Adaptively Secure Batched IBE from LWE

Jiaqi Cheng* UW–Madison
Pratish Datta NTT Research†

November 16, 2026

Abstract

We construct the *first adaptively* secure (epoch-based) batched identity-based encryption (BIBE) scheme from the learning with errors (LWE) assumption in the *plain* model. In addition to achieving adaptive security, we obtain significant efficiency improvements over existing lattice-based constructions. The public parameters, batch keys, and ciphertexts have sizes independent of the batch size, and setup requires no prescribed batch-size bound, unlike other lattice-based constructions. The authority generates batch keys from short, publicly computable digests. Encryption time is independent of the batch size, while decrypting an entire batch takes time linear in its size, including public preprocessing.

We first give a plain-model *generic* construction from laconic function evaluation and tagged one-key functional encryption. Building on this generic construction, we then exploit the linear structure of laconic encryption to obtain a more efficient algebraic construction using tagged inner-product functional encryption. Using receiver non-committing encryption, we further obtain an adaptively secure variant in the random oracle model that avoids noise flooding in the laconic encryption component in our plain-model construction and permits a polynomial modulus-to-noise ratio for that component.

Finally, we give a *generic* transformation from BIBE to threshold BIBE using threshold fully homomorphic encryption. The transformation preserves adaptive security against static corruptions and requires only *one* round of communication for batch-key generation after a trusted setup.

*This work was done when the author was an intern at NTT Research. Email: jiaqicheng@cs.wisc.edu.
†Email: pratish.datta@ntt-research.com.

Contents

1 Introduction **3**

2 Technical Overview **6**
- 2.1 A Generic Construction from LFE and Tagged FE . . . 7
- 2.2 From LFE–FE to Laconic Encryption and IPFE . . . 9
- 2.3 Adaptive Security and Error Leakage . . . 10
- 2.4 Thresholdization . . . 11
- 2.5 Ring Translation . . . 12

3 Preliminaries **12**
- 3.1 Lattice Preliminaries . . . 12
- 3.2 Laconic Function Evaluation . . . 13
- 3.3 Laconic Encryption . . . 14
- 3.4 Tagged Functional Encryption against Bounded Collusions . . . 15
- 3.5 Tagged Receiver Non-Committing Encryption . . . 17
- 3.6 Threshold Fully Homomorphic Encryption . . . 18

4 Batched IBE **19**
- 4.1 Batched IBE . . . 19
- 4.2 Threshold Batched IBE . . . 20

5 Batched IBE from Laconic Encryption and Inner-Product Functional Encryption **21**
- 5.1 Security of the LE–IPFE Construction . . . 23

6 Batched IBE from Laconic Function Evaluation and Tagged Functional Encryption **31**
- 6.1 Security of the LFE–Tagged FE Construction . . . 32

7 Batched IBE from RNCE and Error-Leakage LWE **35**
- 7.1 Security . . . 36
- 7.2 Ring-Based Variant . . . 43

8 Threshold Batched IBE from Threshold FHE **43**

9 Implementation and Evaluation **45**

1 Introduction

Batched IBE. Batched identity-based encryption (BIBE) allows a key authority to issue a single short decryption key for a set of identities, called a batch. In ordinary identity-based encryption (IBE) [Sha84, BF01], the authority can authorize decryption for such a set by issuing a separate key for each identity, but the total key size then grows linearly with the batch size. BIBE replaces these separate keys with a single batch key that decrypts ciphertexts under identities in the batch while preserving the privacy of ciphertexts under all other identities. In the epoch-based setting, each ciphertext and batch key is also associated with an epoch, or tag, and a batch key authorizes decryption only when its epoch matches that of the ciphertext. Throughout this paper, BIBE refers to the epoch-based version unless stated otherwise. Encryption does not require knowledge of the batches for which keys will later be issued. We require compact batch keys whose size is independent of the batch size, together with efficient key issuance. Specifically, anyone should be able to compress a batch into a short public digest, from which the authority generates the corresponding batch key without processing the full set. For security, ciphertexts not authorized by any queried batch key must remain private. We seek adaptive security, in which the adversary may request batch keys both before and after receiving the challenge ciphertext. Every queried key must be unauthorized to decrypt the challenge ciphertext, meaning that its epoch differs from the challenge epoch or its batch excludes the challenge identity.

Threshold BIBE distributes the key-generation authority among several authorities, each holding a secret share of the master secret key. Given a batch digest and an epoch, each authority produces a batch-key share, and shares from any set of authorities meeting the reconstruction threshold combine into a single batch key. Efficiency requires the size of each batch-key share and the time to generate and combine shares from the public digest to be independent of the batch size. Security must hold even when an adversary obtains the secret shares of fewer than the threshold number of parties, in addition to batch-key shares for admissible batches.

Mempool Privacy. An important application of BIBE is the efficient realization of mempool privacy in blockchain transactions. A mempool is a collection of pending transactions awaiting inclusion in a block. Block producers select transactions from this pool and determine their order within the block. In a public mempool, adversaries can inspect transaction contents and exploit them through front-running and strategic reordering to obtain maximal extractable value (MEV) [DGK+20]. An *encrypted mempool* stores these transactions as ciphertexts, keeping their contents hidden until consensus fixes which transactions are included in a block and in what order [BO22, KLJD23, FPTX25].

With threshold BIBE, users encrypt each transaction under a distinct identity and an epoch corresponding to the intended block, and submit the ciphertext to the mempool. After consensus fixes the block's contents and order, anyone can compute a short digest of the selected identities. A decryption committee then generates batch-key shares from this digest and epoch, which combine into a batch key once the reconstruction threshold is met [AFP25]. The key remains short independently of the batch size, limiting the communication and on-chain storage needed to publish decryption material [CGPP24, SAS24, CGPW25, BFOQ25, BCF+25]. Publishing this key enables decryption of the selected transactions while preserving *pending transaction privacy* for transactions under excluded identities [CGPW25].

Existing approaches and their limitations. Agarwal, Fernando, and Pinkas [AFP25] introduced the notion of BIBE and presented the first pairing-based instantiation. Their construction supports non-interactive threshold key issuance and is proved secure in the generic group and random oracle models, with one batch key per epoch. TrX [FPTX25] gives epochless batched threshold encryption in the generic group and random oracle models. However, its public parameters depend

on an a priori bound on the total number of batches, after which the public key must be refreshed. Policharla [Pol26] gives a construction with constant-size ciphertexts and multiple batch keys per epoch through multi-message signatures. The number of keys per epoch is fixed at setup, and security is proved in the generic bilinear group model without random oracles. The paper also sketches non-interactive threshold key issuance. Gong et al. [GWWW26] is the first to present a pairing-based BIBE construction in the plain model. Their construction achieves selective security and obtains constant-size ciphertexts and batch keys under a q-type assumption. Their construction also admits a threshold extension with silent setup. They additionally give an adaptively secure variant in the generic group and random oracle models, with a threshold extension secure against static corruptions. The work of [SMM26] gives batched ABE and a threshold extension under the decisional bilinear Diffie–Hellman assumption in the random oracle model. Another work [Lee26] gives verifiable outsourced batched threshold decryption with silent setup under a q-type assumption and extractable proofs of knowledge. Its security reduction avoids the generic group model, while its proposed proof instantiation uses random oracles.

Boneh et al. [BLT25] give the first lattice-based instantiation of BIBE. They present an epochless BIBE scheme from LWE and a threshold extension whose batch-key generation takes two rounds of interaction after setup. They also discuss one-round alternatives that either increase ciphertext and key sizes by a factor equal to the committee size or require additional offline preprocessing. Their concrete construction is selectively secure.[1] However, their constructions make use of random oracles. Xiong et al. [XZC26] construct selectively secure epoch-based BIBE in the plain model under the ℓ-succinct LWE assumption [Wee24]. This removes the random oracle, but at the cost of a parameterized assumption. Their security definition permits polynomially many batch-key queries per epoch, provided that every batch queried at the challenge epoch excludes the challenge identity. They also sketch a two-round threshold extension using the lattice-trapdoor sharing technique of [BKP13]. This leaves the following questions:

Open Problem 1. *Can we construct adaptively secure BIBE in the plain model from a static assumption, in particular from LWE?*

Open Problem 2. *Can such an adaptively secure BIBE scheme support one-round threshold key generation after setup while keeping each authority's computation and communication independent of the batch size, without increasing ciphertext size or requiring offline preprocessing?*

Our Results

We answer both questions in the affirmative. More precisely, our contributions are the following:

- We construct the first adaptively secure BIBE scheme from a static assumption, in particular LWE with superpolynomial modulus-to-noise ratio, in the plain model. For identities of length k and one-bit messages, our public parameters and batch keys have size $O_\lambda(1)$, and our ciphertexts have size $O_\lambda(k)$. Here $O_\lambda(\cdot)$ hides polynomial factors in the security parameter. None of these sizes grows with the batch size, and no maximum batch size is fixed at setup. In [BLT25, XZC26], the modulus and Gaussian sampling widths are chosen according to a prescribed batch-size bound.
- To avoid superpolynomial modulus-to-noise ratio of our plain-model construction, we also give an adaptively secure variant under LWE using tagged receiver non-committing encryption. This variant works in the random oracle model with a polynomial modulus-to-noise ratio.

[1] They suggest obtaining adaptive security from the IBE of Agrawal et al. [ABB10], but do not give the corresponding construction or proof which do not appear to be straightforward, so our comparison records the guarantee established by their formal theorem.

- Finally, we give a generic transformation from any adaptively secure BIBE scheme to a threshold one using threshold fully homomorphic encryption [BGG+18]. Batch-key generation takes, for the first time in lattice-based constructions, one round after a trusted setup, and the transformation preserves adaptive security against static corruptions.

Both the challenge identity and the challenge epoch may be chosen adaptively. At the challenge epoch, the adversary may obtain at most one batch key, either before or after the challenge ciphertext, for a batch excluding the challenge identity. At other epochs, it may obtain polynomially many batch keys. We follow the original epoch-based definition of BIBE [AFP25], also used in the plain model BIBE construction of [GWWW26], so a batch key decrypts ciphertexts only at its own epoch. The single-query restriction at the challenge epoch matches how the primitive is used in practice. For example, in the encrypted mempool application, an epoch identifies a block height, and protocols typically decrypt only one block at each height. It differs from the epochless functionality of [BLT25] and from the multiple admissible challenge epoch keys supported by [XZC26]. We elaborate on our contributions below.

