z}_p$. In this case, the distribution of ct_4 is uniform over $\mathbb{Z}_p$. This is the distribution of ct_4 in $\mathsf{Hyb}_7^{(\beta)}$.

We conclude that depending on the distribution of z, the challenge ciphertext in the reduction is distributed either according to the specification of $\mathsf{Hyb}_6^{(\beta)}$ or the specification of $\mathsf{Hyb}_7^{(\beta)}$. To complete the proof, it thus suffices to consider the key-computation queries. Suppose $\mathcal{A}$ makes a key-computation query on a set of identities $S \subseteq \mathbb{Z}_p$ and a tag $\mathrm{tg} \in \mathbb{Z}_p$. As in the reduction, we consider two cases:

- Suppose $\mathrm{tg} \neq \mathrm{tg}^*$. In this case, algorithm $\mathcal{B}$ samples $\tilde{\mathbf{y}} \xleftarrow{R} \mathbb{Z}_p^K$. Algorithm $\mathcal{B}$ then sets the key-generation randomness r to be

$$r = u_1 = \tilde{r} + (\mathrm{tg} - \mathrm{tg}^*)^{-1}(1 - \tilde{\mathbf{y}}^\top \tilde{\mathbf{c}} \cdot F_S(\mathrm{id}^*)) \cdot a,$$

  where $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$. Since $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$, the distribution of r coincides with the distribution in $\mathsf{Hyb}_6^{(\beta)}$ and $\mathsf{Hyb}_7^{(\beta)}$. Thus, it suffices to argue that the component u_2 is correctly constructed (with respect to algorithm $\mathcal{B}$'s choice of r and $\tilde{\mathbf{y}}$). In $\mathsf{Hyb}_6^{(\beta)}$ and $\mathsf{Hyb}_7^{(\beta)}$, the challenger would first compute the digest $\mathrm{dig} = [d]_2 = [F_S(\tau)]_2$ and then set

$$u_2 = \alpha + r(v + h \cdot \mathrm{tg}) + \tilde{\mathbf{y}}^\top \mathbf{w} \cdot F_S(\tau) = \alpha + u_1(v + h \cdot \mathrm{tg}) + \tilde{\mathbf{y}}^\top \mathbf{w} \cdot F_S(\tau). \tag{C.7}$$

  In the reduction, algorithm $\mathcal{B}$ implicitly sets $\tau = \hat{\tau} + \mathrm{id}^*$. Thus, we can write

$$\begin{aligned}F_S(\tau) &= F_S(\mathrm{id}^*) + F_S(\tau) - F_S(\mathrm{id}^*) \\ &= F_S(\mathrm{id}^*) + F_S(\hat{\tau} + \mathrm{id}^*) - F_S(\mathrm{id}^*) \\ &= F_S(\mathrm{id}^*) + G_S(\hat{\tau}),\end{aligned}$$

  where G_S is the polynomial from Eq. (C.4). By definition of the coefficients $\tilde{g}_i$ and using the fact that algorithm $\mathcal{B}$ implicitly defines $\mathbf{w} = \tilde{\mathbf{w}} + ab\tilde{\mathbf{c}}$, we can write

$$\begin{aligned}\tilde{\mathbf{y}}^\top \mathbf{w} \cdot F_S(\tau) &= \tilde{\mathbf{y}}^\top (\tilde{\mathbf{w}} + ab\tilde{\mathbf{c}}) \cdot (F_S(\mathrm{id}^*) + G_S(\hat{\tau})) \\ &= \tilde{\mathbf{y}}^\top \tilde{\mathbf{w}} \cdot F_S(\mathrm{id}^*) + \tilde{\mathbf{y}}^\top \tilde{\mathbf{c}} \cdot ab \cdot F_S(\mathrm{id}^*) + \sum_{i \in [|S|]} \tilde{\mathbf{y}}^\top \tilde{\mathbf{w}} \cdot (\tilde{g}_i \cdot \hat{\tau}^i) + \sum_{i \in [|S|]} \tilde{\mathbf{y}}^\top \tilde{\mathbf{c}} \cdot (\tilde{g}_i \cdot ab\hat{\tau}^i).\end{aligned}$$

In the reduction, algorithm $\mathcal{B}$ implicitly constructs u_2 as follows:

$$\begin{aligned}
u_2 &= \tilde{\alpha} + \tilde{r}b(\mathrm{tg} - \mathrm{tg}^*) + u_1(\tilde{v} + \tilde{h} \cdot \mathrm{tg}) + \sum_{i \in [|S|]} \tilde{\mathbf{y}}^\intercal \tilde{\mathbf{w}} \cdot \tilde{g}_i \cdot \hat{\tau}^i + \sum_{i \in [|S|]} \tilde{\mathbf{y}}^\intercal \tilde{\mathbf{c}} \cdot \tilde{g}_i \cdot ab\hat{\tau}^i + \tilde{\mathbf{y}}^\intercal \tilde{\mathbf{w}} \cdot F_S(\mathrm{id}^*) \\
&= \tilde{\alpha} + \tilde{r}b(\mathrm{tg} - \mathrm{tg}^*) + u_1(\tilde{v} + \tilde{h} \cdot \mathrm{tg}) + \tilde{\mathbf{y}}^\intercal \tilde{\mathbf{w}} \cdot G_S(\hat{\tau}) + \tilde{\mathbf{y}}^\intercal \tilde{\mathbf{c}} \cdot ab \cdot G_S(\hat{\tau}) + \tilde{\mathbf{y}}^\intercal \tilde{\mathbf{w}} \cdot F_S(\mathrm{id}^*) \\
&= \tilde{\alpha} + \tilde{r}b(\mathrm{tg} - \mathrm{tg}^*) + u_1(\tilde{v} + \tilde{h} \cdot \mathrm{tg}) + \tilde{\mathbf{y}}^\intercal \tilde{\mathbf{w}} \cdot F_S(\tau) + \tilde{\mathbf{y}}^\intercal \tilde{\mathbf{c}} \cdot ab \cdot G_S(\hat{\tau})
\end{aligned}$$

Using the relations $\alpha = \tilde{\alpha} - ab$ and $\mathbf{w} = \tilde{\mathbf{w}} + ab\tilde{\mathbf{c}}$ as well as the fact that $F_S(\tau) = F_S(\mathrm{id}^*) + G_S(\hat{\tau})$, we have that u_2 as computed by the reduction satisfies

$$\begin{aligned}
u_2 &= \tilde{\alpha} + \tilde{r}b(\mathrm{tg} - \mathrm{tg}^*) + u_1(\tilde{v} + \tilde{h} \cdot \mathrm{tg}) + \tilde{\mathbf{y}}^\intercal \tilde{\mathbf{w}} \cdot F_S(\tau) + \tilde{\mathbf{y}}^\intercal \tilde{\mathbf{c}} \cdot ab \cdot G_S(\hat{\tau}) \\
&= \tilde{\alpha} + \tilde{r}b(\mathrm{tg} - \mathrm{tg}^*) + u_1(\tilde{v} + \tilde{h} \cdot \mathrm{tg}) + \tilde{\mathbf{y}}^\intercal (\mathbf{w} - ab\tilde{\mathbf{c}}) \cdot F_S(\tau) + \tilde{\mathbf{y}}^\intercal \tilde{\mathbf{c}} \cdot ab \cdot G_S(\hat{\tau}) \\
&= \tilde{\alpha} + \tilde{r}b(\mathrm{tg} - \mathrm{tg}^*) + u_1(\tilde{v} + \tilde{h} \cdot \mathrm{tg}) + \tilde{\mathbf{y}}^\intercal \mathbf{w} \cdot F_S(\tau) + \tilde{\mathbf{y}}^\intercal \tilde{\mathbf{c}} \cdot ab \cdot (G_S(\hat{\tau}) - F_S(\tau)) \\
&= \alpha + \tilde{r}b(\mathrm{tg} - \mathrm{tg}^*) + u_1(\tilde{v} + \tilde{h} \cdot \mathrm{tg}) + \tilde{\mathbf{y}}^\intercal \mathbf{w} \cdot F_S(\tau) + ab(1 - \tilde{\mathbf{y}}^\intercal \tilde{\mathbf{c}} \cdot F_S(\mathrm{id}^*)).
\end{aligned} \tag{C.8}$$

Consider now the value of $u_1(v + h \cdot \mathrm{tg})$ from Eq. (C.7) instantiated with the reduction's setting of the variables:

$$\begin{aligned}
u_1(v + h \cdot \mathrm{tg}) &= u_1(\tilde{v} - b \cdot \mathrm{tg}^* + (\tilde{h} + b) \cdot \mathrm{tg}) \\
&= u_1(\tilde{v} + \tilde{h} \cdot \mathrm{tg}) + u_1 b(\mathrm{tg} - \mathrm{tg}^*) \\
&= u_1(\tilde{v} + \tilde{h} \cdot \mathrm{tg}) + (\tilde{r} + (\mathrm{tg} - \mathrm{tg}^*)^{-1}(1 - \tilde{\mathbf{y}}^\intercal \tilde{\mathbf{c}} \cdot F_S(\mathrm{id}^*)) \cdot a)b(\mathrm{tg} - \mathrm{tg}^*) \\
&= u_1(\tilde{v} + \tilde{h} \cdot \mathrm{tg}) + \tilde{r}b(\mathrm{tg} - \mathrm{tg}^*) + ab(1 - \tilde{\mathbf{y}}^\intercal \tilde{\mathbf{c}} \cdot F_S(\mathrm{id}^*)).
\end{aligned}$$

Replacing the terms in green in Eq. (C.8) with $u_1(v + h \cdot \mathrm{tg})$, we have

$$u_2 = \alpha + u_1(v + h \cdot \mathrm{tg}) + \tilde{\mathbf{y}}^\intercal \mathbf{w} \cdot F_S(\tau).$$

This is precisely the expression in Eq. (C.7) so we conclude that algorithm $\mathcal{B}$ answers the key-computation query exactly according to the specification of $\mathrm{Hyb}_6^{(\beta)}$ and $\mathrm{Hyb}_7^{(\beta)}$.

- Suppose $\mathrm{tg} = \mathrm{tg}^*$. In this case, algorithm $\mathcal{B}$ first samples $\tilde{\mathbf{y}} \xleftarrow{R} \mathbb{Z}_p^K$ subject to $\tilde{\mathbf{y}}^\intercal \tilde{\mathbf{c}} = 1/F_S(\mathrm{id}^*)$. If this is the K^{th} key-computation query on $\mathrm{tg} = \mathrm{tg}^*$ and $\tilde{y}_K = 0$, algorithm $\mathcal{B}$ outputs 0 exactly as in $\mathrm{Hyb}_6^{(\beta)}$ and $\mathrm{Hyb}_7^{(\beta)}$.

  Otherwise, algorithm $\mathcal{B}$ samples $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$ and sets $u_1 = \tilde{r}$. As in the previous case, it now suffices now to show that the component u_2 is correctly computed (with respect to algorithm $\mathcal{B}$'s choice of u_1 and $\tilde{\mathbf{y}}$). As in the previous case, in $\mathrm{Hyb}_6^{(\beta)}$ and $\mathrm{Hyb}_7^{(\beta)}$, the challenger would set

$$u_2 = \alpha + \tilde{r}(v + h \cdot \mathrm{tg}) + \tilde{\mathbf{y}}^\intercal \mathbf{w} \cdot F_S(\tau). \tag{C.9}$$

  Suppose now that we substitute the values of $\alpha, v, h, \mathbf{w}, \tilde{\mathbf{y}}$ from the reduction (see Eq. (C.6)) into Eq. (C.9). Using again the fact that $F_S(\tau) = F_S(\mathrm{id}^*) + G_S(\hat{\tau})$, we now have the following:

$$\begin{aligned}
u_2 &= \alpha + \tilde{r}(v + h \cdot \mathrm{tg}^*) + \tilde{\mathbf{y}}^\intercal \mathbf{w} \cdot F_S(\tau) \\
&= \tilde{\alpha} - ab + \tilde{r}(\tilde{v} - b \cdot \mathrm{tg}^* + (\tilde{h} + b) \cdot \mathrm{tg}^*) + \tilde{\mathbf{y}}^\intercal (\tilde{\mathbf{w}} + ab\tilde{\mathbf{c}})(F_S(\mathrm{id}^*) + G_S(\hat{\tau})) \\
&= \tilde{\alpha} + \tilde{r}(\tilde{v} + \tilde{h} \cdot \mathrm{tg}^*) + \sum_{i \in [|S|]} (\tilde{\mathbf{y}}^\intercal \tilde{\mathbf{w}} \cdot \tilde{g}_i \cdot \hat{\tau}^i) + \sum_{i \in [|S|]} (\tilde{\mathbf{y}}^\intercal \tilde{\mathbf{c}} \cdot \tilde{g}_i \cdot ab\hat{\tau}^i) + \tilde{\mathbf{y}}^\intercal \tilde{\mathbf{w}} \cdot F_S(\mathrm{id}^*) - ab(1 - \tilde{\mathbf{y}}^\intercal \tilde{\mathbf{c}} \cdot F_S(\mathrm{id}^*)) \\
&= \tilde{\alpha} + \tilde{r}(\tilde{v} + \tilde{h} \cdot \mathrm{tg}^*) + \sum_{i \in [|S|]} (\tilde{\mathbf{y}}^\intercal \tilde{\mathbf{w}} \cdot \tilde{g}_i \cdot \hat{\tau}^i) + \sum_{i \in [|S|]} (\tilde{\mathbf{y}}^\intercal \tilde{\mathbf{c}} \cdot \tilde{g}_i \cdot ab\hat{\tau}^i) + \tilde{\mathbf{y}}^\intercal \tilde{\mathbf{w}} \cdot F_S(\mathrm{id}^*)
\end{aligned}$$

  where the final cancellation relies on the fact that the reduction chose $\tilde{\mathbf{y}}$ such that $\tilde{\mathbf{y}}^\intercal \tilde{\mathbf{c}} = 1/F_S(\mathrm{id}^*)$. This precisely coincides with how algorithm $\mathcal{B}$ constructs u_2. We conclude that algorithm $\mathcal{B}$ answers the key-computation query according to the specification of $\mathrm{Hyb}_6^{(\beta)}$ and $\mathrm{Hyb}_7^{(\beta)}$.

We conclude that algorithm $\mathcal{B}$ responds to the key-generation queries with the same procedure as in $\mathsf{Hyb}_6^{(\beta)}$ and $\mathsf{Hyb}_7^{(\beta)}$. Thus, as argued above, if $z = abs$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_6^{(\beta)}$, whereas if $z \xleftarrow{R} \mathbb{Z}_p$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_7^{(\beta)}$. Thus, algorithm $\mathcal{B}$ breaks Assumption C.5 with the same advantage ε. The lemma follows. □

Lemma C.14. *It holds that* $\Pr\left[\mathsf{Hyb}_7^{(0)}(\mathcal{A}) = 1\right] = \Pr\left[\mathsf{Hyb}_7^{(1)}(\mathcal{A}) = 1\right]$.

Proof. These experiments are identical by construction (i.e., the view of the adversary in hybrid $\mathsf{Hyb}_3^{(\beta)}$ is independent of the bit $\beta \in \{0, 1\}$). □

Since $M = \mathsf{poly}(\lambda)$, Theorem C.6 now follows by combining Lemmas C.7 to C.14 and a standard hybrid argument. □

Remark C.15 (From Tag-Based ABE for Subsets to Batched IBE)**.** It is straightforward to derive a batched IBE scheme from any tag-based ABE scheme for subset policies. Let $\mathcal{I}$ be the identity space for the batched IBE scheme and $\mathcal{I}'$ be the identity space for the tag-based ABE scheme for subset policies. All we need is a way to map an identity $\mathsf{id} \in \mathcal{I}$ onto a set of identities $I \subset \mathcal{I}'$. We denote this mapping by $H\colon \mathcal{I} \to 2^{\mathcal{I}'}$, where $2^{\mathcal{I}'}$ denotes the power set of $\mathcal{I}'$ (i.e., the set of all subsets of $\mathcal{I}'$). Then, we proceed as follows:

- **Public parameters:** The public parameters for the batched IBE scheme are the public parameters for the tag-based ABE scheme. The message space and the batch label space of the batched IBE scheme correspond to the message space $\mathcal{M}$ and tag space $\mathcal{T}$ of the tag-based ABE scheme, respectively.
- **Encryption:** A batched IBE encryption of a message m to an identity $\mathsf{id} \in \mathcal{I}$ and tag $\mathsf{tg} \in \mathcal{T}$ is a tag-based ABE encryption of the message with respect to the set $H(\mathsf{id})$ and the tag tg.
- **Key-generation:** The secret key for a set of identities $S \subseteq \mathcal{I}$ and batch label tg for the batched IBE scheme is a secret key for the set $\bigcup_{\mathsf{id}\in S} H(\mathsf{id})$ and tag tg.

Correctness for the batched IBE scheme now follows immediately from correctness of the tag-based ABE scheme.

Moreover, if the mapping H is *B-cover-free*, where B is the batch size, then security of the batched IBE scheme follows immediately from security of the tag-based ABE scheme. More precisely, we say that H is B-cover free [KS64] if for all $\mathsf{id}^* \in \mathcal{I}$ and all sets $S \subseteq \mathcal{I}$ of size at most B where $\mathsf{id}^* \notin S$, we have that $H(\mathsf{id}^*) \not\subseteq \bigcup_{\mathsf{id}\in S} H(\mathsf{id})$. Recall that in the batched IBE security game, the adversary is not allowed to query for a key on the challenge batch label tg^* and a set S (of size at most B) that contains the challenge identity id^*. If the hash function H is B-cover-free, this means $H(\mathsf{id}^*) \not\subseteq \bigcup_{\mathsf{id}\in S} H(\mathsf{id})$. This is precisely our admissibility requirement for a tag-based ABE scheme for subset functionalities (see Definition C.1). Moreover, if the underlying tag-based ABE scheme allows the adversary to adaptively choose the challenge set, then that corresponds to a batched IBE scheme where the adversary can adaptively choose the challenge identity.

There are many standard instantiations of cover-free set systems. For instance, using the formalization from [LWW25, Fact 6.2], the work of [EFF85] give a cover-free set system with the following parameters:

- Let q be any prime power and take any integer $t < q$. Let $K = \lfloor q/(t-1) \rfloor$. Then there is an explicit and efficiently-computable K-cover-free hash function $H\colon [q^t] \to 2^X$ where $|X| = q^2$.

Thus, if we want a batched IBE scheme with an exponential-size identity space $|\mathcal{I}| \geq 2^\lambda$ and batch size B, then we can instantiate the cover-free hash family with the smallest prime power q where $q \geq (\lambda + 1) \cdot B$ and $t = \lambda$. Then, the [EFF85] construction gives a B-cover-free hash function with domain $\mathcal{I} = [q^t]$ and range 2^X where $\mathcal{I}' = X$ is a set of size $|\mathcal{I}'| = |X| = q^2 = O(\lambda^2 B^2)$.

The key observation is that to build a batched IBE scheme with an exponential-size identity space $\mathcal{I}$, we only need a tag-based ABE scheme for subset policies with a $O(\lambda^2 B^2)$-size identity space $\mathcal{I}'$. We can now instantiate this tag-based ABE scheme for subset functionalities using Construction C.3. As we showed in Theorem C.6, Construction C.3 is secure against adversaries that can adaptively choose the challenge set, and thus, we obtain a batched IBE scheme where the adversary can adaptively choose the challenge identity. Security relies on the same q-type assumption (Assumption C.5).

Finally, the use of cover-free sets only affects the size of the master public key. It does not affect the size of the ciphertext or the secret key since neither of these depends on the maximum batch size or the size of the identity space. The size of the public key in Construction C.3 contains $O(KB')$ group elements, where B' is a bound on the size of the maximum set associated with a decryption key and K is the collusion bound. Using the above cover-free hash function, we can bound $B' \leq B \cdot |X| = O(\lambda^2 B^3)$. This means the resulting batched IBE scheme will have size $O(\lambda^2 K B^3)$. While this is significantly worse than the selectively-secure scheme (Construction 4.2) or the adaptively-secure scheme in the generic group model (Construction D.1), we believe this construction is still interesting in that it demonstrates the fact that adaptive security is achievable in the *plain* model.

Remark C.16 (Supporting an Adaptive Choice of Batch Label)**.** Remark C.15 (and Construction C.3) shows how we can construct a batched IBE scheme (or tag-based ABE scheme for subset functionalities) that allows an adversary to adaptively choose the challenge identity (or challenge set). However, these schemes are still selectively-secure in the choice of batch label (or tag). This is because we embed the batch label or the tag using the Boneh-Boyen mechanism. Specifically, recall from Section 2 that we can view our batched IBE constructions (and its generalization to tag-based ABE for subset functionalities) as a composition of a one-key scheme (without batch labels or tags) together with a pairing-based IBE scheme (where we treat the batch label or tag as the identity). Our limitation to selective security in the choice of the batch label or tag is due to the fact that the Boneh-Boyen IBE scheme is only selectively secure in the choice of the identity. A natural approach then to achieve adaptive security is to substitute the Boneh-Boyen IBE scheme with the Waters' IBE [Wat05] scheme which is adaptively secure. In conjunction with Construction C.3 and Remark C.15, we believe this yields a plausible route to an batched IBE scheme in the plain model that satisfies full adaptive security.

Generic hardness of Assumption C.5. As in Appendix B, we can use Theorem B.2 to prove that Assumption C.5 holds in the generic asymmetric bilinear group model.

Theorem C.17 (Generic Hardness of Assumption C.5)**.** *Let $N \in \mathbb{N}$ and let $\mathcal{A}$ be any adversary for Assumption C.5 with parameter N. If $\mathcal{A}$ makes at most q generic group oracle queries, then the advantage of $\mathcal{A}$ is at most $O(q^2 N^3)/p$ in the generic asymmetric bilinear group model. In particular, whenever $p > \lambda^{\omega(1)}$, the advantage of $\mathcal{A}$ is negligible for all polynomials $N, q = \mathsf{poly}(\lambda)$.*

Proof. We start by defining the sets of polynomials $\mathcal{P}, \mathcal{Q}$ and the challenge polynomial T associated with the challenge terms in Assumption C.5. By construction, each polynomial is over the formal variables a, b, s, τ. Then, the polynomials are defined as follows:

$$\mathcal{P} = \{1, b, s, ab, \{\tau^i, ab\tau^i, s\tau^i, abs\tau^i\}_{i\in[N]}\}$$
$$\mathcal{Q} = \{1, a, b, \{\tau^i, ab\tau^i\}_{i\in[N]}\}$$
$$T = abs.$$

To appeal to Theorem B.2, we need to show that T is independent with respect to the product $\mathcal{PQ}$. Since $\mathcal{P}$ and $\mathcal{Q}$ consist of monomials and T is also a monomial, it suffices to argue that $T \notin \mathcal{PQ}$. This follows by inspection. Namely, suppose we write $T = PQ$ where $P \in \mathcal{P}$. Then, the following holds:

- If $P \in \{1, b, s, ab\}$ and $PQ = T = abs$, then $Q \in \{abs, as, ab, s\}$. By definition of $\mathcal{Q}$, this means $Q \notin \mathcal{Q}$.
- If $P \in \{\tau^i, ab\tau^i, s\tau^i, abs\tau^i\}_{i\in[N]}$ and $PQ = T = abs$, then Q must be a multiple of τ^{-i}, for some $i \in [N]$. This means $Q \notin \mathcal{Q}$.

We conclude that T is independent with respect to $\mathcal{PQ}$. Finally, to invoke Theorem B.2, we compute the maximum degree d among polynomials in $\mathcal{PQ} \cup \{T\}$. By inspection, the maximum degree $d_{\mathcal{P}}$ of polynomials in $\mathcal{P}$ is $d_{\mathcal{P}} = O(N)$ and the maximum degree $d_{\mathcal{Q}}$ of polynomials in $\mathcal{Q}$ is $d_{\mathcal{Q}} = O(N)$. The degree of T is 3. Thus $d = O(N)$. Finally, $|\mathcal{P}| = O(N)$ and $|\mathcal{Q}| = O(N)$. The claim now follows by Theorem B.2. □

D Adaptively-Secure Batched IBE in the Generic Group Model

In this section, we show a variant of Construction 4.2 where we integrate our basic approach with ideas from the Boneh-Franklin [BF01] IBE scheme. As discussed in Section 2, we work in the generic bilinear group model (and the random oracle model) and prove adaptive security of our resulting construction. Compared to the previous scheme of [AFP25], we save one group element in our ciphertext, one exponentiation during encryption, and one pairing operation during decryption (see Tables 1 and 2).