A modular construction. We first give a generic construction in the plain model from laconic function evaluation [QWW18] and tagged one-key functional encryption [GGLW22]. LFE compresses the membership circuit of the batch into a short digest. The functional key uses this digest to perform LFE encoding during decryption, so encryption does not need to know the batch. We instantiate laconic function evaluation with the adaptively secure scheme of [AMR25]. The tagged scheme is obtained from one-key functional encryption [SS10, GVW12] using the compiler of [GGL24]. We use the epoch as the tag and require adaptive simulation security for one functional key at the challenge tag. Polynomially many keys may be queried at other tags. Our LWE-based construction builds on our generic construction by exploiting the linear structure of laconic encryption [DKL+23] and only requires tagged inner-product functional encryption. Only one inner-product term in the LE ciphertext depends on the digest. We generate the remaining ciphertext components directly and use IPFE for this term, instead of evaluating the full LE encryption circuit through general FE. Both constructions separate the public processing of the set from the use of functional keys. For more details, see the technical overview in Section 2 and the constructions and proofs in Sections 5 and 6.

In our LE–IPFE construction, the digest is the root of a Merkle tree [Mer88] built from the classical SIS hashing construction [Ajt96, GGH11], as in lattice-based accumulators [LLNW16]. It consists of one hash output and does not grow with the batch size, giving optimal dependence on this parameter. For a set S, digest generation takes at most $k|S|$ public hash evaluations and does not require the batch-specific trapdoor generation used in [BLT25, WWW25]. The authority then issues the batch key from the digest with one tagged-IPFE key-generation call.

Improved noise parameters in the random oracle model. Our plain-model proof uses noise flooding and requires LWE with a superpolynomial modulus-to-noise ratio for the LE component. We also give an adaptively secure variant in the random oracle model that avoids this additional smudging. The random-oracle variant encrypts the LE–IPFE ciphertext under tagged receiver noncommitting encryption [GSW21]. This allows the proof to use error-leakage LWE [DKL+23], yielding security from LWE with a polynomial modulus-to-noise ratio for the LE component. The RNCE ciphertext length is the inner LE ciphertext length plus a polynomial in the security parameter and epoch length. The receiver-key size is polynomial in the same two parameters. We use the same digest and public preprocessing as in the plain-model construction, and retain its bounds in the batch size.

Efficiency comparison. Our construction improves the dependence on batch size and identity length in the size bounds of the direct lattice-based schemes in Table 1. Compared with [BLT25],

Table 1: Comparison with prior lattice-based BIBE schemes.

Scheme	$\lvert\mathsf{pp}\rvert$	$\lvert\mathsf{sk}\rvert$	$\lvert\mathsf{ct}\rvert$	Security	Model	Assumption
[BGG+14]	$O_\lambda(k \log^2 \ell)$	$O_\lambda(\log^2 \ell)$	$O_\lambda(k \log^2 \ell)$	Selective	Plain	LWE
[CW23]	$O_\lambda(k \log^2 \ell)$	$O_\lambda(1)$	$O_\lambda(k \log^2 \ell)$	Selective	Plain	LWE
[CW24]	$O_\lambda(\log^{O(1)} \ell)$	$O_\lambda(\log^{O(1)} \ell)$	$O_\lambda(k \log^2 \ell)$	Semi-adaptive	Plain	LWE
[Wee25]	$O_\lambda(\log^{O(1)} \ell)$	$O_\lambda(\log^{O(1)} \ell)$	$O_\lambda(\log^{O(1)} \ell)$	Selective	Plain	Succinct LWE
[BLT25, Remark 2]	$O_\lambda(\log^2 \ell)$	$O_\lambda(\log \ell)$	$O_\lambda(k \log^2 \ell)$	Selective	ROM	LWE
[XZC26]	$O_\lambda(k^6)$	$O_\lambda(k \log \ell)$	$O_\lambda(k^2)$	Selective	Plain	Succinct LWE
This work	$\boldsymbol{O_\lambda(1)}$	$\boldsymbol{O_\lambda(1)}$	$\boldsymbol{O_\lambda(k)}$	**Adaptive**	**Plain**	**LWE**

Here k is the identity length and ℓ bounds the batch size. Our construction requires no such bound at setup. The key bound for [CW24] excludes a reusable public component of size $O_\lambda(k \log^{O(1)} \ell)$. The bounds for [XZC26] are re-expressed for k-bit identities using short-integer encodings. Our scheme is epoch-based and permits at most one batch-key query at the challenge epoch, either before or after the challenge ciphertext.

we remove the $\log^2 \ell$ factors in the public-parameter and ciphertext bounds and the $\log \ell$ factor in the batch-key bound. All three sizes are independent of ℓ. This prior construction combines a lattice IBE with the shifted multi-preimage trapdoor sampler of [WWW25]. For [XZC26], the public-parameter and batch-key bounds in the table are $O_\lambda(k^6)$ and $O_\lambda(k \log \ell)$, whereas both are $O_\lambda(1)$ in our construction. We also reduce the ciphertext bound from $O_\lambda(k^2)$ to $O_\lambda(k)$. That scheme modifies the BGG$^+$ IPE scheme and uses matrix commitments [Wee25] under ℓ-succinct LWE [Wee24]. In the table, we also compare with the generic KP-ABE approach, using an ABE key for membership in S with the circuit-ABE schemes of [BGG+14, CW23, CW24, Wee25].

Encryption in both our constructions takes $O_\lambda(k)$ time, independent of ℓ, including IPFE encryption and the outer encryption in the RNCE variant. In [XZC26, Section 1.3], preparing a fresh identity for encryption requires at least linear work in the batch-size bound ℓ. For batch decryption, our constructions take $O_\lambda(\ell k)$ total time for ℓ ciphertexts under distinct identities, including the cost of building the sparse tree once. The explicit construction of [BLT25, Section 5 and Appendix B.4] requires at least quadratic work in ℓ to construct a public trapdoor containing ℓ^2 matrix blocks.

These improvements in ℓ and k already hold for our plain-model construction, but its concrete efficiency is limited by the large lattice parameters required for noise flooding, whose cost is hidden by O_λ. The base construction of [BLT25, Section 5.1] admits a modulus polynomial in the security parameter and batch-size bound, whereas our parameter setting in Section 5 uses an LE modulus $q = 2^{\Theta(\lambda)}$. The RNCE variant of Section 7 avoids this flooding and allows an LE modulus $q = \mathsf{poly}(\lambda)$. This variant preserves adaptive security and the bounds in ℓ and k, but requires the random oracle model and outer RNCE encryption and decryption.

Thresholdization. We give a generic transformation from adaptively secure BIBE to threshold BIBE using TFHE. Batch-key generation takes one round after a trusted setup and agreement on the digest and epoch. The transformation preserves adaptive security against static corruptions and leaves encryption and decryption unchanged. The construction and proof appear in Section 8.

2 Technical Overview

Our goal is to obtain adaptive BIBE from LWE without using a random oracle. This requires the reduction to answer batch-key queries without fixing the challenge identity, epoch, or batch at

setup. We first examine the difficulties with the two most natural approaches given the existing state of the art, namely, the circuit-ABE approach and with removing the random oracle from the construction of [BLT25].

BGG$^+$ encodings and adaptive security. One can obtain BIBE by using a circuit-ABE key for membership in S. The BGG$^+$ construction [BGG+14] supports this functionality through encodings containing noisy products $\mathbf{s}^\top(\mathbf{A}_i - x[i]\mathbf{G})$, where x is the ciphertext attribute. Here $\mathbf{A}_i$ is the public matrix associated with the i-th attribute coordinate, and $\mathbf{A}_0$ is a public matrix whose trapdoor is used for key generation. In the selective proof, the reduction knows x^* before setup and sets $\mathbf{A}_i = \mathbf{A}_0\mathbf{R}_i + x^*[i]\mathbf{G}$ for short matrices $\mathbf{R}_i$. At x^*, the gadget terms cancel, allowing the challenge ciphertext to be generated from LWE samples under $\mathbf{A}_0$. For a key whose circuit rejects x^*, homomorphic evaluation leaves a nonzero gadget term that allows the reduction to sample the key without the master trapdoor. Thus both challenge generation and key simulation depend on the same early choice of x^*. There is also an explicit attack on the underlying encodings when the attribute length is sufficiently large relative to the lattice parameters: choosing x as a function of the public matrices can reveal $\mathbf{s}$ [AMR25].

Removing the random oracle from BLT. In [BLT25], random oracles determine the identity-dependent matrices and the randomness of a public batch-preimage sampler. One can try to derive the identity matrices from fixed public matrices as in [ABB10], add an epoch-dependent matrix, and fix the sampler's randomness at setup. Write $\mathbf{M}_r$ and $\mathbf{F}_t$ for the public matrices associated with identity r and epoch t. The shifted multi-preimage sampler of [WWW25] produces a common digest $\mathbf{d}_S$ and short vectors $\mathbf{w}_r$ such that $\mathbf{M}_r\mathbf{w}_r + \mathbf{G}\mathbf{d}_S = \mathbf{u}$ for every $r \in S$. The authority uses its master trapdoor to sample a short batch key $\mathsf{sk}_{S,t}$ with $\mathbf{F}_t\mathsf{sk}_{S,t} = \mathbf{G}\mathbf{d}_S$. The pair $(\mathbf{w}_r, \mathsf{sk}_{S,t})$ is then a short preimage of $\mathbf{u}$ under $[\mathbf{M}_r \mid \mathbf{F}_t]$ and serves as a decryption key for the corresponding LWE ciphertext. Anyone who knows S can recompute $\mathbf{w}_r$ from the fixed public randomness.

The difficulty arises at the challenge-epoch key query. The reduction does not know a trapdoor for $\mathbf{F}_{t^*}$. As in BLT, it would first sample a short sk^* and then choose $\mathbf{d}^*$ so that $\mathbf{F}_{t^*}\mathsf{sk}^* = \mathbf{G}\mathbf{d}^*$. But $\mathbf{d}^*$ must also be the output of the public digest computation on S^*. In BLT, the simulator can program the sampler's random oracle to return coins that reproduce the desired preimage once the batch is known. Without this oracle, the simulator would have to choose those coins before it publishes the public parameters. This argument still requires a static choice of (r^*, t^*, S^*) before setup. If S^* is chosen later, the coins are already public and cannot be changed.

2.1 A Generic Construction from LFE and Tagged FE

We take an entirely different approach. Our starting point is a generic BIBE construction in which we combine LFE with tagged one-key FE. Laconic function evaluation [QWW18] lets an encoder work with a short digest of a circuit rather than the circuit itself. It has four algorithms. $\mathsf{LFE.Setup}(1^\lambda)$ generates public parameters pk. $\mathsf{LFE.Hash}(\mathsf{pk}, f)$ compresses a circuit f into a digest h and produces auxiliary information ψ. Using this digest, $\mathsf{LFE.Enc}(\mathsf{pk}, h, x)$ produces an encoding E of x without seeing f. Given the circuit and auxiliary information, $\mathsf{LFE.Dec}(\mathsf{pk}, E, f, \psi)$ recovers $f(x)$. The encoding hides everything else about x, and its length and generation time, as well as the digest length, should be succinct. The original LWE-based construction is selectively secure and has a dependence on the circuit depth. We use the adaptively secure construction of [AMR25], based on LWE.