Construction D.1 (Batched Identity-Based Encryption)**.** Let GroupGen be a prime-order bilinear group generator. We construct a batched IBE scheme $\Pi_{\mathsf{BatchIBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Digest}, \mathsf{ComputeKey}, \mathsf{Decrypt})$ as follows:

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter λ, the setup algorithm samples $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$. Next, let $\mathsf{H} \colon \{0,1\}^\lambda \to \mathbb{G}_2$ be a hash function (which we model as a random oracle in the security analysis). The setup algorithm outputs the public parameters $\mathsf{pp} = (\mathcal{G}, \mathsf{H})$. The message space associated with pp is $\mathbb{G}_\mathsf{T}$, the identity space is $\{0,1\}^*$, and the tag space is $\{0,1\}^\lambda$.

- $\mathsf{KeyGen}(\mathsf{pp}, 1^B)$: On input the public parameters $\mathsf{pp} = \mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e)$ and a bound on the batch size B, the key-generation algorithm samples exponents $\tau, w, \alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and outputs the master public key

$$\mathsf{mpk} = \left(\mathcal{G}, \mathsf{H}, [\tau]_1, [\tau]_2, [\tau^2]_2, \ldots, [\tau^B]_2, [w]_1, [w\tau]_1, [\alpha]_1\right) \tag{D.1}$$

  and the master secret key $\mathsf{msk} = (w, \alpha)$.

- $\mathsf{Encrypt}(\mathsf{mpk}, [m]_\mathsf{T}, \mathsf{id}, \mathsf{tg})$: On input the master public key mpk (parsed according to Eq. (D.1)), a message $[m]_\mathsf{T} \in \mathbb{G}_\mathsf{T}$, an identity $\mathsf{id} \in \mathbb{Z}_p$, and a batch label $\mathsf{tg} \in \{0,1\}^\lambda$, the encryption algorithm computes $[h_\mathsf{tg}]_2 = \mathsf{H}(\mathsf{tg})$ and samples $s \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. It then outputs the ciphertext

$$\begin{aligned}\mathsf{ct} &= \left([s]_1, s \cdot [w\tau]_1 - s \cdot \mathsf{id} \cdot [w]_1, s \cdot [\alpha]_1 \cdot [h_\mathsf{tg}]_2 + [m]_\mathsf{T}\right) \\ &= \left([s]_1, [sw(\tau - \mathsf{id})]_1, [s\alpha h_\mathsf{tg} + m]_\mathsf{T}\right)\end{aligned}$$

- $\mathsf{Digest}(\mathsf{mpk}, S)$: On input the master public key mpk (parsed according to Eq. (D.1)) and a set of identities $S \subseteq \mathbb{Z}_p$ where $|S| \le B$, the digest algorithm defines the polynomial $F_S(x) = \prod_{\mathsf{id} \in S}(x - \mathsf{id})$ whose roots are the identities $\mathsf{id} \in S$. Write $F(x) = \sum_{i \in [0,|S|]} f_i x^i$. Output the digest

$$\mathsf{dig} = \sum_{i \in [0,|S|]} f_i \cdot [\tau^i]_2 = [F_S(\tau)]_2.$$

- $\mathsf{ComputeKey}(\mathsf{msk}, \mathsf{dig}, \mathsf{tg})$: On input the master secret key $\mathsf{msk} = (w, \alpha)$, a digest $\mathsf{dig} = [d]_2$, and a batch label $\mathsf{tg} \in \{0,1\}^\lambda$, the key-computation algorithm computes $h_\mathsf{tg} = \mathsf{H}(\mathsf{tg})$ and outputs the secret key

$$\mathsf{sk} = \alpha \cdot [h_\mathsf{tg}]_2 + w \cdot [d]_2 = [\alpha h_\mathsf{tg} + wd]_2.$$

- $\mathsf{Decrypt}(\mathsf{mpk}, \mathsf{sk}, S, (\mathsf{id}, \mathsf{tg}), \mathsf{ct})$: On input the master public key mpk (parsed according to Eq. (D.1)), a secret key $\mathsf{sk} = [u]_2$, the set of identities $S \subseteq \mathbb{Z}_p$, a target identity $\mathsf{id} \in S$, a batch label $\mathsf{tg} \in \{0,1\}^\lambda$, and the ciphertext $\mathsf{ct} = ([\mathsf{ct}_1]_1, [\mathsf{ct}_2]_1, [\mathsf{ct}_3]_\mathsf{T})$, the decryption algorithm proceeds as follows:
  - First, it defines the polynomial
$$F_{S\setminus\{\mathsf{id}\}}(x) = \prod_{\mathsf{id}' \in S \setminus \{\mathsf{id}\}} (x - \mathsf{id}').$$
    Compute $[F_{S\setminus\{\mathsf{id}\}}(\tau)]_2 = \sum_{i \in [0,|S|-1]} f_i [\tau^i]_2$, where $F_{S\setminus\{\mathsf{id}\}}(x) = \sum_{i \in [0,|S|-1]} f_i x^i$.
  - Then it computes and outputs
$$[\mathsf{ct}_3]_\mathsf{T} - [\mathsf{ct}_1]_1 \cdot [u]_2 + [\mathsf{ct}_2]_1 \cdot [F_{S\setminus\{\mathsf{id}\}}(\tau)]_2. \tag{D.2}$$

Theorem D.2 (Correctness)**.** *Construction D.1 is correct.*

Proof. Take any $\lambda, B \in \mathbb{N}$ and any $(\mathcal{G}, \mathsf{H})$ in the support of $\mathsf{Setup}(1^\lambda)$, where $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e)$. Take any $[m]_\mathsf{T} \in \mathbb{G}_\mathsf{T}$, any $\mathsf{id}^* \in \mathbb{Z}_p$, batch label $\mathsf{tg}^* \in \{0,1\}^\lambda$, and set $S \subseteq \mathbb{Z}_p$ of size at most B where $\mathsf{id}^* \in S$. Sample $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, 1^B)$ and $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, m, \mathsf{id}^*, \mathsf{tg}^*)$. Compute

$$\begin{aligned}\mathsf{dig} &= \mathsf{Digest}(\mathsf{mpk}, S)\\ \mathsf{sk} &\leftarrow \mathsf{ComputeKey}(\mathsf{msk}, \mathsf{dig}, \mathsf{tg}^*).\end{aligned}$$

Let $[h_{\mathsf{tg}^*}]_2 = \mathsf{H}(\mathsf{tg}^*)$. By construction, this means

$$\begin{aligned}\mathsf{mpk} &= \left(\mathcal{G}, \mathsf{H}, [\tau]_1, [\tau]_2, [\tau^2]_2, \ldots, [\tau^B]_2, [w]_1, [w\tau]_1, [\alpha]_1\right)\\ \mathsf{ct} &= \left([s]_1, [sw(\tau - \mathsf{id}^*)]_1, [s\alpha h_{\mathsf{tg}^*} + m]_\mathsf{T}\right)\\ \mathsf{sk} &= [\alpha h_{\mathsf{tg}^*} + wF_S(\tau)]_2\end{aligned}$$

where $F_S(x) = \prod_{\mathsf{id}\in S}(x - \mathsf{id})$. Consider now $\mathsf{Decrypt}(\mathsf{mpk}, \mathsf{sk}, S, (\mathsf{id}^*, \mathsf{tg}^*), \mathsf{ct})$. If we write $\mathsf{sk} = [u]_2$ and $\mathsf{ct} = ([\mathsf{ct}_1]_1, [\mathsf{ct}_2]_1, [\mathsf{ct}_3]_\mathsf{T})$, then the decryption algorithm computes

$$\begin{aligned}\mathsf{ct}_1 \cdot u &= s\alpha h_{\mathsf{tg}^*} + swF_S(\tau)\\ \mathsf{ct}_2 \cdot F_{S\setminus\{\mathsf{id}^*\}}(\tau) &= sw(\tau - \mathsf{id}^*) \cdot \prod_{\mathsf{id}\in S\setminus\{\mathsf{id}^*\}} (\tau - \mathsf{id}) = sw \cdot \prod_{\mathsf{id}\in S}(\tau - \mathsf{id}) = swF_S(\tau).\end{aligned}$$

This means

$$\mathsf{ct}_1 u - \mathsf{ct}_2 \cdot F_{S\setminus\{\mathsf{id}^*\}}(\tau) = (s\alpha h_{\mathsf{tg}^*} + swF_S(\tau)) - swF_S(\tau) = s\alpha h_{\mathsf{tg}^*}.$$

The decryption relation (Eq. (D.2)) now yields:

$$[\mathsf{ct}_3 - (\mathsf{ct}_1 u - \mathsf{ct}_2 \cdot F_{S\setminus\{\mathsf{id}^*\}}(\tau))]_\mathsf{T} = [s\alpha h_{\mathsf{tg}^*} + m - s\alpha h_{\mathsf{tg}^*}]_\mathsf{T} = [m]_\mathsf{T}$$

and correctness holds. □

Theorem D.3 (Adaptive Security)**.** *Take any polynomial $B = B(\lambda)$. Then, Construction D.1 is adaptively secure with batch size B if we model* GroupGen *as a generic group and the hash function* H *as a random oracle.*

Proof. Take any polynomial $B = B(\lambda)$ and let $\mathcal{A}$ be an adversary for the adaptive security experiment. Let $Q = Q(\lambda)$ be a bound on the number of oracle queries (to the generic group oracle and the random oracle) that algorithm $\mathcal{A}$ makes. We assume here that Q is polynomially-bounded, but otherwise, impose no further restrictions on the running time of the adversary (as is typical when analyzing security in idealized models). To prove adaptive security, we define two experiments, each indexed by a bit $\beta \in \{0,1\}$:

- $\mathsf{Hyb}_0^{(\beta)}$: This is the adaptive batched IBE security experiment with bit $\beta \in \{0,1\}$ in the generic group model and random oracle model. Let $n \geq \log p$. The challenger maintains random injective functions $\varphi_1, \varphi_2, \varphi_\mathsf{T} : \mathbb{Z}_p \to \mathcal{L}$ and a hash function map $\varphi_\mathsf{H} : \{0,1\}^\lambda \to \mathbb{Z}_p$. The adversary has access to the group operation oracle and the pairing oracle as described in Appendix B. The challenger implements oracle queries to H as follows:
  - **Hash oracle:** On input a batch label $r \in \{0,1\}^\lambda$, the hash oracle returns $\varphi_2(\varphi_\mathsf{H}(r))$.

  The experiment proceeds as follows:

  - The challenger sets $\mathsf{pp} = (\varphi_1(1), \varphi_2(1), \varphi_\mathsf{T}(1))$. The challenger samples exponents $\tau, w, \alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_p^*$ and constructs the master public key as follows

    $$\mathsf{mpk} = (\varphi_1(\tau), \varphi_2(\tau), \varphi_2(\tau^2), \ldots, \varphi_2(\tau^B), \varphi_1(w), \varphi_1(\alpha), \varphi_1(w\tau)).$$

    The challenger gives pp and mpk to $\mathcal{A}$.

– When algorithm $\mathcal{A}$ makes a key-computation query on a set of identities $S \subseteq \mathbb{Z}_p$ where $|S| \leq B$ and a batch label $\mathsf{tg} \in \{0,1\}^\lambda$, the challenger returns $\perp$ if $\mathcal{A}$ has made a query on the same tg. Otherwise, it defines the polynomial $F_S(x) = \prod_{\mathsf{id} \in S}(x - \mathsf{id})$ whose roots are the identities $\mathsf{id} \in S$ and write $F(x) = \sum_{i \in [0,|S|]} f_i x^i$. It then returns the secret key

$$\mathsf{sk} = \varphi_2(\alpha \varphi_{\mathsf{H}}(\mathsf{tg}) + w F_S(\tau)).$$

– When algorithm $\mathcal{A}$ outputs a challenge identity $\mathsf{id}^* \in \mathbb{Z}_p$, batch label $\mathsf{tg}^* \in \{0,1\}^\lambda$, and two messages m_0, m_1 in the image of φ_{T}, the challenger returns $\perp$ if $\mathcal{A}$ made a key-computation query $(\mathsf{tg}, S) \in Q$ such that $\mathsf{tg} = \mathsf{tg}^*$ and $\mathsf{id}^* \in S$. Otherwise, the challenger samples $s \xleftarrow{\mathrm{R}} \mathbb{Z}_p^*$. It then returns the challenge ciphertext

$$\mathsf{ct} = (\varphi_1(s), \varphi_1(sw(\tau - \mathsf{id}^*)), \varphi_{\mathsf{T}}(s\alpha\varphi_{\mathsf{H}}(\mathsf{tg}^*) + \varphi_{\mathsf{T}}^{-1}(m_\beta))).$$

– Algorithm $\mathcal{A}$ can continue to make key-computation queries. The challenger responds as described above except that it returns $\perp$ if $\mathsf{tg} = \mathsf{tg}^*$ and $\mathsf{id}^* \in S$.