The privacy guarantee is simulation-based. Using only the output $f(x)$, together with the public parameters and the circuit f, a simulator can generate an encoding computationally indistinguishable from a real one, without knowing x. For adaptive security, the order in which f and x are

chosen matters. In the security experiment of [AMR25, Definition 2.8], the adversary chooses f and x after seeing pk, and the challenger then hashes f. We also need to allow x to be chosen after seeing h, since a batch key may be issued before the challenge ciphertext. The hashing algorithm in the concrete construction of [AMR25] is deterministic. Given (pk, f), the adversary can therefore compute the same h itself before choosing x. Thus the security guarantee also covers the order needed here.

Our second ingredient is one-key functional encryption. Functional encryption [BSW11] encrypts an input x so that a key sk_f reveals $f(x)$, rather than all of x. $\mathsf{FE.Setup}(1^\lambda)$ generates a master public key mpk and a master secret key msk. $\mathsf{FE.KeyGen}(\mathsf{msk}, f)$ issues a functional key sk_f, and $\mathsf{FE.Enc}(\mathsf{mpk}, x)$ produces a ciphertext ct. Decryption $\mathsf{FE.Dec}(\mathsf{sk}_f, \mathsf{ct})$ returns $f(x)$. In one-key FE, security is required against an adversary that obtains a single functional key. Such schemes for polynomial-size circuits follow from public-key encryption [SS10, GVW12], and hence from LWE. We use adaptive simulation security as in [GVW12]. The function key may be requested before or after the challenge ciphertext. Apart from public lengths, the simulator only learns $f(x)$ about the challenge plaintext.

We use the tagged version of one-key FE [GGLW22]. Key generation and encryption take an additional tag t, giving $\mathsf{sk}_{t,f}$ and ct_t. Decryption recovers $f(x)$ when the tags agree. The one-key restriction applies at the challenge tag, while polynomially many keys may be queried at other tags. We view epochs as these tags. The compiler of [GGL24] builds tagged FE from ordinary bounded-collusion FE and IBE. Taking the collusion bound to be one and using an LWE-based IBE [ABB10] gives the required tagged one-key FE without a random oracle from LWE.

Combining the two primitives. Let S be a set of identities and define the circuit $F_S(r, m)$ to output m if $r \in S$ and $\perp$ otherwise. Hashing F_S gives (h_S, ψ_S), and we use h_S as the BIBE digest. However, BIBE encryption must work before S is known, while LFE encoding requires h_S. We use FE to perform the LFE encoding at decryption, as in the succinct-FE construction of [QWW18, Appendix C].

For a fixed random tape R, the LFE encoder is a deterministic circuit. The batch key for (S, t) is a tagged FE key for $G_{h_S}(r, m, R) = \big(r, \mathsf{LFE.Enc}(\mathsf{pk}_{\mathsf{LFE}}, h_S, (r, m); R)\big)$. An encryption of m under (r, t) is a tagged FE encryption of (r, m, R) for fresh R. FE decryption returns (r, E), where E is an LFE encoding relative to h_S. The decryptor recomputes the auxiliary hashing information from S and uses it to decode E. Notice that the auxiliary information is not included in the digest: it is used only at decryption, where the set S is already available. Key generation receives only the short value h_S.

The LFE digest replaces the circuit F_S in the batch key, and tagged FE keeps the ciphertext independent of the set. It also explains why simulation security, rather than only indistinguishability security, is useful here. At the challenge epoch, the FE simulator does not receive the plaintext (r^*, m_β, R). It receives only the output of G_{h_S} for the single admissible challenge-epoch key. Suppose first that this key is queried before the challenge. The set S and its digest h_S are already fixed. After the adversary chooses (r^*, m_β), admissibility gives $r^* \notin S$ and hence $F_S(r^*, m_\beta) = \perp$. Adaptive encoder privacy of LFE lets us replace the real encoding in the FE function output by an encoding simulated from $\perp$. Now suppose that the challenge is issued first. In this case, the tagged FE ciphertext simulator is initially given only the plaintext length. If the adversary later asks for a key for (S, t^*), the simulator computes h_S and supplies the same simulated LFE encoding to the post-challenge FE key-generation algorithm. The two query orders therefore lead to the same final view. Once the LFE encoding has been replaced, the only function output revealed at the challenge epoch is generated from $\perp$ and is independent of the challenge message. This gives an adaptively secure BIBE scheme from adaptively private LFE and one-key tagged FE.

2.2 From LFE–FE to Laconic Encryption and IPFE

The generic construction already achieves adaptive security, but uses LFE for a membership circuit and general FE to evaluate the LFE encoder. For better efficiency, we replace LFE with the LWE-based laconic encryption scheme of [DKL+23], which handles the identities in S without first building a membership circuit. LE represents a collection of indexed public keys by a short state. Encryption uses this state and an index, without needing the public key registered at that index. LE consists of the following algorithms. $\mathsf{LE.Setup}(1^\lambda, 1^k)$ generates public parameters pp, an initial state st, and auxiliary information aux. $\mathsf{LE.KGen}(\mathsf{pp})$ generates a public and secret key pair $(\mathsf{pk}, \mathsf{sk})$. The update algorithm $\mathsf{LE.Upd}$ registers pk at an index $r \in \{0,1\}^k$ and updates both st and aux. $\mathsf{LE.Enc}(\mathsf{pp}, \mathsf{st}, r, m)$ encrypts m to index r. The witness-generation algorithm $\mathsf{LE.WGen}$ uses aux to produce a witness wit for the registered pair (r, pk). Decryption $\mathsf{LE.Dec}(\mathsf{sk}, \mathsf{wit}, \mathsf{ct})$ recovers m. The state size and encryption cost are polynomial in λ and k, independent of the number of registered keys. The registration data used to generate witnesses is stored in aux.

In [DKL+23], the state is the root of a sparse binary hash tree. Each index corresponds to a leaf holding its registered public key. Each internal label is the hash of its two child labels. A witness is a hash opening, consisting of the sibling labels along the path from the leaf to the root. We generate one pair $(\mathsf{pk}^\star, \mathsf{sk}^\star)$ at setup and make both keys public. For a set S, we register $\mathsf{pk}^\star$ at each index $r \in S$ and leave the other leaves empty. The root st_S is our digest. The node labels are computed using the classical SIS hash [Ajt96, GGH11]. The digest contains only one hash value regardless of $|S|$, giving optimal dependence on the batch size. Computing it takes at most $k|S|$ public hash evaluations and requires no batch-specific trapdoor generation. Given S, anyone can reconstruct the tree and use $\mathsf{sk}^\star$ to decrypt an LE ciphertext for $r \in S$.

We can combine this with general tagged one-key FE just as before. Encryption under (r, t) encrypts (r, m, R) using FE with tag t, where R is fresh randomness for LE encryption. The batch key for (S, t) is a functional key for the circuit that returns $(r, \mathsf{LE.Enc}(\mathsf{pp}, \mathsf{st}_S, r, m; R))$. Thus FE decryption produces an LE ciphertext under the digest st_S. For $r \in S$, the decryptor reconstructs the tree from S, obtains the witness for r, and uses the public $\mathsf{sk}^\star$ to recover m. Key generation still uses only the digest, and encryption does not need to know S. LE hides a message encrypted to an honestly registered index whose secret key is unknown to the adversary. It also hides a message encrypted to an unregistered index, even if the adversary chose the keys at all other indices and knows their secret keys. In either case, encryptions of two equal-length messages are computationally indistinguishable, given the public state and auxiliary information. The adversary may choose the registrations, challenge index, and messages adaptively after seeing the public parameters [DKL+23, Section 7.1]. For our design, we use the guarantee for unregistered indices. For $r \notin S$, the leaf is unregistered under st_S, so this guarantee applies even though $\mathsf{sk}^\star$ is public.

This LE–FE construction also achieves adaptive security. Recall that FE decryption with a batch key returns an LE ciphertext, not the message itself. For a challenge encrypting m_β under (r^*, t^*) and a batch key for (S, t^*), let E_β be this LE ciphertext under (st_S, r^*). We first use the simulation security of FE. The simulator generates the FE parameters, keys, and challenge ciphertext using only the function output (r^*, E_β) as information about the challenge plaintext. The simulation does not depend on m_β or R beyond this output. What remains is to show that E_β hides m_β. The set S in a permitted challenge-epoch key query must exclude r^*, so no key is registered at the challenge index. By LE message privacy, we can replace E_β with an encryption of the all-zero message of the same length without the adversary noticing. The FE simulator then receives an LE ciphertext that is independent of the challenge message, and so its simulated view also hides that message. Keys at other epochs carry different tags, so simulating them requires no function output on the challenge plaintext.

The LFE construction of [AMR25, Construction 5.3] supports general circuits by combining the homomorphic encryption scheme of [GSW13] with reverse trapdoor hashing. Its encoding overhead grows with the circuit depth. For our membership functionality, LE instead uses lattice hashing and an LWE ciphertext chain, without homomorphic circuit evaluation. Its encryption and decryption use matrix–vector products along a path of length k. The ring-based LE implementation in [DKL+23, Section 12] reports millisecond-scale encryption and decryption for 50-bit indices, excluding the time to sample encryption randomness.

The LE component thus already admits a practically efficient implementation, but evaluating its entire encryption circuit through general tagged FE remains costly. To reduce this cost, we examine how the digest enters the LE ciphertext. The concrete LE construction of [DKL+23] uses an LWE ciphertext chain along the identity path. Write the public matrices as $\mathbf{A}_0, \mathbf{A}_1, \mathbf{B} \in \mathbb{Z}_q^{n \times m}$ and set $\mathbf{A} = [\mathbf{A}_0 \mid \mathbf{A}_1]$. The registered public key is $\mathbf{y}^\star = \mathbf{B}\mathbf{x}^\star$, where $\mathbf{x}^\star$ is the corresponding short secret key, also made public. An empty subtree has label $\mathbf{y}_\perp$. The hash of two child labels is

$$\mathsf{H}_{\mathbf{A}}(\mathbf{y}_0, \mathbf{y}_1) = \mathbf{A}_0\big(-\mathbf{G}^{-1}(\mathbf{y}_0)\big) + \mathbf{A}_1\big(-\mathbf{G}^{-1}(\mathbf{y}_1)\big),$$

where $\mathbf{G}^{-1}$ denotes binary decomposition. We write $\mathbf{y}_S$ for the root representing S. For a bit message μ and identity $r = b_1 \cdots b_k$, an LE ciphertext consists of path components $\mathbf{c}_0, \ldots, \mathbf{c}_k$ and a scalar d. For $j < k$, the path components have the form $\mathbf{c}_j^\top = \mathbf{s}_j^\top \mathbf{A} + \mathbf{s}_{j+1}^\top \mathbf{D}_j + \mathbf{e}_j^\top$, where $\mathbf{D}_j$ selects one of the two gadget blocks according to the next bit of r. The last component is $\mathbf{c}_k^\top = \mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top$. The scalar is

$$d = \mathbf{s}_0^\top \mathbf{y}_S + \bar{e} + \lfloor q/2 \rfloor \mu,$$

where the vectors $\mathbf{s}_j$ are fresh randomness and $\mathbf{e}_j, \bar{e}$ are small errors.