– At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $\beta' \in \{0,1\}$, which the challenger also outputs.

- $\mathsf{Hyb}_1^{(\beta)}$: This is the symbolic version of the adaptive batched IBE security experiment with bit β, which is the same as $\mathsf{Hyb}_0^{(\beta)}$ except that the domain of $\varphi_1, \varphi_2, \varphi_{\mathsf{T}}$ and the range of φ_{H} are now treated as formal polynomials over $\mathbb{Z}_p$ (with variables defined implicitly below). In the following, we write $\mathbb{Z}_p[\star]$ to denote the set of formal polynomials over $\mathbb{Z}_p$. The challenger in this experiment lazily initializes the labeling functions $\varphi_i \colon \mathbb{Z}_p[\star] \to \mathcal{L}$ for each $i \in \{1, 2, \mathsf{T}\}$ and the hash function map $\varphi_{\mathsf{H}} \colon \{0,1\}^\lambda \to \mathbb{Z}_p[\star]$. Namely, whenever the challenger needs to compute $\varphi_i(f)$, where $f \in \mathbb{Z}_p[\star]$ and $\varphi_i(f)$ has not yet been defined, then the challenger samples a random label $\ell \xleftarrow{\mathrm{R}} \mathcal{L}$ (that is not already in the image of φ_i) and associates the label ℓ with $\varphi_i(f)$. If no such label exists, then the challenger halts the experiment with output $\perp$. With this procedure, the evaluation oracle and pairing oracle behave exactly as before. Next, to implement the hash oracle, the challenger now proceeds as follows:

  – **Hash oracle:** On input $r \in \{0,1\}^\lambda$, if $\varphi_{\mathsf{H}}(r)$ has not yet been initialized, then set $\varphi_{\mathsf{H}}(r) = R$ where R is a fresh formal variable. Return $\varphi_2(\varphi_{\mathsf{H}}(r))$.

  The experiment proceeds as follows:

  – The challenger sets $\mathsf{pp} = (\varphi_1(1), \varphi_2(1), \varphi_{\mathsf{T}}(1))$. The challenger defines formal variables X_τ, X_w, X_α corresponding to τ, w, α and constructs the master public key as follows

  $$\mathsf{mpk} = (\varphi_1(X_\tau), \varphi_2(X_\tau), \varphi_2(X_\tau^2), \ldots, \varphi_2(X_\tau^B), \varphi_1(X_w), \varphi_1(X_\alpha), \varphi_1(X_w X_\tau)).$$

  The challenger gives pp and mpk to $\mathcal{A}$.

  – When algorithm $\mathcal{A}$ makes a key-computation query on a set of identities $S \subseteq \mathbb{Z}_p$ where $|S| \leq B$ and a batch label $\mathsf{tg} \in \{0,1\}^\lambda$, the challenger returns $\perp$ if $\mathcal{A}$ has made a query on the same tg. Otherwise, it defines the polynomial $F_S(x) = \prod_{\mathsf{id} \in S}(x - \mathsf{id})$ whose roots are the identities $\mathsf{id} \in S$ and writes $F(x) = \sum_{i \in [0,|S|]} f_i x^i$. It then makes a hash oracle query on tg so that $\varphi_{\mathsf{H}}(\mathsf{tg})$ is defined internally and returns the secret key

  $$\mathsf{sk} = \varphi_2(X_\alpha \varphi_{\mathsf{H}}(\mathsf{tg}) + X_w F_S(X_\tau)).$$

  – When algorithms $\mathcal{A}$ outputs a challenge identity $\mathsf{id}^* \in \mathbb{Z}_p$, batch label $\mathsf{tg}^* \in \{0,1\}^\lambda$, and two messages m_0, m_1 in the image of φ_{T}, the challenger returns $\perp$ if $\mathcal{A}$ made a key-computation query $(\mathsf{tg}, S) \in Q$ such that $\mathsf{tg} = \mathsf{tg}^*$ and $\mathsf{id}^* \in S$. Otherwise, the challenger defines a formal variable X_s. It then makes a hashing oracle query on tg so that the value $\varphi_{\mathsf{H}}(\mathsf{tg})$ is defined. Then, it returns the challenge ciphertext

  $$\mathsf{ct} = (\varphi_1(X_s), \varphi_1(X_s X_w(X_\tau - \mathsf{id}^*)), \varphi_{\mathsf{T}}(X_s X_\alpha \varphi_{\mathsf{H}}(\mathsf{tg}^*) + \varphi_{\mathsf{T}}^{-1}(m_\beta))).$$

  – Algorithm $\mathcal{A}$ can continue to make key-computation queries. The challenger responds as described above except that it returns $\perp$ if $\mathsf{tg} = \mathsf{tg}^*$ and $\mathsf{id}^* \in S$.

– At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $\beta' \in \{0,1\}$, which the challenger also outputs.

We write $\mathsf{Hyb}_i^{(\beta)}(\mathcal{A})$ to denote the output distribution of $\mathsf{Hyb}_i^{(\beta)}$ with adversary $\mathcal{A}$. We now analyze each pair of adjacent hybrid experiments.

Lemma D.4. *For all $\beta \in \{0,1\}$, we have*

$$\left|\Pr[\mathsf{Hyb}_0^{(\beta)} = 1] - \Pr[\mathsf{Hyb}_1^{(\beta)} = 1\right| \leq \binom{Q+B+7}{2}\frac{B+1}{p} = \frac{\mathsf{poly}(Q,B)}{p}.$$

Proof. Consider an extended version of $\mathsf{Hyb}_1^{(\beta)}$ where at the end of the experiment, we sample uniform random values from $\mathbb{Z}_p$ for all formal variables, and replace all formal polynomials with their evaluations on these values. This yields a perfect simulation of $\mathsf{Hyb}_0^{(\beta)}$ unless there exist two distinct polynomials which share the same evaluation. We use Bad to denote this event. Then,

$$\left|\Pr[\mathsf{Hyb}_0^{(\beta)} = 1] - \Pr[\mathsf{Hyb}_1^{(\beta)} = 1]\right| \leq \Pr[\mathsf{Bad}].$$

We bound the probability of Bad by the following observations:

- There are $B+4$ polynomials from pp and mpk, 3 polynomials from ct, and (at most) Q polynomials from the adversary's queries. All $Q+B+7$ polynomials have degree at most $B+1$.
- By the Schwartz-Zippel lemma [Sch80, Zip79], the probability that a pair of these polynomials share the same evaluation (at a randomly-selected evaluation point) is at most $\frac{B+1}{p}$.

Taking now a union bound over the adversary's queries, we have

$$\Pr[\mathsf{Bad}] \leq \binom{Q+B+7}{2}\frac{B+1}{p}.$$

This proves the lemma. □

Lemma D.5. *It holds that* $\Pr[\mathsf{Hyb}_1^{(0)} = 1] = \Pr[\mathsf{Hyb}_1^{(1)} = 1]$.

Proof. It suffices to prove that the target polynomial $X_s \cdot X_\alpha \cdot \varphi_{\mathsf{H}}(\mathsf{tg}^*)$ in $\mathsf{Hyb}_1^{(\beta)}$ is linearly independent of all polynomials in the view of the adversary. To do so, we define the following quantities:

- Let $\mathsf{id}^* \in \mathbb{Z}_p$ and $\mathsf{tg}^* \in \{0,1\}^\lambda$ be the target identity and target batch label, respective, chosen by $\mathcal{A}$.
- Suppose algorithm $\mathcal{A}$ makes a total of n key-computation queries with batch labels $\mathsf{tg}_1, \ldots, \mathsf{tg}_n \in \{0,1\}^\lambda$. Let $S_1, \ldots, S_n \subseteq \mathbb{Z}_p$ be the identities associated with these queries. Let $F_1, \ldots, F_n$ be the polynomials (of degree at most B) associated with these sets. Furthermore, we require that all of the batch labels tg_i are distinct and that $\mathsf{tg}^* = \mathsf{tg}_{i^*}$ for some $i^* \in [n]$. The last condition corresponds to the assumption that the adversary always makes a key-computation query on the challenge batch label tg^*; this is without loss of generality as we can always have the adversary $\mathcal{A}$ make a dummy key-computation query on the challenge batch label.
- Finally, we assume that algorithm $\mathcal{A}$ queries the hash oracle on all of the batch labels $\mathsf{tg}_1, \ldots, \mathsf{tg}_n$. This is without loss of generality since any adversary that does not do this can be generically converted into one that does with only n additional queries and no change to the advantage. We write $H_i = \varphi_{\mathsf{H}}(\mathsf{tg}_i)$ for all $i \in [n]$.

The view of adversary then consists of the following two sets of polynomials over $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively:

$$L_1 = \overbrace{\{1, X_\tau, X_w, X_\alpha, X_w X_\tau\}}^{L_{1,1}} \cup \overbrace{\{X_s\}}^{L_{1,2}} \cup \overbrace{\{X_s X_w (X_\tau - \mathsf{id}^*)\}}^{L_{1,3}}$$

$$L_2 = \underbrace{\{H_i\}_{i\in[n]}}_{L_{2,1}} \cup \underbrace{\{X_\tau^i\}_{i\in[0,B]}}_{L_{2,2}} \cup \underbrace{\{X_\alpha H_i + X_w F_i(X_\tau)\}_{i\in[n]}}_{L_{2,3}}$$

In the following, we write L_1L_2 to denote the set of product polynomials $\{fg : f \in L_1, g \in L_2\}$. By contradiction, we assume that the target polynomial satisfies $X_sX_\alpha H_{i^*} \in \mathsf{span}(L_1L_2)$. Namely, suppose that $X_sX_\alpha H_{i^*}$ can be expressed as a non-trivial linear combination of terms in

$$L_1L_2 = L_{1,1}L_2 \cup \big((L_{1,2}L_{2,1}) \cup (L_{1,3}L_{2,1})\big) \cup \big((L_{1,2}L_{2,2}) \cup (L_{1,3}L_{2,3})\big) \cup \big((L_{1,2}L_{2,3}) \cup (L_{1,3}L_{2,2})\big)$$

Then the following properties hold:

- The coefficients of the polynomials in $L_{1,1}L_2$ must be zero due to the lack of the formal variable X_s.
- The coefficients of the polynomials in $(L_{1,2}L_{2,1}) \cup (L_{1,3}L_{2,1})$ will be zero due to the lack of the formal variable X_α.
- The coefficients of the polynomials in $(L_{1,2}L_{2,2}) \cup (L_{1,3}L_{2,3})$ will be zero due to the presence of the monomial $X_sX_\tau^i$ or $X_sX_w^2$.

It remains to consider the following terms in $(L_{1,2}L_{2,3}) \cup (L_{1,3}L_{2,2})$:

$$\{X_sX_wX_\tau^i(X_\tau - \mathsf{id}^*)\}_{i\in[0,B]} \cup \{X_sX_\alpha H_i + X_sX_wF_i(X_\tau)\}_{i\in[n]}$$

Observe that

- For all $i \neq i^*$, the coefficients of $\{X_sX_\alpha H_i + X_sX_wF_i(X_\tau)\}_{i\neq i^*}$ must be zero since they involve $X_sX_\alpha H_i$ which appears neither in the target polynomials nor other polynomials in the set. Here we use the fact that H_i are all distinct.
- For the remaining polynomials $\{X_sX_wX_\tau^i(X_\tau - \mathsf{id}^*)\}_{i\in[0,B]} \cup \{X_sX_\alpha H_{i^*} + X_sX_wF_{i^*}(X_\tau)\}$, we must have $\{c_i\}_{i\in[0,B]}$ and d such that

  $$X_sX_\alpha H_{i^*} = \sum_{i\in[0,B]} c_i \cdot X_sX_wX_\tau^i(X_\tau - \mathsf{id}^*) + d \cdot (X_sX_\alpha H_{i^*} + X_sX_wF_{i^*}(X_\tau)).$$

  Clearly, we must set $d = 1$ and this means

  $$0 = \sum_{i\in[0,B]} c_i \cdot X_sX_wX_\tau^i(X_\tau - \mathsf{id}^*) + X_sX_wF_{i^*}(X_\tau).$$

  This means

  $$\sum_{i\in[0,B]} c_iX_\tau^i(X_\tau - \mathsf{id}^*) = -F_{i^*}(X_\tau).$$

  The left-hand-side is a polynomial with a root at $X_\tau = \mathsf{id}^*$, so this means F_{i^*} must also have a root at id^*. Thus $F_{i^*}(\mathsf{id}^*) = 0$. By construction of F_{i^*}, this means $\mathsf{id}^* \in S_{i^*}$. This contradicts the requirements of the security game (i.e., the adversary cannot request a key that is able to decrypt the challenge ciphertext).

We conclude that the monomial $X_sX_\alpha H_{i^*} = X_sX_\alpha\varphi_{\mathsf{H}}(\mathsf{tg}^*)$ is linearly independent of the other components in the adversary's view. Since the message in $\mathsf{Hyb}_1^{(0)}$ and $\mathsf{Hyb}_1^{(1)}$ is blinded by $\varphi_{\mathsf{T}}(X_sX_\alpha H_{i^*})$, we conclude that the message is perfectly hidden from the view of the adversary. This proves the lemma. □

Theorem D.3 now follows from Lemmas D.4 and D.5 by a hybrid argument. □

Corollary D.6 (Batched Identity-Based Encryption)**.** *Let λ be a security parameter. If we model* GroupGen *as a generic bilinear group, and* H *as a random oracle, then Construction D.1 is an adaptively-secure batched IBE scheme with the following efficiency properties:*

- **Public key size:** *For a batch size B, the public key contains 4 $\mathbb{G}_1$ elements and B $\mathbb{G}_2$ elements.*
- **Ciphertext size:** *A ciphertext contains 2 $\mathbb{G}_1$ elements and 1 $\mathbb{G}_{\mathsf{T}}$ element.*
- **Digest size:** *A digest contains 1 $\mathbb{G}_2$ element.*
- **Decryption key size:** *A decryption key contains 1 $\mathbb{G}_2$ element.*

E Threshold Batched IBE in the Generic Group Model

In this section, we give a threshold version of Construction D.1. Note that this scheme is in the centralized model where a trusted dealer generates the decryption shares for each user. We start by recalling the formal definition of a threshold batched IBE scheme from [AFP25]. Note here that we work under the weaker notion of security from [AFP25] where the attacker in the security game can only query for the same set S to all decryption authorities for any particular batch label tg. Because we derive our construction by directly thresholding the scheme from Appendix D, we can only argue security under this weaker definition. To achieve the stronger notion of security where the adversary can request shares for different sets to different decryption authorities, we need an alternative approach to secret share the master secret key (see Remark 5.3 and Section 5 for one such approach). In addition, following [AFP25], we consider the setting of *static* corruptions where the adversary declares upfront the set of corrupted users (but is allowed to make adaptive key-generation queries).

Definition E.1 (Threshold Batched Identity-Based Encryption [AFP25, adapted])**.** A threshold batched identity-based encryption scheme $\Pi_{\mathsf{ThBatchIBE}}$ consists of a tuple of efficient algorithms $\Pi_{\mathsf{ThBatchIBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Digest}, \mathsf{CompKeyShare}, \mathsf{CompKeyAggregate}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: On input the security parameter $\lambda \in \mathbb{N}$, the setup algorithm outputs a set of public parameters pp. We assume that the public parameters (implicitly) specifies the message space $\mathcal{M}$, identity space $\mathcal{I}$, and batch label space $\mathcal{T}$ for the encryption scheme.
- $\mathsf{KeyGen}(\mathsf{pp}, 1^B, 1^L, 1^T) \to (\{\mathsf{pk}_\ell, \mathsf{sk}_\ell\}_{\ell\in[L]}, \mathsf{mpk})$: On input the public parameters pp, an upper bound on the batch size B, the size of the decryption committee L, and the threshold T, the key-generation algorithm outputs a set of user public keys pk_i and user secret keys sk_i along with a master public key mpk. We assume that mpk, pk_ℓ, and sk_ℓ also include an implicit description of the message space $\mathcal{M}$, identity space $\mathcal{I}$, and batch label space $\mathcal{T}$ (from pp).
- $\mathsf{Encrypt}(\mathsf{mpk}, m, \mathsf{id}, \mathsf{tg}) \to \mathsf{ct}$: On input the global master public key mpk, a message $m \in \mathcal{M}$, an identity $\mathsf{id} \in \mathcal{I}$, and a batch label $\mathsf{tg} \in \mathcal{T}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Digest}(\mathsf{mpk}, S) \to \mathsf{dig}$: On input the global master public key mpk and a set of identities S, the digest algorithm output a digest dig. This algorithm is deterministic.
- $\mathsf{CompKeyShare}(\mathsf{sk}, \mathsf{dig}, \mathsf{tg}) \to \sigma$: On input a user secret key sk, a digest dig, and a batch label tg, the key-share-computation algorithm outputs a decryption key share σ associated with dig and tg.
- $\mathsf{VerifyKeyShare}(\mathsf{pk}, \mathsf{dig}, \mathsf{tg}, \sigma) \to b$: On input a user public key pk, a digest dig, a batch label tg and a decryption key share σ, the key-share verification algorithm outputs a bit $b \in \{0, 1\}$ indicating whether the decryption key share σ is valid under pk for dig and tg. This algorithm is deterministic.
- $\mathsf{CompKeyAggregate}(\{\sigma_\ell\}_{\ell\in U}, \mathsf{dig}, \mathsf{tg}) \to \bar{\sigma}$: On input a collection of decryption key shares σ_ℓ for a set of users $\ell \in U$, a digest dig, and a batch label tg, the key-share-aggregation algorithm outputs an aggregated decryption key $\bar{\sigma}$. This algorithm is deterministic.
- $\mathsf{Decrypt}(\mathsf{mpk}, \bar{\sigma}, \mathsf{dig}, S, (\mathsf{id}, \mathsf{tg}), \mathsf{ct}) \to m$: On input the global master public key, a decryption key $\bar{\sigma}$, a digest dig, a set of identities S, an identity-label pair $(\mathsf{id}, \mathsf{tg})$, and a ciphertext ct, the decryption algorithm outputs a message $m \in \mathcal{M}$ (or possibly a special symbol $\perp$ to indicate decryption failed). This algorithm is deterministic.

We require $\Pi_{\mathsf{ThBatchIBE}}$ satisfy the following properties:

- **Completeness:** For all $\lambda, B, L, T \in \mathbb{N}$ where $T \le L$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda)$, batch labels $\mathsf{tg}^* \in \mathcal{T}$, all sets $S \subseteq \mathcal{I}$ of size at most B, all $\ell^* \in [L]$, where $\mathcal{T}$ and $\mathcal{I}$ are the batch label and the identity spaces associated with pp, we have

$$\Pr\left[\mathsf{VerifyKeyShare}(\mathsf{pk}_{\ell^*}, \mathsf{dig}, \mathsf{tg}^*, \sigma_{\ell^*}) = 1 : \begin{array}{c}(\{\mathsf{pk}_\ell, \mathsf{sk}_\ell\}_{\ell\in[L]}, \mathsf{mpk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, 1^B, 1^L, 1^T) \\ \mathsf{dig} = \mathsf{Digest}(\mathsf{mpk}, S) \\ \sigma_{\ell^*} \leftarrow \mathsf{CompKeyShare}(\mathsf{sk}_{\ell^*}, \mathsf{dig}, \mathsf{tg}^*)\end{array}\right] = 1.$$

- **Correctness:** For all $\lambda, B, L, T \in \mathbb{N}$ where $T \le L$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda)$, all $(\{\mathsf{pk}_\ell, \mathsf{sk}_\ell\}_{\ell\in[L]}, \mathsf{mpk})$ in the support of $\mathsf{KeyGen}(\mathsf{pp}, 1^B, 1^L, 1^T)$, all messages $m \in \mathcal{M}$, batch labels $\mathsf{tg}^* \in \mathcal{T}$, and identities $\mathsf{id}^* \in \mathcal{I}$ (where $\mathcal{M}, \mathcal{I}, \mathcal{T}$ are the message, identity, and batch label spaces associated with pp, respectively), all sets $S \subseteq \mathcal{I}$ of size at most B where $\mathsf{id}^* \in S$, all sets $U \subseteq [L]$ where $|U| = T$, all ciphertexts ct in the support of $\mathsf{Encrypt}(\mathsf{mpk}, m, \mathsf{id}^*, \mathsf{tg}^*)$, all decryption key shares $\{\sigma_\ell\}_{\ell\in U}$ where $\mathsf{VerifyKeyShare}(\mathsf{pk}_\ell, \mathsf{dig}, \mathsf{tg}^*, \sigma_\ell) = 1$ for all $\ell \in U$ and $\mathsf{dig} = \mathsf{Digest}(\mathsf{mpk}, S)$, it holds that

$$\mathsf{Decrypt}(\mathsf{mpk}, \mathsf{CompKeyAggregate}(\{\sigma_\ell\}_{\ell\in U}, \mathsf{dig}, \mathsf{tg}^*), \mathsf{dig}, S, (\mathsf{id}^*, \mathsf{tg}^*), \mathsf{ct}) = m.$$

- **Adaptive security with static corruptions:** For a security parameter λ, a batch size B, the size of committee L, the threshold T, a set of corrupted authorities $C \subset [L]$ such that $|C| < T$, a bit $\beta \in \{0,1\}$, and an adversary $\mathcal{A}$, we define the threshold batched IBE security game as follows:
  - The challenger starts by computing $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and $(\{\mathsf{pk}_\ell, \mathsf{sk}_\ell\}_{\ell\in[L]}, \mathsf{mpk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, 1^B, 1^L, 1^T)$. It gives $(1^\lambda, 1^B, 1^L, 1^T, \mathsf{pp}, \{\mathsf{pk}_\ell\}_{\ell\in[L]}, \{\mathsf{sk}_\ell\}_{\ell\in C}, \mathsf{mpk})$ to $\mathcal{A}$. Let $\mathcal{M}, \mathcal{I}, \mathcal{T}$ be the message space, identity space, and batch label space associated with pp.
  - Algorithm $\mathcal{A}$ can now make any number of key-share-computation queries. On each query, algorithm $\mathcal{A}$ specifies a set $S \subseteq \mathcal{I}$ where $|S| = B$ and a batch label $\mathsf{tg} \in \mathcal{T}$. The challenger replies with the decryption key shares $\sigma_\ell \leftarrow \mathsf{ComputeKey}(\mathsf{sk}_\ell, \mathsf{Digest}(\mathsf{mpk}, S), \mathsf{tg})$ for all $\ell \notin C$.
  - After $\mathcal{A}$ is finished making key-share-computation queries, it outputs two messages $m_0, m_1 \in \mathcal{M}$, a challenge identity $\mathsf{id}^* \in \mathcal{I}$ and a challenge batch label $\mathsf{tg}^* \in \mathcal{T}$. The challenger responds with a challenge ciphertext $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, m_\beta, \mathsf{id}^*, \mathsf{tg}^*)$.
  - Algorithm $\mathcal{A}$ can continue to make key-share-computation queries. The challenger answers the queries exactly as before.
  - At the end of the game, algorithm $\mathcal{A}$ outputs a bit $\beta' \in \{0,1\}$, which is the output of the experiment.

  We say an adversary $\mathcal{A}$ is admissible if the following two conditions hold:
  - The batch labels tg in the key-share-computation queries are all distinct (i.e., the adversary sees at most one set of decryption key shares for each batch label $\mathsf{tg} \in \mathcal{T}$).
  - Algorithm $\mathcal{A}$ does not make a key-share-computation query on a pair (S, tg) where $\mathsf{tg} = \mathsf{tg}^*$ and $\mathsf{id}^* \in S$.

  We say $\Pi_{\mathsf{ThBatchIBE}}$ is secure if for all polynomials $B = B(\lambda), L = L(\lambda), T = T(\lambda)$ and all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\beta' = 1 : \beta = 0] - \Pr[\beta' = 1 : \beta = 1]| = \mathsf{negl}(\lambda) \tag{E.1}$$

  in the above security game. We say $\Pi_{\mathsf{ThBatchIBE}}$ is secure for a specific batch size $B = B(\lambda)$, committee size $L = L(\lambda)$, and threshold $T = T(\lambda)$ if the above holds for the specific functions B, L, T.

Succinctness: There exists a universal polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda, B, L, T \in \mathbb{N}$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda)$, all $(\mathsf{mpk}, \mathsf{msk})$ in the support of $\mathsf{KeyGen}(\mathsf{pp}, 1^B, 1^L, 1^T)$, all digests dig in the support of $\mathsf{Digest}(\mathsf{mpk}, \cdot)$, and all batch labels $\mathsf{tg} \in \mathcal{T}$ (where $\mathcal{T}$ is the batch label space associated with pp), the running time of $\mathsf{CompKeyShare}(\mathsf{msk}, \mathsf{dig}, \mathsf{tg})$ and the size of the digest dig is $\mathsf{poly}(\lambda)$ and in particular, independent of B. Similarly, for all $U \subseteq [L]$ and all user decryption shares σ_i where $i \in U$, the size of the aggregated decryption key $\bar{\sigma}$ output by $\mathsf{CompKeyAggregate}(\{\sigma_\ell\}_{\ell\in U}, \mathsf{dig}, \mathsf{tg})$ is independent of $|U|$.