The key observation is that the path components do not depend on the digest, and only d involves the root $\mathbf{y}_S$. We can therefore generate the path components directly and use tagged IPFE to recover d. The IPFE ciphertext encrypts $(\mathbf{s}_0, \bar{e} + \lfloor q/2 \rfloor \mu)$ under tag t, and the batch key is an IPFE key for $(\mathbf{y}_S, 1)$ at the same tag. Their inner product is exactly d, so decryption then proceeds as in LE. Both vectors have dimension $n + 1$, independent of the number of identities in S and the path length k. The BIBE ciphertext contains the public path components and this one IPFE ciphertext, rather than an FE encryption of the inputs to the entire LE encryption circuit.

We again require adaptive simulation security for one function key at the challenge tag, with polynomially many keys allowed at other tags. The identity-based IPFE of [LLW21], instantiated with the IPFE scheme of [ALS16], provides the tagged structure. Its IND security is lifted to one-key adaptive simulation security using the transformation of [ET25]. The transformation adds one coordinate and chooses it so that the simulated plaintext has the prescribed inner product. Only one underlying IPFE key is queried at the challenge tag, so the state used by the modulo-p IPFE construction is immaterial there. It remains to prove adaptive security when the path components are published outside FE, which we discuss next.

2.3 Adaptive Security and Error Leakage

Our goal is to avoid noise flooding in the LE component and prove adaptive security with a polynomial modulus-to-noise ratio. We first explain why the error-leakage proof does not apply directly. The IPFE plaintext $(\mathbf{s}_0, \bar{e} + \lfloor q/2 \rfloor \mu)$ contains the same $\mathbf{s}_0$ used in the public path. In particular, the first path component and the IPFE output satisfy

$$\mathbf{c}_0^\top = \mathbf{s}_0^\top \mathbf{A} + \mathbf{s}_1^\top \mathbf{D}_0 + \mathbf{e}_0^\top, \qquad d = \mathbf{s}_0^\top \mathbf{y}_S + \bar{e} + \lfloor q/2 \rfloor \mu.$$

After invoking IPFE simulation security, we therefore need to show that d hides the message even when the public path components are given. Rather than invoking LE security as a black box, we need to examine the algebraic structure of the construction. The hash equations express the root term $\mathbf{s}_0^\top \mathbf{y}_S$ through the path ciphertexts and error terms such as $\mathbf{e}_i^\top \mathbf{z}_i$, where $\mathbf{z}_i$ comes from the hash opening for S. The reduction receives an LWE sample but does not know its error $\mathbf{e}_i$, so it cannot directly compute this contribution to d. The LE proof of [DKL+23] uses error-leakage LWE to obtain a noisy version of the required inner product. However, the leakage vector must be fixed before the LWE sample is received. When the batch key is queried after the challenge, the reduction must obtain the sample and issue the public path before learning S and the corresponding leakage vector. We can supply d to the IPFE simulator later, but cannot request the required leakage after receiving the sample. Thus the error-leakage proof cannot be applied directly.

In the standard model, we instead hide these error terms with a wider independent error $\bar{e}$ in d. For a correction Δ_S at one hybrid step, we choose the width so that $\bar{e} - \Delta_S$ is statistically close to $\bar{e}$. The short opening vectors give a bound on Δ_S for every S, so this width can be fixed without knowing the set. If the challenge comes first, the simulated IPFE ciphertext uses only the plaintext length, and the public path does not involve $\bar{e}$. The later choice of S is therefore independent of this masking error. Noise flooding removes the corrections, after which ordinary LWE randomizes the ciphertext chain. Since the challenge identity lies outside S, its path reaches an empty subtree, where a uniform scalar hides the message. This proves adaptive security, at the cost of a larger noise term. For more details, see Section 5.1.

Using RNCE to Delay the Inner Ciphertext. We can avoid the additional smudging by encrypting the LE–IPFE ciphertext under tagged receiver non-committing encryption, with the epoch as the tag. The RNCE simulator can issue the outer ciphertext using only the length of the inner ciphertext. Once a later batch-key query supplies S, the reduction fixes the leakage vector and generates the inner ciphertext. A simulated receiver key then opens the outer ciphertext to this value. This restores the order required by the error-leakage proof [DKL+23]. With the RNCE scheme of [GSW21], we obtain adaptive security in the random oracle model from LWE with a polynomial modulus-to-noise ratio. The construction and proof appear in Section 7.

2.4 Thresholdization

We apply the threshold-FHE approach of [BGG+18, Section 7] to BIBE key generation. A trusted setup publishes a TFHE encryption of the BIBE master secret key msk and a PRF seed s, and gives each of N parties a share of the TFHE secret key. The seed fixes the random tape for each digest–epoch pair $(\mathbf{d}, t)$ as $F_s(\langle \mathbf{d}, t \rangle)$. Given $(\mathbf{d}, t)$, the parties homomorphically evaluate the same deterministic circuit for $\mathsf{BIBE.KeyGen}$ on this ciphertext. Each party independently publishes a partial decryption of the result. Combining any θ honestly generated partial decryptions recovers the batch key, so batch-key generation takes one round after setup. The circuit takes only the digest and epoch as public inputs, not the full set S. Digest computation, encryption, and decryption remain unchanged.

For security, the TFHE simulator generates the published ciphertext and the shares using only the requested batch keys. We then replace the PRF values by independent random tapes, retaining the same tape for repeated queries with the same $(\mathbf{d}, t)$. The resulting experiment reduces to adaptive BIBE security. The corrupted set is fixed before setup, while the challenge identity, epoch, and share queries may be chosen adaptively. The construction and proof appear in Section 8.

2.5 Ring Translation

We give an informal overview of the translation to the ring setting. For better concrete efficiency, we consider a ring-based instantiation of LE. This requires tagged IPFE over the same ring, which we obtain from IPFE over $\mathbb{Z}_q$ through polynomial evaluation. We choose a prime q such that the defining polynomial splits into distinct linear factors over $\mathbb{Z}_q$. A ring element is then uniquely determined by its values at these roots. The number theoretic transform (NTT) computes these values. Ring addition and multiplication act pointwise, so a ring inner product becomes an ordinary inner product over $\mathbb{Z}_q$ at each evaluation point. We evaluate the IPFE plaintext and function vectors at each point and use an independent tagged-IPFE instance with epoch t as the tag. For ring degree N, this gives N instances of dimension $n+1$, each with at most one function key at the challenge epoch. An inverse NTT combines their outputs into the ring element needed for LE decryption. IPFE returns exact values modulo q, so we recover exactly the same result as the ring inner product. A brief description appears in Section 7.2.

3 Preliminaries

3.1 Lattice Preliminaries

Notation. Let PPT denote probabilistic polynomial-time. We will use lowercase bold letters for vectors (e.g. $\mathbf{v}$), uppercase bold letters for matrices (e.g. $\mathbf{A}$) and assume all vectors are column vectors. The jth row of a matrix $\mathbf{A}$ is denoted by $\mathbf{A}[j]$. For any integer $q \geq 2$, we let $\mathbb{Z}_q$ denote the ring of integers modulo q. We represent $\mathbb{Z}_q$ as integers in the range $(-q/2, q/2]$. For a vector $\mathbf{v}$, we let $\|\mathbf{v}\|$ denote its ℓ_2 norm and $\|\mathbf{v}\|_\infty$ denote its infinity norm. Similarly, for matrices $\|\cdot\|$ and $\|\cdot\|_\infty$ denote their ℓ_2 and infinity norms (respectively).

For two distributions X, Y, over a finite domain Ω, the statistical distance between X and Y is defined as $\mathsf{SD}(X,Y) \stackrel{\text{def}}{=} \frac{1}{2}\sum_{\omega\in\Omega}|X(\omega) - Y(\omega)|$. A family of distributions $\mathcal{D}_1 = \{\mathcal{D}_1(\lambda)\}_\lambda$ and $\mathcal{D}_2 = \{\mathcal{D}_2(\lambda)\}_\lambda$, parameterized by security parameter λ, are said to be statistically indistinguishable, represented by $\mathcal{D}_1 \approx_s \mathcal{D}_2$, if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{SD}(\mathcal{D}_1(\lambda), \mathcal{D}_2(\lambda)) \leq \mathsf{negl}(\lambda)$. For a family of distributions $\mathcal{D} = \{\mathcal{D}(\lambda)\}_\lambda$ over the integers, and integers bounds $B = \{B(\lambda)\}_\lambda$, we say that $\mathcal{D}$ is B-bounded if $\Pr[|x| \leq B(\lambda) : x \leftarrow \mathcal{D}(\lambda)] = 1$. In words, a B-bounded distribution is supported only on the range $[-B, B]$. Below we state the "smudging" lemma as it appears in prior works.

Learning with Errors For our constructions, we rely on the learning with errors assumption.

Assumption 3.1 (Learning with Errors)**.** Let n, m, q be positive integers and let χ be a noise distribution over $\mathbb{Z}_q$. The Learning with Errors assumption $\mathsf{LWE}_{n,m,q,\chi}$ states that the following two distributions are computationally indistinguishable:

$$\left\{\left(\mathbf{A}, \mathbf{s}^\top\mathbf{A} + \mathbf{e}^\top\right) : \mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m},\ \mathbf{s} \leftarrow \mathbb{Z}_q^n,\ \mathbf{e} \leftarrow \chi^m\right\} \approx_c \left\{\left(\mathbf{A}, \mathbf{u}^\top\right) : \mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m},\ \mathbf{u} \leftarrow \mathbb{Z}_q^m\right\}.$$

Learning with Errors with Error Leakage We use the following single-leakage variant of LWE, adapted from [DKL+23, Definition 5].

Assumption 3.2 (Learning with Errors with Error Leakage)**.** Let n, d, q be positive integers and $\rho > 0$, let χ and $\bar{\chi}$ be distributions over the integers, and let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a two-stage PPT adversary. The $\mathsf{elLWE}_{n,d,q,\chi,\bar{\chi},\rho}$ experiment is defined as follows:

1. The challenger samples $b \leftarrow \{0,1\}$ and $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times d}$, and gives $\mathbf{A}$ to $\mathcal{A}_1$.
2. The adversary outputs $\mathbf{z} \in \mathbb{Z}^d$. If $\|\mathbf{z}\|_\infty > \rho$, the experiment samples $b' \leftarrow \{0,1\}$ and terminates.
3. The challenger samples $\mathbf{e} \leftarrow \chi^d$ and $\bar{e} \leftarrow \bar{\chi}$, and sets $l := \mathbf{e}^\top \mathbf{z} + \bar{e}$. If $b = 0$, it samples $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ and sets $\mathbf{v}^\top := \mathbf{s}^\top \mathbf{A} + \mathbf{e}^\top$. If $b = 1$, it samples $\mathbf{v} \leftarrow \mathbb{Z}_q^d$ uniformly.
4. It gives $(\mathbf{A}, \mathbf{v}, l)$ to $\mathcal{A}_2$, which outputs a bit b'.

The $\mathsf{elLWE}_{n,d,q,\chi,\bar{\chi},\rho}$ assumption states that, for every such adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $|\Pr[b' = b] - 1/2| \leq \mathsf{negl}(\lambda)$. The vector $\mathbf{e}$ is reduced modulo q in $\mathbf{v}$, while l is computed over the integers.

Corollary 3.3 (Error-Leakage LWE from LWE)**.** Let d and ρ be polynomials and let $\epsilon > 0$ be negligible. Let $a, b \geq 2\sqrt{2}$ satisfy $b^2 \geq ((a^2+1)(a^2+2)/a^2)d\rho^2$. For a common base width $\sigma_0 \geq \max\{\eta_\epsilon(\mathbb{Z}^d), \eta_\epsilon(\mathbb{Z})\}$, where η_ϵ denotes the standard smoothing parameter, let $\chi^\star$, χ, and $\bar{\chi}$ be discrete Gaussians over $\mathbb{Z}$ with widths $(a/2)\sigma_0$, $\sqrt{a^2+1}\sigma_0$, and $\sqrt{b^2+1}\sigma_0$. An elLWE distinguisher of advantage δ against $\mathsf{elLWE}_{n,d,q,\chi,\bar{\chi},\rho}$ gives an LWE distinguisher of advantage at least $\delta - 44\epsilon$ against $\mathsf{LWE}_{n,d,q,\chi^\star}$.

Proof. Apply [DKL+23, Theorem 2] with $\Lambda_1 = \mathbb{Z}^d$, $\Lambda_2 = \mathbb{Z}$, $Z = \mathbf{z}^\top$, $\Sigma_0 = \sigma_0^2 I_d$, and $T_0 = \sigma_0^2$. The spectral norm satisfies $\|\mathbf{z}^\top\|_2 = \|\mathbf{z}\|_2 \leq \sqrt{d}\,\rho$. The stated bounds therefore satisfy the theorem's conditions, and the reduction in [DKL+23, Theorem 3] gives the claim. □

Lemma 3.4 (Smudging Lemma [AJL+12, Lemma 2.1, paraphrased])**.** Let B_1, B_2 be integer-valued functions that are efficiently representable and let $\mathcal{D} = \{\mathcal{D}(\lambda)\}_\lambda$ be any B_1-bounded distribution family. Let $U = \{U(\lambda)\}_\lambda$ and $U(\lambda)$ denote the uniform distribution over integers $[-B_2(\lambda), B_2(\lambda)]$. The distribution families $\mathcal{D}+U$ and U are statistically indistinguishable, $\mathcal{D}+U \approx_s U$, if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $B_1(\lambda)/B_2(\lambda) \leq \mathsf{negl}(\lambda)$.

Gadget Matrix. Let $\tau := \lceil \log q \rceil$ and $m := n\tau$. Define $\mathbf{g}^\top := (1, 2, \ldots, 2^{\tau-1})$ and $\mathbf{G} := I_n \otimes \mathbf{g}^\top \in \mathbb{Z}_q^{n \times m}$. We use the deterministic canonical decomposition $\mathbf{G}^{-1} : \mathbb{Z}_q^n \to \{0,1\}^m$, which satisfies $\mathbf{G}\mathbf{G}^{-1}(\mathbf{y}) = \mathbf{y}$ for every $\mathbf{y} \in \mathbb{Z}_q^n$. In particular, $\|\mathbf{G}^{-1}(\mathbf{y})\|_1 \leq m$ and $\|\mathbf{G}^{-1}(\mathbf{y})\|_\infty \leq 1$.

3.2 Laconic Function Evaluation

Let $m = m(\lambda)$ and $\ell = \ell(\lambda)$ be positive integers, and let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ be a function class where every $f \in \mathcal{F}_\lambda$ maps $\{0,1\}^m$ to $\{0,1\}^\ell$. An LFE scheme for $\mathcal{F}$ consists of four PPT algorithms $\mathsf{LFE} = (\mathsf{LFE.Setup}, \mathsf{LFE.Hash}, \mathsf{LFE.Enc}, \mathsf{LFE.Dec})$ with the following syntax:

$\mathsf{LFE.Setup}(1^\lambda) \to \mathsf{pk}$. The setup algorithm takes as input the security parameter λ, and outputs a public key pk.

$\mathsf{LFE.Hash}(\mathsf{pk}, f) \to (h, \psi)$. The deterministic hashing algorithm takes as input a public key pk and a function $f \in \mathcal{F}_\lambda$. It outputs a digest h and the hasher's private information ψ.

$\mathsf{LFE.Enc}(\mathsf{pk}, h, x) \to E$. The encoding algorithm takes as input a public key pk, a digest h, and an input $x \in \{0,1\}^m$. It outputs an encoding E.

$\mathsf{LFE.Dec}(\mathsf{pk}, E, f, \psi) \to y$. The deterministic decoding algorithm takes as input a public key pk, an encoding E, a function $f \in \mathcal{F}_\lambda$, and the hasher's private information ψ. It outputs a value $y \in \{0,1\}^\ell$.

We require an LFE scheme to satisfy the following properties:

Correctness. An LFE scheme LFE is said to be correct if there exists a negligible function $\mathsf{negl}(\cdot)$ such that, for every sufficiently large $\lambda \in \mathbb{N}$, every $f \in \mathcal{F}_\lambda$, and every $x \in \{0,1\}^m$, the following holds:

$$\Pr\left[\mathsf{LFE.Dec}(\mathsf{pk}, E, f, \psi) = f(x) : \begin{array}{c} \mathsf{pk} \leftarrow \mathsf{LFE.Setup}(1^\lambda) \\ (h, \psi) \coloneqq \mathsf{LFE.Hash}(\mathsf{pk}, f) \\ E \leftarrow \mathsf{LFE.Enc}(\mathsf{pk}, h, x) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Adaptive encoder privacy. An LFE scheme LFE satisfies adaptive encoder privacy if there exists a PPT simulator $\mathsf{LFE.Sim}$ such that, for every PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that, for every $\lambda \in \mathbb{N}$,

$$\left|\Pr\left[b' = b : \begin{array}{c} \mathsf{pk} \leftarrow \mathsf{LFE.Setup}(1^\lambda) \\ (f, \mathsf{aux}_0) \leftarrow \mathcal{A}_0(1^\lambda, \mathsf{pk}) \\ (h, \psi) \coloneqq \mathsf{LFE.Hash}(\mathsf{pk}, f) \\ (x, \mathsf{aux}_1) \leftarrow \mathcal{A}_1(h, \mathsf{aux}_0); \quad b \leftarrow \{0,1\} \\ E_b \leftarrow \begin{cases} \mathsf{LFE.Enc}(\mathsf{pk}, h, x), & \text{if } b = 0, \\ \mathsf{LFE.Sim}(1^\lambda, \mathsf{pk}, f, f(x)), & \text{if } b = 1 \end{cases} \\ b' \leftarrow \mathcal{A}_2(E_b, \mathsf{aux}_1) \end{array}\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda),$$

Succinctness. The laconic requirement is that the digest and the encoder's complexity are sublinear in the description size of f, ideally, they are independent of $|f|$.

Corollary 3.5 (Laconic Function Evaluation from LWE)**.** Let $m = m(\lambda)$, $\ell = \ell(\lambda)$, and $D = D(\lambda)$ be polynomials. Assuming the hardness of LWE with a superpolynomial modulus-to-noise ratio, there exists a correct LFE scheme with deterministic hashing for depth-D functions $f : \{0,1\}^m \to \{0,1\}^\ell$ that satisfies adaptive encoder privacy and has communication $m + \ell + D \cdot \mathsf{poly}(\lambda)$.

Abram, Malavolta, and Roy [AMR25] give such an LFE construction under a slightly different definition, where x is chosen before h is revealed, while here x may be chosen after seeing h. Since their hashing algorithm is deterministic, the adversary can compute h from (pk, f) before choosing x, so their security implies adaptive encoder privacy as defined above.

3.3 Laconic Encryption

Syntax. A laconic encryption scheme for index space $\{0,1\}^k$, public-key space PK, and message space $\mathcal{M}$ consists of the following polynomial-time algorithms [DKL+23]:

$\mathsf{LE.Setup}(1^\lambda, 1^k) \to (\mathsf{pp}, \mathsf{st}, \mathsf{aux})$. The setup algorithm takes as input the security parameter λ and the index length k, and outputs public parameters pp, a public state st, and auxiliary information aux.

$\mathsf{LE.KGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$. The key-generation algorithm takes as input the public parameters and outputs a public and secret key pair $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{LE.Upd}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{st}, \mathsf{ind}, \mathsf{pk}) \to \mathsf{st}'$. The update algorithm, with read-and-write random access to aux, takes as input an index $\mathsf{ind} \in \{0,1\}^k$ and either a public key $\mathsf{pk} \in \mathsf{PK}$ or $\perp$, and outputs an updated public state st'. The input $\perp$ removes the key currently registered at ind.

$\mathsf{LE.Enc}(\mathsf{pp}, \mathsf{st}, \mathsf{ind}, \mathsf{msg}) \to \mathsf{ct}$. The encryption algorithm takes as input the public parameters, the current public state, an index $\mathsf{ind} \in \{0,1\}^k$, and a message $\mathsf{msg} \in \mathcal{M}$, and outputs a ciphertext ct.

$\mathsf{LE.WGen}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{st}, \mathsf{ind}, \mathsf{pk}) \to \mathsf{wit}$. The witness-generation algorithm, with read-only random access to aux, takes as input the public parameters, the current public state, an index, and a public key, and outputs a membership or non-membership witness wit.