Construction E.2 (Threshold Batched Identity-Based Encryption)**.** Let GroupGen be a prime-order bilinear group generator. We construct a threshold batched IBE scheme $\Pi_{\mathsf{ThBatchIBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Digest}, \mathsf{CompKeyShare}, \mathsf{CompKeyAggregate}, \mathsf{Decrypt})$ as follows:

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter λ, the setup algorithm samples $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$. Next, let $\mathsf{H}\colon \{0,1\}^\lambda \to \mathbb{G}_2$ be a hash function (which we model as a random oracle in the security analysis), and outputs the public parameters $\mathsf{pp} = (\mathcal{G}, \mathsf{H})$. The message space associated with pp is $\mathbb{G}_\mathsf{T}$, the identity space is $\mathbb{Z}_p$, and the tag space is $\{0,1\}^\lambda$.

- KeyGen(pp, $1^B, 1^L, 1^T$): On input the public parameters $\mathsf{pp} = \mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e)$, a bound on the batch size B, the size of the decryption committee L, and the threshold $T \le L$, the key-generation algorithm proceeds as follows:
  - Sample exponents $\tau, w, \alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_p$.
  - Let $\mathbf{M} \in \mathbb{Z}_p^{L \times T}$ be the share-generation matrix for a T-out-of-L threshold policy over $\mathbb{Z}_p$. Sample $\boldsymbol{\alpha}, \mathbf{w} \xleftarrow{\mathrm{R}} \mathbb{Z}_p^L$ where $\alpha_1 = \alpha$ and $w_1 = w$.

  For $\ell \in [L]$, let $\mathbf{m}_\ell^\intercal$ be the ℓ^{th} row of $\mathbf{M}$. It then outputs the user public keys $\mathsf{pk}_\ell = ([\mathbf{m}_\ell^\intercal \boldsymbol{\alpha}]_1, [\mathbf{m}_\ell^\intercal \mathbf{w}]_1)$ and user secret keys $\mathsf{sk}_\ell = (\mathbf{m}_\ell^\intercal \boldsymbol{\alpha}, \mathbf{m}_\ell^\intercal \mathbf{w})$ for all $\ell \in [L]$ as well as the global master public key

$$\mathsf{mpk} = \big(\mathcal{G}\,,\ \mathsf{H}\,,\ [\tau]_1\,,\ [\tau]_2\,,\ [\tau^2]_2\,,\ \ldots\,,\ [\tau^B]_2\,,\ [w]_1\,,\ [w\tau]_1\,,\ [\alpha]_1\big). \tag{E.2}$$

- Encrypt(mpk, $[m]_\mathsf{T}$, id, tg): On input the master public key mpk (parsed according to Eq. (E.2)), a message $[m]_\mathsf{T} \in \mathbb{G}_\mathsf{T}$, an identity $\mathsf{id} \in \mathbb{Z}_p$, and a batch label $\mathsf{tg} \in \{0,1\}^\lambda$, the encryption algorithm computes $[h_{\mathsf{tg}}]_2 = \mathsf{H}(\mathsf{tg})$ and samples $s \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. It then outputs the ciphertext

$$\begin{aligned}\mathsf{ct} &= \big([s]_1\,,\ s \cdot [w\tau]_1 - s \cdot \mathsf{id} \cdot [w]_1, s \cdot [\alpha]_1 \cdot [h_{\mathsf{tg}}]_2 + [m]_\mathsf{T}\big)\\ &= \big([s]_1\,,\ [sw(\tau - \mathsf{id})]_1\,,\ [s\alpha h_{\mathsf{tg}} + m]_\mathsf{T}\big)\end{aligned}$$

- Digest(mpk, S): On input the master public key mpk (parsed according to Eq. (E.2)) and a set of identities $S \subseteq \mathbb{Z}_p$ where $|S| \le B$, the digest algorithm defines the polynomial $F_S(x) = \prod_{\mathsf{id} \in S}(x - \mathsf{id})$ whose roots are the identities $\mathsf{id} \in S$. Write $F(x) = \sum_{i \in [0,|S|]} f_i x^i$. Output the digest

$$\mathsf{dig} = \sum_{i \in [0,|S|]} f_i \cdot [\tau^i]_2 = [F_S(\tau)]_2.$$

- CompKeyShare(sk, dig, tg): On input a user secret key $\mathsf{sk} = (u_1, u_2)$, a digest $\mathsf{dig} = [d]_2$, and a batch label $\mathsf{tg} \in \{0,1\}^\lambda$, the key-share-computation algorithm computes $[h_{\mathsf{tg}}]_2 = \mathsf{H}(\mathsf{tg})$ and outputs the decryption key share

$$\sigma = u_1 \cdot [h_{\mathsf{tg}}]_2 + u_2 \cdot [d]_2 = [u_1 h_{\mathsf{tg}} + u_2 d]_2.$$

- VerifyKeyShare(pk, dig, tg, σ): On input a user public key $\mathsf{pk}_i = ([u_1]_1, [u_2]_1)$, a digest $\mathsf{dig} = [d]_2$, a batch label $\mathsf{tg} \in \{0,1\}^\lambda$, and a decryption key share $\sigma = [v]_2$, the key-share verification algorithm computes $[h_{\mathsf{tg}}]_2 = \mathsf{H}(\mathsf{tg})$ and outputs 1 if the following relation holds (and 0 otherwise):

$$[u_1]_1 \cdot [h_{\mathsf{tg}}]_2 + [u_2]_1 \cdot [d]_2 = [1]_1 \cdot [v]_2.$$

- CompKeyAggregate($\{\sigma_i\}_{i \in U}$, dig, tg): On input a collection of decryption key shares $\sigma_\ell = [v_\ell]_2$ for a set of users $\ell \in U \subseteq [L]$ where $|U| = T$, a digest $\mathsf{dig} = [d]_2$, and a batch label $\mathsf{tg} \in \{0,1\}^\lambda$, the key-share-aggregation computes the interpolation vector $\boldsymbol{\omega} \in \mathbb{Z}_p^L$ such that $\boldsymbol{\omega}^\intercal \mathbf{M} = \mathbf{e}_1^\intercal$ and $\omega_\ell = 0$ for all $\ell \notin U$ where $\mathbf{M} \in \mathbb{Z}_p^{T \times L}$ is the share-generating matrix for the T-out-of-L threshold policy. It then outputs the aggregated decryption key:

$$\bar{\sigma} = \sum_{\ell \in U} \omega_\ell \cdot [v_\ell]_2.$$

- Decrypt(mpk, $\bar{\sigma}$, S, (id, tg), ct): On input the master public key mpk (parsed according to Eq. (E.2)), an aggregated decryption key $\bar{\sigma} = [\bar{v}]_2$, the set of identities $S \subseteq \mathbb{Z}_p$, an identity $\mathsf{id} \in S$, a batch label $\mathsf{tg} \in \{0,1\}^\lambda$, and the ciphertext $\mathsf{ct} = ([c_1]_1, [c_2]_1, [c_3]_\mathsf{T})$, the decryption algorithm proceeds as follows:
  - First, it defines the polynomial

$$F_{S \setminus \{\mathsf{id}\}}(x) = \prod_{\mathsf{id}' \in S \setminus \{\mathsf{id}\}} (x - \mathsf{id}').$$

  Compute $[F_{S \setminus \{\mathsf{id}\}}(\tau)]_2 = \sum_{i \in [0,|S|-1]} f_i [\tau^i]_2$, where $F_{S \setminus \{\mathsf{id}\}}(x) = \sum_{i \in [0,|S|-1]} f_i x^i$.

– Then it computes and outputs

$$[c_3]_{\mathsf{T}} - [c_1]_1 \cdot [\bar{v}]_2 + [c_2]_1 \cdot [F_{S\setminus\{\mathsf{id}\}}(\tau)]_2. \tag{E.3}$$

Theorem E.3 (Completeness). *Construction E.2 is complete.*

Proof. Take any $\lambda, B, L, T \in \mathbb{N}$ satisfying $T \le L$ and any choice of public parameters $\mathsf{pp} = (\mathcal{G}, \mathsf{H})$ in the support of $\mathsf{Setup}(1^\lambda)$, where $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathsf{T}}, p, g_1, g_2, e)$. Take any batch label $\mathsf{tg}^* \in \{0,1\}^\lambda$, any set $S \subseteq \mathbb{Z}_p$ of size B, and any index $\ell^* \in [L]$. Sample $(\{\mathsf{pk}_\ell, \mathsf{sk}_\ell\}_{\ell\in[L]}, \mathsf{mpk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, 1^B, 1^L, 1^T)$. By construction, sk_{ℓ^*} and pk_{ℓ^*} have the following form:

$$\mathsf{sk}_{\ell^*} = (u_1, u_2) \quad \text{and} \quad \mathsf{pk}_{\ell^*} = ([u_1]_1, [u_2]_1)$$

where $u_1, u_2 \in \mathbb{Z}_p$. Let $\mathsf{dig} = [d]_2 = \mathsf{Digest}(\mathsf{mpk}, S)$ and $\sigma_{\ell^*} \leftarrow \mathsf{CompKeyShare}(\mathsf{sk}_{\ell^*}, \mathsf{dig}, \mathsf{tg}^*)$. Let $[h_{\mathsf{tg}}]_2 = \mathsf{H}(\mathsf{tg}^*)$. By construction of CompKeyShare, this means

$$\sigma_{\ell^*} = [v]_2 = [u_1 h_{\mathsf{tg}} + u_2 d]_2$$

By definition, the key-share-verification algorithm $\mathsf{VerifyKeyShare}(\mathsf{pk}_{\ell^*}, \mathsf{dig}, \mathsf{tg}^*, \sigma_{\ell^*})$ now checks that

$$[u_1]_1 \cdot [h_{\mathsf{tg}}]_2 + [u_2]_1 \cdot [d]_2 = [1]_1 \cdot [v]_2,$$

which holds by construction. □

Theorem E.4 (Correctness). *Construction E.2 is correct.*

Proof. Let $\lambda, B, L, T \in \mathbb{N}$ where $T \le L$. Take any public parameters $\mathsf{pp} = (\mathcal{G}, \mathsf{H})$ in the support of $\mathsf{Setup}(1^\lambda)$, where $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathsf{T}}, p, g_1, g_2, e)$. Take any message $[m]_{\mathsf{T}} \in \mathcal{M}$, any batch label $\mathsf{tg}^* \in \{0,1\}^\lambda$, any identity $\mathsf{id}^* \in \mathbb{Z}_p$, any set $S \subseteq \mathbb{Z}_p$ of size at most B where $\mathsf{id}^* \in \mathbb{Z}_p$, and any set $U \subseteq [L]$ where $|U| = T$. Let $\mathsf{dig} = \mathsf{Digest}(\mathsf{mpk}, S)$ and sample $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, [m]_{\mathsf{T}}, \mathsf{id}^*, \mathsf{tg}^*)$. Take any collection of decryption key shares $\{\sigma_\ell\}_{\ell\in U}$ where $\mathsf{VerifyKeyShare}(\mathsf{pk}_\ell, \mathsf{dig}, \mathsf{tg}^*, \sigma_\ell) = 1$ for all $\ell \in U$. Let $\bar{\sigma} = \mathsf{CompKeyAggregate}(\{\sigma_\ell\}_{\ell\in U}, \mathsf{dig}, \mathsf{tg}^*)$. Consider now $\mathsf{Decrypt}(\mathsf{mpk}, \bar{\sigma}, \mathsf{dig}, S, (\mathsf{id}^*, \mathsf{tg}^*), \mathsf{ct})$:

- First, let $\mathbf{M} \in \mathbb{Z}_p^{L\times T}$ be the share-generating matrix for the T-out-of-L threshold policy and let $\boldsymbol{\omega} \in \mathbb{Z}_p^L$ be the interpolation vector where $\boldsymbol{\omega}^\intercal \mathbf{M} = \mathbf{e}_1^\intercal$ and $\omega_\ell = 0$ for all $\ell \notin U$. Let $\mathbf{m}_\ell^\intercal$ be the ℓ^{th} row of $\mathbf{M}$,
- Let $[h_{\mathsf{tg}^*}]_2 = \mathsf{H}(\mathsf{tg}^*)$ and define the polynomial $F_S(x) = \prod_{\mathsf{id}\in S}(x - \mathsf{id})$.
- By construction of the above algorithms, the above components can now be expressed as follows:

$$\begin{aligned}
\mathsf{mpk} &= \left(\mathcal{G}, \mathsf{H}, [\tau]_1, [\tau]_2, [\tau^2]_2, \ldots, [\tau^B]_2, [w]_1, [w\tau]_1, [\alpha]_1\right)\\
\mathsf{pk}_\ell &= ([\mathbf{m}_\ell^\intercal \boldsymbol{\alpha}]_1, [\mathbf{m}_\ell^\intercal \mathbf{w}]_1)\\
\mathsf{dig} &= [F_S(\tau)]_2\\
\mathsf{ct} &= ([s]_1, [sw(\tau - \mathsf{id}^*)]_1, [s\alpha h_{\mathsf{tg}^*} + m]_{\mathsf{T}})
\end{aligned}$$

- Write each $\sigma_\ell = [v_\ell]_2$. Since $\mathsf{VerifyKeyShare}(\mathsf{pk}_\ell, \mathsf{dig}, \mathsf{tg}^*, \sigma_\ell) = 1$ for all $\ell \in U$, this means

$$v_\ell = \mathbf{m}_\ell^\intercal \boldsymbol{\alpha} \cdot h_{\mathsf{tg}^*} + \mathbf{m}_\ell^\intercal \mathbf{w} \cdot F_S(\tau).$$

- By construction of CompKeyAggregate, we have

$$\bar{\sigma} = \sum_{\ell\in U} \omega_\ell \cdot [v_\ell]_2 = \sum_{\ell\in U} [\omega_\ell (\mathbf{m}_\ell^\intercal \boldsymbol{\alpha} \cdot h_{\mathsf{tg}^*} + \mathbf{m}_\ell^\intercal \mathbf{w} \cdot F_S(\tau))].$$

  Since $\boldsymbol{\omega}^\intercal \mathbf{M} = \mathbf{e}_1^\intercal$, $\mathbf{e}_1^\intercal \boldsymbol{\alpha} = \alpha$, and $\mathbf{e}_1^\intercal \mathbf{w} = w$, this means

$$\bar{\sigma} = \sum_{\ell\in U} [\omega_\ell (\mathbf{m}_\ell^\intercal \boldsymbol{\alpha} \cdot h_{\mathsf{tg}^*} + \mathbf{m}_\ell^\intercal \mathbf{w} \cdot F_S(\tau))]_2 = [\alpha h_{\mathsf{tg}^*} + w F_S(\tau)]_2.$$

  For ease of notation, define $\bar{v} = \alpha h_{\mathsf{tg}^*} + w F_S(\tau)$. Then, $\bar{\sigma} = [\bar{v}]_2$.

- The decryption algorithm now computes

$$[s\alpha h_{\mathsf{tg}^*} + m]_{\mathsf{T}} - [s]_1 \cdot [\bar{v}]_2 + [sw(\tau - \mathsf{id}^*)]_1 \cdot [F_{S\setminus\{\mathsf{id}\}}(\tau)]_2 = [s\alpha h_{\mathsf{tg}^*} + m - s\alpha h_{\mathsf{tg}^*} - swF_S(\tau) + swF_S(\tau)]_{\mathsf{T}} \\ = [m]_{\mathsf{T}},$$

  and correctness holds. □

Theorem E.5 (Adaptive Security with Static Corruptions)**.** *Take any polynomials $B = B(\lambda)$, $L = L(\lambda)$ and any $T \le L$. Then, Construction E.2 satisfies adaptive security with static corruptions with batch size B, committee size L, threshold T if we model* GroupGen *as a generic group and the hash function* H *as a random oracle.*

Proof. Similar to [AFP25], we will show that security (with static corruptions) of Construction E.2 follows via adaptive security of the centralized batched IBE scheme from Construction D.1. Then, Theorem D.3 immediately implies Theorem E.5. To start, we review the security experiments $\mathsf{Expt}^{(\beta)}_{\mathsf{BatchIBE}}$ and $\mathsf{Expt}^{(\beta)}_{\mathsf{BatchTIBE}}$ associated with the batched IBE scheme (Definition 3.2) and the threshold batched IBE scheme (Definition E.1), respectively, in the generic group model and random oracle model. As in the proof of Theorem D.3, the challenger maintains random injective functions $\varphi_1, \varphi_2, \varphi_{\mathsf{T}} \colon \mathbb{Z}_p \to \mathcal{L}$ and a hash function mapping $\varphi_{\mathsf{H}} \colon \{0,1\}^\lambda \to \mathbb{Z}_p$. The adversary has access to the evaluation oracle and pairing oracle described in Appendix B and the hash oracle behaves as follows:

- **Hash oracle:** On input a batch label $r \in \{0,1\}^\lambda$, the hash oracle returns $\varphi_2(\varphi_{\mathsf{H}}(r))$.

For any adversary $\mathcal{A}$, a batch size parameter $B \in \mathbb{N}$, and a bit $\beta \in \{0,1\}$, we define $\mathsf{Expt}^{(\beta)}_{\mathsf{BatchIBE}}(\mathcal{A}, B)$ as follows:

- The challenger sets $\mathsf{pp} = (\varphi_1(1), \varphi_2(1), \varphi_{\mathsf{T}}(1))$. The challenger samples exponents $\tau, w, \alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_p^*$ and constructs the master public key as follows

$$\mathsf{mpk} = (\varphi_1(\tau), \varphi_2(\tau), \varphi_2(\tau^2), \ldots, \varphi_2(\tau^B), \varphi_1(w), \varphi_1(\alpha), \varphi_1(w\tau)).$$

  The challenger gives pp and mpk to $\mathcal{A}$.

- When algorithm $\mathcal{A}$ makes a key-computation query on a set of identities $S \subseteq \mathbb{Z}_p$ where $|S| \le B$ and a batch label $\mathsf{tg} \in \{0,1\}^\lambda$, the challenger returns $\perp$ if $\mathcal{A}$ has made a query on the same tg. Otherwise, it defines the polynomial $F_S(x) = \prod_{\mathsf{id} \in S}(x - \mathsf{id})$ whose roots are the identities $\mathsf{id} \in S$ and writes $F(x) = \sum_{i \in [0,|S|]} f_i x^i$. It then responds with the secret key

$$\mathsf{sk} = \varphi_2(\alpha\varphi_{\mathsf{H}}(\mathsf{tg}) + wF_S(\tau)).$$

- When algorithms $\mathcal{A}$ outputs a challenge identity id^*, the associated batch label tg^*, and two messages $[m_0]_{\mathsf{T}}, [m_1]_{\mathsf{T}}$ in the image of φ_{T}, the challenger responds with $\perp$ if $\mathcal{A}$ made a key-computation query (tg, S) such that $\mathsf{tg} = \mathsf{tg}^*$ and $\mathsf{id}^* \in S$. Otherwise, the challenger samples $s \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and responds with the challenge ciphertext

$$\mathsf{ct} = (\varphi_1(s), \varphi_1(sw(\tau - \mathsf{id}^*)), \varphi_{\mathsf{T}}(s\alpha\varphi_{\mathsf{H}}(\mathsf{tg}^*) + \varphi_{\mathsf{T}}^{-1}(m_\beta))).$$

- Algorithm $\mathcal{A}$ can continue to make key-computation queries. The challenger responds as described above except that it returns $\perp$ if $\mathsf{tg} = \mathsf{tg}^*$ and $\mathsf{id}^* \in S$.

- At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $\beta' \in \{0,1\}$, which the challenger also outputs.

Next, for any adversary $\mathcal{A}$, parameters $B, L, T \in \mathbb{N}$, a corruption set $C \subseteq [L]$, and a bit $\beta \in \{0,1\}$, we define $\mathsf{Expt}^{(\beta)}_{\mathsf{BatchTIBE}}(\mathcal{A}, C, B, L, T)$ as follows.

- The challenger sets $\mathsf{pp} = (\varphi_1(1), \varphi_2(1), \varphi_{\mathsf{T}}(1))$. The challenger samples exponents $\tau, w, \alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_p^*$ and constructs the master public key as follows

$$\mathsf{mpk} = (\varphi_1(\tau), \varphi_2(\tau), \varphi_2(\tau^2), \ldots, \varphi_2(\tau^B), \varphi_1(w), \varphi_1(\alpha), \varphi_1(w\tau)).$$

Let $\mathbf{M} \in \mathbb{Z}_p^{L \times T}$ be the share-generation matrix for a T-out-of-L threshold policy over $\mathbb{Z}_p$. For $\ell \in [L]$, let $\mathbf{m}_\ell^\intercal$ be the ℓ^{th} row of $\mathbf{M}$. The challenger samples $\boldsymbol{\alpha}, \mathbf{w} \xleftarrow{R} \mathbb{Z}_p^L$ where $\alpha_1 = \alpha$ and $w_1 = w$, and constructs the user public keys and user secret keys for all $\ell \in [L]$ as follows:

$$\mathsf{pk}_\ell = (\varphi_1(\mathbf{m}_\ell^\intercal \boldsymbol{\alpha}), \varphi_1(\mathbf{m}_\ell^\intercal \mathbf{w}))$$
$$\mathsf{sk}_\ell = (\mathbf{m}_\ell^\intercal \boldsymbol{\alpha}, \mathbf{m}_\ell^\intercal \mathbf{w}).$$

The challenger gives the public parameters pp, the master public key mpk, the users' public keys $\{\mathsf{pk}_\ell\}_{\ell \in [L]}$ and the corrupted users' secret keys $\{\mathsf{sk}_\ell\}_{\ell \in C}$ to algorithm $\mathcal{A}$.

- When algorithm $\mathcal{A}$ makes a key-computation query on a set of identities $S \subseteq \mathbb{Z}_p$ where $|S| \leq B$ and a batch label $\mathsf{tg} \in \{0,1\}^\lambda$, the challenger returns $\perp$ if $\mathcal{A}$ has made a query on the same tg. Otherwise, it defines the polynomial $F_S(x) = \prod_{\mathsf{id} \in S}(x - \mathsf{id})$ whose roots are the identities $\mathsf{id} \in S$ and write $F(x) = \sum_{i \in [0,|S|]} f_i x^i$. It then makes a hash oracle query on tg so that $\varphi_{\mathsf{H}}(\mathsf{tg})$ is defined internally and returns decryption key shares

$$\sigma_\ell = \varphi_2(\mathbf{m}_\ell^\intercal \boldsymbol{\alpha} \varphi_{\mathsf{H}}(\mathsf{tg}) + \mathbf{m}_\ell^\intercal \mathbf{w} F_S(\tau)) \quad \forall \ell \notin C$$

- When algorithms $\mathcal{A}$ outputs a challenge identity id^*, the associated batch label tg^*, and two messages $[m_0]_{\mathsf{T}}, [m_1]_{\mathsf{T}}$ in the image of φ_{T}, the challenger returns $\perp$ if $\mathcal{A}$ made a key-computation query (tg, S) such that $\mathsf{tg} = \mathsf{tg}^*$ and $\mathsf{id}^* \in S$. Otherwise, the challenger samples $s \xleftarrow{R} \mathbb{Z}_p$ and returns the challenge ciphertext

$$\mathsf{ct} = (\varphi_1(s), \varphi_1(sw(\tau - \mathsf{id}^*)), \varphi_{\mathsf{T}}(s\alpha\varphi_{\mathsf{H}}(\mathsf{tg}^*) + \varphi_{\mathsf{T}}^{-1}(m_\beta))).$$

- Algorithm $\mathcal{A}$ can continue to make key-computation queries. The challenger responds as described above except that it returns $\perp$ if $\mathsf{tg} = \mathsf{tg}^*$ and $\mathsf{id}^* \in S$.

- At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $\beta' \in \{0,1\}$, which the challenger also outputs.