$\mathsf{LE.Dec}(\mathsf{sk}, \mathsf{wit}, \mathsf{ct}) \to \mathsf{msg}$. The decryption algorithm takes as input a secret key, a membership witness, and a ciphertext, and outputs a message $\mathsf{msg} \in \mathcal{M}$ or $\perp$.

A laconic encryption scheme LE as defined above is required to satisfy the following properties. The experiments maintain shared variables $(\mathsf{st}, \mathsf{aux})$, a dictionary Honest, and a set $\mathsf{Malicious}$, and use the following stateful oracles:

$\mathcal{O}_{\mathsf{kgen}}(\mathsf{ind})$**:** Sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{LE.KGen}(\mathsf{pp})$, set $\mathsf{st} \leftarrow \mathsf{LE.Upd}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{st}, \mathsf{ind}, \mathsf{pk})$ and $\mathsf{Honest}[\mathsf{ind}] := (\mathsf{pk}, \mathsf{sk})$, and return $(\mathsf{st}, \mathsf{aux}, \mathsf{pk})$.

$\mathcal{O}_{\mathsf{upd}}(\mathsf{ind}, \mathsf{pk})$**:** Set $\mathsf{st} \leftarrow \mathsf{LE.Upd}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{st}, \mathsf{ind}, \mathsf{pk})$ and $\mathsf{Honest}[\mathsf{ind}] := \perp$. If $\mathsf{pk} \neq \perp$, add ind to $\mathsf{Malicious}$. Return $(\mathsf{st}, \mathsf{aux})$.

Correctness. The scheme is statistically correct if, for every polynomial $k = k(\lambda)$ and every possibly unbounded adversary $\mathcal{A}$ whose output satisfies $\mathsf{Honest}[\mathsf{ind}^*] \neq \perp$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr\left[\mathsf{LE.Dec}(\mathsf{sk}^*, \mathsf{wit}^*, \mathsf{ct}^*) = \mathsf{msg}^* : \begin{array}{c} \mathsf{Honest} := \emptyset; \quad \mathsf{Malicious} := \emptyset \\ (\mathsf{pp}, \mathsf{st}, \mathsf{aux}) \leftarrow \mathsf{LE.Setup}(1^\lambda, 1^k) \\ (\mathsf{ind}^*, \mathsf{msg}^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{kgen}}, \mathcal{O}_{\mathsf{upd}}}(\mathsf{pp}, \mathsf{st}, \mathsf{aux}) \\ (\mathsf{pk}^*, \mathsf{sk}^*) := \mathsf{Honest}[\mathsf{ind}^*] \\ \mathsf{ct}^* \leftarrow \mathsf{LE.Enc}(\mathsf{pp}, \mathsf{st}, \mathsf{ind}^*, \mathsf{msg}^*) \\ \mathsf{wit}^* \leftarrow \mathsf{LE.WGen}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{st}, \mathsf{ind}^*, \mathsf{pk}^*) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Security. A laconic encryption scheme LE is secure if, for every admissible stateful PPT adversary $\mathcal{A}$ and every polynomial $k = k(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that, for every $\lambda \in \mathbb{N}$,

$$\left|\Pr\left[b' = b : \begin{array}{c} \mathsf{Honest} := \emptyset; \quad \mathsf{Malicious} := \emptyset \\ (\mathsf{pp}, \mathsf{st}, \mathsf{aux}) \leftarrow \mathsf{LE.Setup}(1^\lambda, 1^k) \\ ((\mathsf{ind}_0, \mathsf{msg}_0), (\mathsf{ind}_1, \mathsf{msg}_1)) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{kgen}}, \mathcal{O}_{\mathsf{upd}}}(\mathsf{pp}, \mathsf{st}, \mathsf{aux}) \\ b \leftarrow \{0,1\} \\ \mathsf{ct}^* \leftarrow \mathsf{LE.Enc}(\mathsf{pp}, \mathsf{st}, \mathsf{ind}_b, \mathsf{msg}_b) \\ b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{kgen}}, \mathcal{O}_{\mathsf{upd}}}(\mathsf{ct}^*) \end{array}\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda),$$

where $\mathcal{A}$ is admissible if $\{\mathsf{ind}_0, \mathsf{ind}_1\} \cap \mathsf{Malicious} = \emptyset$ when it outputs its challenge.

3.4 Tagged Functional Encryption against Bounded Collusions

Syntax. A tagged functional encryption scheme for a circuit family $\mathcal{C}$, input space $\mathcal{X}$, and tag space $\mathcal{T}$ consists of the following polynomial-time algorithms:

$\mathsf{Setup}(1^\lambda) \to (\mathsf{mpk}, \mathsf{msk})$. The setup algorithm takes as input the security parameter λ, and outputs a master public key mpk and a master secret key msk.

$\mathsf{KeyGen}(\mathsf{msk}, t, C) \to \mathsf{sk}_{t,C}$. The key generation algorithm takes as input the master secret key msk, a tag $t \in \mathcal{T}$, and a circuit $C \in \mathcal{C}$, and outputs a functional secret key $\mathsf{sk}_{t,C}$.

$\mathsf{Enc}(\mathsf{mpk}, t, x) \to \mathsf{ct}_t$. The encryption algorithm takes as input the master public key mpk, a tag $t \in \mathcal{T}$, and an input $x \in \mathcal{X}$, and outputs a ciphertext ct_t.

$\mathsf{Dec}(\mathsf{sk}_{t,C}, \mathsf{ct}_t) \to y$. The decryption algorithm takes as input a functional secret key $\mathsf{sk}_{t,C}$ and a ciphertext ct_t associated with the same tag t, and outputs a string y.

Definition 3.6 (Tagged Functional Encryption)**.** A tagged functional encryption scheme FE as defined above is required to satisfy the following properties:

Correctness. For every tag $t \in \mathcal{T}$, circuit $C \in \mathcal{C}$, and input $x \in \mathcal{X}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that, for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_{t,C}, \mathsf{ct}_t) = C(x) : \begin{array}{c}(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{sk}_{t,C} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, t, C) \\ \mathsf{ct}_t \leftarrow \mathsf{Enc}(\mathsf{mpk}, t, x)\end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

q-Adaptive Simulation Security. There exists a stateful PPT simulator S with the following algorithms:

$$S.\mathsf{Setup}(1^\lambda) \to (\mathsf{mpk}, \mathsf{st}), \qquad S.\mathsf{KeyGen}(\mathsf{st}, t, C, \mu) \to \mathsf{sk}_{t,C}, \qquad S.\mathsf{Enc}(\mathsf{st}, t, 1^L, \Pi) \to \mathsf{ct}_t.$$

Here μ is either $\perp$ or a function value, and Π is a list of circuit–value pairs. For every stateful PPT adversary $\mathcal{A}$ making at most q functional-key queries in total at the challenge tag, the following experiments are computationally indistinguishable:

1. **Real experiment.** The challenger samples $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$ and gives mpk to $\mathcal{A}$. Each functional-key query (t_i, C_i) is answered by running $\mathsf{FE.KeyGen}(\mathsf{msk}, t_i, C_i)$ and returning the resulting key sk_{t_i,C_i}. When $\mathcal{A}$ submits a challenge tag and input (t^*, x^*), the challenger returns $\mathsf{ct}^* \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}, t^*, x^*)$. The adversary may continue to make functional-key queries before outputting a bit.
2. **Ideal experiment.** The simulator generates $(\mathsf{mpk}, \mathsf{st}) \leftarrow S.\mathsf{Setup}(1^\lambda)$ and gives mpk to $\mathcal{A}$. Each pre-challenge functional-key query (t_i, C_i) is answered using $S.\mathsf{KeyGen}$ with $\mu = \perp$. When $\mathcal{A}$ submits (t^*, x^*), let $\Pi^* := \{(C_i, C_i(x^*)) \mid t_i = t^*\}$ for all pre-challenge queries associated with the challenge tag. The simulator is given only $(t^*, 1^{|x^*|}, \Pi^*)$ and returns $\mathsf{ct}^* \leftarrow S.\mathsf{Enc}(\mathsf{st}, t^*, 1^{|x^*|}, \Pi^*)$. Each post-challenge query (t_j, C_j) with $t_j = t^*$ is answered using $S.\mathsf{KeyGen}$ with $\mu = C_j(x^*)$, while every post-challenge query (t_j, C_j) with $t_j \neq t^*$ is answered using $S.\mathsf{KeyGen}$ with $\mu = \perp$. Finally, the experiment outputs the bit produced by $\mathcal{A}$.

Formally, for every such adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that, for every $\lambda \in \mathbb{N}$,

$$\left|\Pr\left[\mathsf{Real}^{\mathsf{Tag}}_{\mathsf{FE},\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Ideal}^{\mathsf{Tag}}_{\mathsf{FE},S,\mathcal{A}}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

Corollary 3.7 (Tagged Functional Encryption from LWE)**.** Assuming the standard LWE assumption, there exists a tagged functional encryption scheme for the class of all polynomial-size circuits $\mathcal{C}$ that satisfies 1-adaptive simulation security following Definition 3.6.

Agrawal, Boneh, and Boyen [ABB10] construct an adaptively secure identity-based encryption scheme in the standard model from LWE. Under LWE, the construction of Gorbunov, Vaikuntanathan, and Wee [GVW12] gives a 1-adaptive simulation-secure functional encryption scheme for all polynomial-size circuits. Applying the generic non-black-box compiler of Garg, Goyal, and Lu [GGL24] gives the corollary.

Corollary 3.8 (Tagged Inner-Product FE from LWE)**.** Let p be a prime and $\ell = \ell(\lambda)$ be a polynomial. Assuming the standard LWE assumption, there exists a tagged functional encryption scheme for the function family

$$\mathcal{C}_{\mathsf{IP}} := \left\{C_{\mathbf{y}} : \mathbb{Z}_p^\ell \to \mathbb{Z}_p\right\}_{\mathbf{y} \in \mathbb{Z}_p^\ell}, \qquad C_{\mathbf{y}}(\mathbf{x}) := \langle \mathbf{x}, \mathbf{y} \rangle \bmod p,$$

that satisfies 1-adaptive simulation security following Definition 3.6.

The identity-based scheme in [LLW21] uses the IPFE of [ALS16]. The scheme is written for integer inner products. For inner products over $\mathbb{Z}_p$, we use modulo-p ALS in the same proof. This scheme works over an LWE modulus $Q = p^a$ and sets $\Delta = Q/p$. It encodes $\mathbf{x}$ as $\Delta\mathbf{x}$. Decryption recovers $\Delta\langle \mathbf{x}, \mathbf{y}\rangle$ modulo Q up to a small error. Since $\Delta p = Q$, rounding to the nearest multiple of Δ gives $\langle \mathbf{x}, \mathbf{y}\rangle \bmod p$.