Suppose now that there exists an adversary $\mathcal{A}$ such that

$$|\Pr[\mathsf{Expt}^{(0)}_{\mathsf{BatchTIBE}}(\mathcal{A}, C, B, L, T) = 1] - \Pr[\mathsf{Expt}^{(1)}_{\mathsf{BatchTIBE}}(\mathcal{A}, C, B, L, T) = 1]| \geq \varepsilon.$$

We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ where

$$|\Pr[\mathsf{Expt}^{(0)}_{\mathsf{BatchIBE}}(\mathcal{A}, B) = 1] - \Pr[\mathsf{Expt}^{(1)}_{\mathsf{BatchIBE}}(\mathcal{A}, B) = 1]| \geq \varepsilon.$$

To do so, we show that for all $B, L, T \in \mathbb{N}$, all $C \subseteq [L]$, all $\beta \in \{0,1\}$, and all adversaries $\mathcal{A}$, there exists an adversary $\mathcal{B}$ such that

$$\Pr[\mathsf{Expt}^{(\beta)}_{\mathsf{BatchTIBE}}(\mathcal{A}, C, B, L, T) = 1] = \Pr[\mathsf{Expt}^{(\beta)}_{\mathsf{BatchIBE}}(\mathcal{B}, B) = 1] \tag{E.4}$$

Algorithm $\mathcal{B}$ works as follows:

- Algorithm $\mathcal{B}$ obtains $\mathsf{pp} = (\varphi_1(1), \varphi_2(1), \varphi_{\mathsf{T}}(1))$ and the master public key

$$\mathsf{mpk} = (\varphi_1(\tau), \varphi_2(\tau), \varphi_2(\tau^2), \ldots, \varphi_2(\tau^B), \varphi_1(w), \varphi_1(\alpha), \varphi_1(w\tau))$$

from the challenger. Let $\mathbf{M} \in \mathbb{Z}_p^{L \times T}$ be the share-generation matrix for a T-out-of-L threshold policy over $\mathbb{Z}_p$. For $\ell \in [L]$, let $\mathbf{m}_\ell^\intercal$ be the ℓ^{th} row of $\mathbf{M}$. Algorithm $\mathcal{B}$ computes a vector $\mathbf{w}^\perp \in \mathbb{Z}_p^L$ such that $\mathbf{m}_\ell^\intercal \mathbf{w}^\perp = 0$ for all indices $\ell \in C$ and $w_1 = 1$. It samples $\tilde{\boldsymbol{\alpha}}, \tilde{\mathbf{w}} \xleftarrow{R} \mathbb{Z}_p$ with $\tilde{\alpha}_1 = 0, \tilde{w}_1 = 0$. For all $\ell \in C$, it computes $\mathsf{sk}_\ell = (\mathbf{m}_\ell^\intercal \tilde{\boldsymbol{\alpha}}, \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}})$. Algorithm $\mathcal{B}$ constructs the user public keys $\{\mathsf{pk}_\ell\}_{\ell \in [L]}$ as follows:

  - For all $\ell \in C$, algorithm $\mathcal{B}$ uses the generic group oracle and $\varphi_1(1)$ to compute $\varphi_1(\mathbf{m}_\ell^\intercal \tilde{\boldsymbol{\alpha}})$ and $\varphi_1(\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}})$. It sets $\mathsf{pk}_\ell = (\varphi_1(\mathbf{m}_\ell^\intercal \tilde{\boldsymbol{\alpha}}), \varphi_1(\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}}))$ and $\mathsf{sk}_\ell = (\mathbf{m}_\ell^\intercal \tilde{\boldsymbol{\alpha}}, \mathbf{m}_\ell^\intercal \tilde{\mathbf{w}})$.
  - For each $\ell \notin C$, algorithm $\mathcal{B}$ uses the generic group oracle in conjunction with $\varphi_1(\alpha)$ and $\varphi_1(w)$ to compute $\varphi_1(\mathbf{m}_\ell^\intercal \tilde{\boldsymbol{\alpha}} + \mathbf{m}_\ell^\intercal \mathbf{w}^\perp \cdot \alpha)$ and $\varphi_1(\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} + \mathbf{m}_\ell^\intercal \mathbf{w}^\perp \cdot w)$. It sets $\mathsf{pk}_\ell = (\varphi_1(\mathbf{m}_\ell^\intercal \tilde{\boldsymbol{\alpha}} + \mathbf{m}_\ell^\intercal \mathbf{w}^\perp \cdot \alpha), \varphi_1(\mathbf{m}_\ell^\intercal \tilde{\mathbf{w}} + \mathbf{m}_\ell^\intercal \mathbf{w}^\perp \cdot w))$.

Algorithm $\mathcal{B}$ forwards the public parameters pp, the master public key mpk, the users' public keys $\{\mathsf{pk}_\ell\}_{\ell\in[L]}$ and the corrupted users' secret keys $\{\mathsf{sk}_\ell\}_{\ell\in C}$ to algorithm $\mathcal{A}$.

- When algorithm $\mathcal{A}$ makes a key-computation query on a set of identities $S \subseteq \mathbb{Z}_p$ where $|S| \le B$ and a batch label $\mathsf{tg} \in \{0,1\}^\lambda$, the challenger returns $\perp$ if $\mathcal{A}$ has already made a query on the same tg. Otherwise, it defines the polynomial $F_S(x) = \prod_{\mathsf{id}\in S}(x - \mathsf{id})$ whose roots are the identities $\mathsf{id} \in S$. Write $F(x) = \sum_{i\in[0,|S|]} f_i x^i$. Then, algorithm $\mathcal{B}$ proceeds as follows:

  1. Algorithm $\mathcal{B}$ uses the generic group oracle in conjunction with $\varphi_2(1), \varphi_2(\tau), \ldots, \varphi_2(\tau^B)$ to compute $f_S = \varphi_2(F_S(\tau))$.
  2. Algorithm $\mathcal{B}$ makes a key-computation query (S, tg) to the batched IBE challenger and receives a secret key $\mathsf{sk}_{S,\mathsf{tg}} = \varphi_2(\alpha\varphi_\mathsf{H}(\mathsf{tg}) + wF_S(\tau))$.
  3. Then, for each $\ell \notin C$, algorithm $\mathcal{B}$ queries the hash oracle on tg to obtain $h_{\mathsf{tg}} = \varphi_2(\varphi_\mathsf{H}(\mathsf{tg}))$. Using the generic group oracle, algorithm $\mathcal{B}$ now computes

$$\sigma_\ell = \varphi_2\big(\mathbf{m}_\ell^\intercal\tilde{\boldsymbol{\alpha}} \cdot \underbrace{\varphi_\mathsf{H}(\mathsf{tg})}_{h_{\mathsf{tg}}} + \mathbf{m}_\ell^\intercal\tilde{\mathbf{w}} \cdot \underbrace{F_S(\tau)}_{f_S} + \mathbf{m}_\ell^\intercal\mathbf{w}^\perp \cdot \underbrace{(\alpha\varphi_\mathsf{H}(\mathsf{tg}) + wF_S(\tau))}_{\mathsf{sk}_{S,\mathsf{tg}}}\big).$$

     Algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with $\{\sigma_\ell\}_{\ell\notin C}$.

- When $\mathcal{A}$ makes a challenge query on an identity id^*, batch label tg^*, and two messages m_0, m_1 in the image of φ_T, the challenger returns $\perp$ if $\mathcal{A}$ made a key-computation query (tg, S) such that $\mathsf{tg} = \mathsf{tg}^*$ and $\mathsf{id}^* \in S$. Otherwise, algorithm $\mathcal{B}$ forwards $(\mathsf{id}^*, \mathsf{tg}^*, m_0, m_1)$ as its challenge to the challenger and receives the challenge ciphertext ct. It replies to $\mathcal{A}$ with ct.

- Algorithm $\mathcal{A}$ can continue to make key-computation queries. Algorithm $\mathcal{B}$ responds as described above except that it returns $\perp$ if $\mathsf{tg} = \mathsf{tg}^*$ and $\mathsf{id}^* \in S$.

- At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $\beta' \in \{0,1\}$, which algorithm $\mathcal{B}$ also outputs.

We now argue that algorithm $\mathcal{B}$ correctly simulates an execution of $\mathsf{Expt}^{(\beta)}_{\mathsf{BatchTIBE}}$ for algorithm $\mathcal{A}$. First, observe that algorithm $\mathcal{B}$ implicitly sets

$$\boldsymbol{\alpha} = \tilde{\boldsymbol{\alpha}} + \mathbf{w}^\perp\alpha \quad \text{and} \quad \mathbf{w} = \tilde{\mathbf{w}} + \mathbf{w}^\perp w$$

where $\alpha, w \xleftarrow{\text{R}} \mathbb{Z}_p$ are the exponents chosen by the batched IBE challenger and the blinding factors $\tilde{\boldsymbol{\alpha}}, \tilde{\mathbf{w}} \xleftarrow{\text{R}} \mathbb{Z}_p^T$ are sampled with the restriction that $\tilde{\alpha}_1 = 0$ and $\tilde{\mathbf{w}}_1 = 0$. Therefore, the distributions of $\boldsymbol{\alpha}$ and $\mathbf{w}$ are uniform over $\mathbb{Z}_p^T$ subject to the restriction that $\alpha_1 = \alpha$ and $w_1 = w$. This is precisely the distribution expected by $\mathcal{A}$. With this choice of variables, we have

$$\mathbf{m}_\ell^\intercal\boldsymbol{\alpha} = \mathbf{m}_\ell^\intercal(\tilde{\boldsymbol{\alpha}} + \mathbf{w}^\perp\alpha) = \begin{cases} \mathbf{m}_\ell^\intercal\tilde{\boldsymbol{\alpha}} + \mathbf{m}_\ell^\intercal\mathbf{w}^\perp \cdot \alpha & \ell \notin C \\ \mathbf{m}_\ell^\intercal\tilde{\boldsymbol{\alpha}} & \ell \in C \end{cases}$$

$$\mathbf{m}_\ell^\intercal\mathbf{w} = \mathbf{m}_\ell^\intercal(\tilde{\mathbf{w}} + \mathbf{w}^\perp w) = \begin{cases} \mathbf{m}_\ell^\intercal\tilde{\mathbf{w}} + \mathbf{m}_\ell^\intercal\mathbf{w}^\perp \cdot w & \ell \notin C \\ \mathbf{m}_\ell^\intercal\tilde{\mathbf{w}} & \ell \in C \end{cases}$$

By construction, this coincides with the distribution that $\mathcal{B}$ uses to simulate the public keys as well as the distribution of the secret keys sk_ℓ for the corrupted users $\ell \in C$. Consider now the response of the key-computation queries from $\mathcal{B}$. For a set $S \subset \mathbb{Z}_p$ where $|S| \le B$, a batch label $\mathsf{tg} \in \mathbb{Z}_p$, and an index $\ell \notin C$, we have

$$\begin{aligned} \mathbf{m}_\ell^\intercal\boldsymbol{\alpha} \cdot \varphi_\mathsf{H}(\mathsf{tg}) + \mathbf{m}_\ell^\intercal\mathbf{w} \cdot F_S(\tau) &= \mathbf{m}_\ell^\intercal\tilde{\boldsymbol{\alpha}} \cdot \varphi_\mathsf{H}(\mathsf{tg}) + \mathbf{m}_\ell^\intercal\tilde{\mathbf{w}} \cdot F_S(\tau) + \mathbf{m}_\ell^\intercal\mathbf{w}^\perp \cdot \alpha\varphi_\mathsf{H}(\mathsf{tg}) + \mathbf{m}_\ell^\intercal\mathbf{w}^\perp \cdot wF_S(\tau) \\ &= \mathbf{m}_\ell^\intercal\tilde{\boldsymbol{\alpha}} \cdot \varphi_\mathsf{H}(\mathsf{tg}) + \mathbf{m}_\ell^\intercal\tilde{\mathbf{w}} \cdot F_S(\tau) + \mathbf{m}_\ell^\intercal\mathbf{w}^\perp \cdot (\alpha\varphi_\mathsf{H}(\mathsf{tg}) + wF_S(\tau)) \end{aligned}$$

This is exactly how algorithm $\mathcal{B}$ answers the key-generation queries in the simulation. Finally, the challenge ciphertext in the two experiments are distributed identically. This means that algorithm $\mathcal{B}$ wins in $\mathsf{Expt}^{(\beta)}_{\mathsf{BatchIBE}}$ as long as algorithm $\mathcal{A}$ wins in $\mathsf{Expt}^{(\beta)}_{\mathsf{BatchTIBE}}$. This shows Eq. (E.4). □

Corollary E.6 (Threshold Batched Identity-Based Encryption)**.** *Let λ be a security parameter. If we model* GroupGen *as a generic bilinear group, and* H *as a random oracle, then Construction E.2 there is a threshold batched IBE scheme satisfying adaptive security with static corruptions together with the following efficiency properties:*

- **Public key size:** *For a batch size B, the public key contains 4 $\mathbb{G}_1$ elements and B $\mathbb{G}_2$ elements.*
- **Ciphertext size:** *A ciphertext contains 2 $\mathbb{G}_1$ elements and 1 $\mathbb{G}_\mathsf{T}$ element.*
- **Digest size:** *A digest contains 1 $\mathbb{G}_2$ element.*
- **Decryption key share size:** *A decryption key share contains 1 $\mathbb{G}_2$ element (as does the aggregated decryption key).*