Note that modulo-p ALS uses stateful key generation to maintain consistency among several function queries after lifting them from $\mathbb{Z}_p$ to $\mathbb{Z}_Q$. Here, the LLW proof sends only the challenge-identity key query to ALS and generates all other keys with $\mathsf{SampleRight}$. Since there is only one key at the challenge tag, ALS is queried only once.

LLW gives IND security. To obtain simulation security, we use the transformation of [ET25]. It adds one coordinate to the IPFE vectors. A real ciphertext encrypts $(\mathbf{x}, 0)$, and a key for $\mathbf{y}$ uses $(\mathbf{y}, t)$. The simulator instead encrypts $(\widehat{\mathbf{x}}, 1)$. Given $z = \langle \mathbf{x}, \mathbf{y}\rangle \bmod p$, it chooses $\widehat{\mathbf{x}}$ and t so that $\langle \widehat{\mathbf{x}}, \mathbf{y}\rangle + t = z \bmod p$. Depending on the query order, the simulator chooses t or $\widehat{\mathbf{x}}$ first and determines the other from this equation. The output remains z, while IND security hides which encoding was encrypted.

Corollary 3.9 (Tagged Inner-Product FE from LWE in the QROM)**.** Assuming the standard LWE assumption, there exists a tagged inner-product functional encryption scheme that satisfies 1-adaptive simulation security following Definition 3.6 with a tight reduction in the quantum random oracle model.

Edamura and Takayasu [ET25] also give an efficient construction with a tight reduction in the quantum random oracle model.

3.5 Tagged Receiver Non-Committing Encryption

Syntax. A tagged receiver non-committing encryption scheme with tag space $\mathcal{T}$ and message space $\mathcal{N}$ consists of four polynomial-time algorithms:

$\mathsf{RNCE.Setup}(1^\lambda) \to (\mathsf{npk}, \mathsf{nmsk})$. The setup algorithm outputs a public key npk and a master secret key nmsk.

$\mathsf{RNCE.KeyGen}(\mathsf{nmsk}, t) \to \mathsf{dk}_t$. The key generation algorithm takes an input tag $t \in \mathcal{T}$ and outputs a receiver key dk_t.

$\mathsf{RNCE.Enc}(\mathsf{npk}, t, M) \to C_t$. The encryption algorithm takes an input tag $t \in \mathcal{T}$ and a message $M \in \mathcal{N}$ and outputs a ciphertext C_t.

$\mathsf{RNCE.Dec}(\mathsf{dk}_t, C_t) \to M$. The decryption algorithm outputs a message in $\mathcal{N}$ or $\perp$.

We require a tagged RNCE scheme to satisfy the following properties.

Correctness. For every $t \in \mathcal{T}$ and $M \in \mathcal{N}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr\left[\mathsf{RNCE.Dec}(\mathsf{dk}_t, C_t) = M : \begin{array}{c}(\mathsf{npk}, \mathsf{nmsk}) \leftarrow \mathsf{RNCE.Setup}(1^\lambda) \\ \mathsf{dk}_t \leftarrow \mathsf{RNCE.KeyGen}(\mathsf{nmsk}, t) \\ C_t \leftarrow \mathsf{RNCE.Enc}(\mathsf{npk}, t, M)\end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Post-challenge receiver simulatability. There exists a stateful PPT simulator S with the following algorithms:

$$\mathsf{RNCE.SimSetup}(1^\lambda) \to (\mathsf{npk}, \mathsf{st}), \qquad \mathsf{RNCE.SimKeyGen}(\mathsf{st}, t) \to \mathsf{dk}_t,$$
$$\mathsf{RNCE.SimEnc}(\mathsf{st}, t, 1^L) \to (C_t, \tau), \qquad \mathsf{RNCE.SimOpen}(\mathsf{st}, \tau, t, M) \to \mathsf{dk}_t,$$

such that the following experiments are computationally indistinguishable for every stateful PPT adversary $\mathcal{A}$.

1. **Real experiment.** The challenger samples $(\mathsf{npk}, \mathsf{nmsk}) \leftarrow \mathsf{RNCE.Setup}(1^\lambda)$ and gives npk to $\mathcal{A}$. Every key query t is answered with $\mathsf{dk}_t \leftarrow \mathsf{RNCE.KeyGen}(\mathsf{nmsk}, t)$. When $\mathcal{A}$ submits a challenge (t^*, M^*), the challenger returns $C^* \leftarrow \mathsf{RNCE.Enc}(\mathsf{npk}, t^*, M^*)$. The adversary may continue to make key queries before outputting a bit.
2. **Ideal experiment.** The simulator generates $(\mathsf{npk}, \mathsf{st}) \leftarrow \mathsf{RNCE.SimSetup}(1^\lambda)$ and gives npk to $\mathcal{A}$. Every key query t before the challenge is answered with $\mathsf{RNCE.SimKeyGen}(\mathsf{st}, t)$. When $\mathcal{A}$ submits (t^*, M^*), the simulator is given only $(t^*, 1^{|M^*|})$ and runs

$$(C^*, \tau) \leftarrow \mathsf{RNCE.SimEnc}(\mathsf{st}, t^*, 1^{|M^*|}).$$

   Every post-challenge query $t \neq t^*$ is answered with $\mathsf{RNCE.SimKeyGen}(\mathsf{st}, t)$. If $\mathcal{A}$ queries t^*, the simulator is given M^* and returns $\mathsf{dk}_{t^*} \leftarrow \mathsf{RNCE.SimOpen}(\mathsf{st}, \tau, t^*, M^*)$.

The adversary is admissible if it makes no key query for t^* before the challenge and at most one such query after the challenge. Formally, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\left|\Pr\left[\mathsf{Real}^{\mathsf{Tag}}_{\mathsf{RNCE},\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Ideal}^{\mathsf{Tag}}_{\mathsf{RNCE},S,\mathcal{A}}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

In the random oracle model, both experiments give $\mathcal{A}$ access to the same random oracle, and the ideal simulator may program it.

Corollary 3.10 (Tagged RNCE from LWE in the ROM)**.** Let $\ell_{\mathcal{T}} = \ell_{\mathcal{T}}(\lambda)$ be a polynomial and let the tags have a fixed injective encoding into $\{0,1\}^{\ell_{\mathcal{T}}}$. Assuming the standard LWE assumption, there exists a tagged RNCE scheme for polynomial-length messages that satisfies post-challenge receiver simulatability in the programmable random oracle model. For messages of length L, the public key, master secret key, and receiver keys have length $\mathsf{poly}(\lambda, \ell_{\mathcal{T}})$ independent of L, and the ciphertext has length $L + \mathsf{poly}(\lambda, \ell_{\mathcal{T}})$.

The construction of [GSW21] gives weak non-committing encryption from semantically secure public-key encryption and a random-oracle extension for arbitrary-length messages whose key sizes are independent of the message length. Instantiating the tagged version with the LWE-based adaptively secure identity-based encryption of [ABB10] gives the corollary.

3.6 Threshold Fully Homomorphic Encryption

We use threshold fully homomorphic encryption as defined in [BGG+18], restricted to threshold access structures. Let $N = N(\lambda)$ be the number of parties, let $\theta \in [N]$ be the reconstruction threshold, and let $D = D(\lambda)$ be a circuit-depth bound. A TFHE scheme consists of five polynomial-time algorithms:

$\mathsf{TFHE.Setup}(1^\lambda, 1^D, 1^N, \theta) \to (\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_N)$. The setup algorithm takes as input the security parameter λ, a circuit-depth bound D, the number of parties N, and a reconstruction threshold $\theta \in [N]$. It outputs a public key pk and a secret key share sk_i for each party $i \in [N]$.

$\mathsf{TFHE.Enc}(\mathsf{pk}, \mu) \to c$. The encryption algorithm takes as input a public key pk and a bit $\mu \in \{0,1\}$. It outputs a ciphertext c.

$\mathsf{TFHE.Eval}(\mathsf{pk}, C, \mathbf{c}) \to \widehat{c}$. The deterministic evaluation algorithm takes as input a public key pk, a circuit $C : \{0,1\}^L \to \{0,1\}$ of depth at most D, and a tuple $\mathbf{c} = (c_1, \ldots, c_L)$ of ciphertexts. It outputs an evaluated ciphertext $\widehat{c}$.

$\mathsf{TFHE.PartDec}(\mathsf{pk}, \widehat{c}, \mathsf{sk}_i) \to p_i$. The partial decryption algorithm takes as input a public key pk, an evaluated ciphertext $\widehat{c}$, and party i's secret key share sk_i. It outputs a partial decryption p_i.

$\mathsf{TFHE.FinDec}(\mathsf{pk}, \{(i, p_i)\}_{i \in J}) \to \mu$. The final decryption algorithm takes as input a public key pk and partial decryptions $\{(i, p_i)\}_{i \in J}$ for a set $J \subseteq [N]$. It outputs a bit $\mu \in \{0,1\}$ or $\perp$.

Correctness. For every polynomially bounded N, D, L, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds for all $\lambda \in \mathbb{N}$, every $\theta \in [N]$, every $J \subseteq [N]$ with $|J| \geq \theta$, every $x \in \{0,1\}^L$, and every circuit $C : \{0,1\}^L \to \{0,1\}$ of depth at most D. Generate $(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_N) \leftarrow \mathsf{TFHE.Setup}(1^\lambda, 1^D, 1^N, \theta)$ and $\mathbf{c} \leftarrow \mathsf{TFHE.Enc}(\mathsf{pk}, x)$, and set $\widehat{c} := \mathsf{TFHE.Eval}(\mathsf{pk}, C, \mathbf{c})$. Then

$$\Pr\left[\mathsf{TFHE.FinDec}(\mathsf{pk}, \{(i, p_i)\}_{i \in J}) = C(x) : \ p_i \leftarrow \mathsf{TFHE.PartDec}(\mathsf{pk}, \widehat{c}, \mathsf{sk}_i) \text{ for } i \in J \ \right] \geq 1 - \mathsf{negl}(\lambda).$$

The probability is over all randomness in these algorithms, including setup and encryption.

Compactness. For a single output bit, the evaluated ciphertext has length $\mathsf{poly}(\lambda, D)$, and each partial decryption has length $\mathsf{poly}(\lambda, D, N)$, independently of the input length and circuit size.

Security. We use the joint formulation of semantic and simulation security from [BGG+18, Section 7], omitting the verification components. There exists a stateful PPT simulator $\mathsf{Sim} = (\mathsf{Sim}_0, \mathsf{Sim}_1)$ such that the following experiments are computationally indistinguishable for every stateful PPT adversary $\mathcal{A}$. In both experiments, $\mathcal{A}$ first chooses N, D, θ, a string $x \in \{0,1\}^L$, and a corrupted set $I \subseteq [N]$ with $|I| < \theta$.

1. **Real experiment.** The challenger generates $(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_N) \leftarrow \mathsf{TFHE.Setup}(1^\lambda, 1^D, 1^N, \theta)$ and $\mathbf{c} \leftarrow \mathsf{TFHE.Enc}(\mathsf{pk}, x)$, and gives $(\mathsf{pk}, \mathbf{c}, \{\mathsf{sk}_i\}_{i \in I})$ to $\mathcal{A}$. On each query (C, J), where C has depth at most D and $J \subseteq [N]$, it computes $\widehat{c} := \mathsf{TFHE.Eval}(\mathsf{pk}, C, \mathbf{c})$ and returns $\{p_i \leftarrow \mathsf{TFHE.PartDec}(\mathsf{pk}, \widehat{c}, \mathsf{sk}_i)\}_{i \in J}$.
2. **Ideal experiment.** The simulator runs $\mathsf{Sim}_0(1^\lambda, 1^D, 1^N, \theta, 1^L)$ to obtain a public key pk, ciphertexts $\mathbf{c}$, shares $\mathsf{sk}_1, \ldots, \mathsf{sk}_N$, and state st. It gives $(\mathsf{pk}, \mathbf{c}, \{\mathsf{sk}_i\}_{i \in I})$ to $\mathcal{A}$. On each query (C, J), it receives only $C(x)$ and answers with $\mathsf{Sim}_1(\mathsf{st}, C, J, C(x))$.

The adversary may make polynomially many adaptive queries, including repeated queries and queries with $|J| < \theta$, before outputting a bit. The simulator is not given x. Formally, for every such adversary, there exists a negligible function $\mathsf{negl}(\cdot)$ such that, for all $\lambda \in \mathbb{N}$,

$$\left|\Pr\left[\mathsf{Real}_{\mathsf{TFHE},\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Ideal}_{\mathsf{TFHE},\mathsf{Sim},\mathcal{A}}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

Corollary 3.11 (Threshold FHE from LWE)**.** Let N and D be polynomials and let $\theta \in [N]$. Assuming LWE for the parameters in [BGG+18, Section 5.2.1], there exists a compact TFHE scheme satisfying correctness and security as above.

4 Batched IBE

4.1 Batched IBE

Let $\mathcal{I}$, $\mathcal{T}$, and $\mathcal{M}$ denote the identity space, epoch space, and message space, respectively. No maximum batch size is fixed during setup. A batched IBE scheme is specified by the following five algorithms: Setup, Digest, KeyGen, Enc, and Dec.

$\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{msk})$. The setup algorithm takes as input the security parameter λ, and outputs public parameters pp and a master secret key msk.

$\mathsf{Digest}(\mathsf{pp}, S) \to \mathbf{d}$. The deterministic digest algorithm takes a finite batch $S \subseteq \mathcal{I}$ and outputs a digest $\mathbf{d}$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathbf{d}, t) \to \mathsf{sk}$. The key generation algorithm takes the master secret key msk, a digest $\mathbf{d}$, and an epoch $t \in \mathcal{T}$, and outputs an epoch-bound secret key sk.

$\mathsf{Enc}(\mathsf{pp}, r, t, m) \to \mathsf{ct}$. The encryption algorithm encrypts $m \in \mathcal{M}$ under identity $r \in \mathcal{I}$ and epoch $t \in \mathcal{T}$.

$\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct}, S, r, t) \to m$. The decryption algorithm takes public parameters pp, an epoch-bound secret key sk, a ciphertext ct, a batch S, and an identity-epoch pair (r, t), and outputs a message m.

Correctness. There exists a negligible function $\mathsf{negl}(\cdot)$ such that, for all $\lambda \in \mathbb{N}$, every finite batch $S \subseteq \mathcal{I}$, every identity $r \in S$, every epoch $t \in \mathcal{T}$, and every message $m \in \mathcal{M}$,

$$\Pr\left[\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct}, S, r, t) = m : \begin{array}{c}(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathbf{d} \leftarrow \mathsf{Digest}(\mathsf{pp}, S) \\ \mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathbf{d}, t) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, r, t, m)\end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Security. A batched IBE scheme satisfies adaptive security if for any stateful admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that, for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}^{\mathcal{O}_{\mathsf{msk}}(\cdot,\cdot)}(\mathsf{ct}^*) = b : \begin{array}{c}(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda);\ b \leftarrow \{0,1\} \\ (r^*, t^*, m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{msk}}(\cdot,\cdot)}(1^\lambda, \mathsf{pp}) \\ \mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pp}, r^*, t^*, m_b)\end{array}\right] \leq 1/2 + \mathsf{negl}(\lambda),$$

where $\mathcal{O}_{\mathsf{msk}}(S, t)$ computes $\mathbf{d} \leftarrow \mathsf{Digest}(\mathsf{pp}, S)$ and returns $\mathsf{KeyGen}(\mathsf{msk}, \mathbf{d}, t)$. We say $\mathcal{A}$ is admissible if every key query (S, t) satisfies $S \subseteq \mathcal{I}$ and $t \in \mathcal{T}$, the challenge tuple satisfies $r^* \in \mathcal{I}$, $t^* \in \mathcal{T}$, and $m_0, m_1 \in \mathcal{M}$ with $|m_0| = |m_1|$, and among all key queries with epoch t^*, there is at most one such query (S, t^*) and it satisfies $r^* \notin S$.

4.2 Threshold Batched IBE

We adapt the one-round threshold BIBE definition of [BLT25] to the epoch-based setting. Let $N = N(\lambda)$ be the number of parties and let $\theta \in [N]$ be the reconstruction threshold. No maximum batch size is fixed during setup. A threshold BIBE scheme consists of six polynomial-time algorithms:

$\mathsf{Setup}(1^\lambda, 1^N, \theta) \to (\mathsf{pp}, \mathsf{msk}_1, \ldots, \mathsf{msk}_N)$. The setup algorithm outputs public parameters pp and a master secret key share msk_i for each party $i \in [N]$.

$\mathsf{Digest}(\mathsf{pp}, S) \to \mathbf{d}$. The deterministic digest algorithm takes a finite batch $S \subseteq \mathcal{I}$ and outputs a digest $\mathbf{d}$.

$\mathsf{KeyShare}(\mathsf{pp}, \mathsf{msk}_i, \mathbf{d}, t) \to \sigma_i$. Party i takes its master secret key share, a digest $\mathbf{d}$, and an epoch $t \in \mathcal{T}$, and outputs a batch-key share σ_i.

$\mathsf{Combine}(\mathsf{pp}, \mathbf{d}, t, \{(i, \sigma_i)\}_{i \in J}) \to \mathsf{sk}$. The combining algorithm takes shares from a set $J \subseteq [N]$ and outputs a batch key sk or $\perp$. It outputs $\perp$ if $|J| < \theta$.

$\mathsf{Enc}(\mathsf{pp}, r, t, m) \to \mathsf{ct}$. The encryption algorithm encrypts $m \in \mathcal{M}$ under identity $r \in \mathcal{I}$ and epoch $t \in \mathcal{T}$.

$\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct}, S, r, t) \to m$. The decryption algorithm takes a batch key, a ciphertext, a batch, and an identity–epoch pair, and outputs a message or $\perp$.

Each party computes its share from the common request $(\mathbf{d}, t)$ without receiving another party's message. Thus key generation requires one round after setup and agreement on the request.

Correctness. For every polynomially bounded $N = N(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that, for all $\lambda \in \mathbb{N}$, every $\theta \in [N]$, every $J \subseteq [N]$ with $|J| \geq \theta$, every finite batch $S \subseteq \mathcal{I}$, every $r \in S$, $t \in \mathcal{T}$, and $m \in \mathcal{M}$,

$$\Pr\left[\mathsf{Dec}(\mathsf{pp},\mathsf{sk},\mathsf{ct},S,r,t) = m : \begin{array}{c}(\mathsf{pp},\mathsf{msk}_1,\ldots,\mathsf{msk}_N) \leftarrow \mathsf{Setup}(1^\lambda,1^N,\theta)\\ \mathbf{d} := \mathsf{Digest}(\mathsf{pp},S)\\ \sigma_i \leftarrow \mathsf{KeyShare}(\mathsf{pp},\mathsf{msk}_i,\mathbf{d},t) \text{ for } i \in J\\ \mathsf{sk} \leftarrow \mathsf{Combine}(\mathsf{pp},\mathbf{d},t,\{(i,\sigma_i)\}_{i\in J})\\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp},r,t,m)\end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Security. A threshold BIBE scheme satisfies adaptive security against static corruptions if, for every stateful admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that, for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}^{\mathcal{O}_{\mathsf{sh}}}(\mathsf{ct}^*) = b : \begin{array}{c}(N,\theta,I) \leftarrow \mathcal{A}(1^\lambda)\\ (\mathsf{pp},\mathsf{msk}_1,\ldots,\mathsf{msk}_N) \leftarrow \mathsf{Setup}(1^\lambda,1^N,\theta)\\ b \leftarrow \{0,1\}\\ (r^*,t^*,m_0,m_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{sh}}}(\mathsf{pp},\{\mathsf{msk}_i\}_{i\in I})\\ \mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pp},r^*,t^*,m_b)\end{array}\right] \leq 1/2 + \mathsf{negl}(\lambda),$$

where $\mathcal{O}_{\mathsf{sh}}(S,t)$ computes $\mathbf{d} := \mathsf{Digest}(\mathsf{pp},S)$ and returns $\{\sigma_i \leftarrow \mathsf{KeyShare}(\mathsf{pp},\mathsf{msk}_i,\mathbf{d},t)\}_{i\in[N]\setminus I}$. The adversary is admissible if N is polynomially bounded, $\theta \in [N]$, $I \subseteq [N]$, $|I| < \theta$, all queries and the challenge tuple lie in the prescribed spaces, $|m_0| = |m_1|$, and among all share queries with epoch t^*, there is at most one query (S,t^*) and it satisfies $r^* \notin S$.

5 Batched IBE from Laconic Encryption and Inner-Product Functional Encryption

We construct a batched IBE scheme from the laconic encryption scheme of [DKL+23] and a tagged, one-key-per-tag simulation-secure inner-product functional encryption scheme IPFE. Its plaintext and function spaces are $\mathbb{Z}_q^{n+1}$, and its tag space contains the epoch space $\mathcal{T}$. We write its syntax as $\mathsf{IPFE.Setup}$, $\mathsf{IPFE.KeyGen}$, $\mathsf{IPFE.Enc}$, and $\mathsf{IPFE.Dec}$. For a polynomially bounded identity length $k = k(\lambda) \geq 1$, we take $\mathcal{I} = \{0,1\}^k$ and $\mathcal{M} = \{0,1\}$.

Fix a constant $0 < \epsilon < 1/2$ and let q be prime. Let χ be the discrete Gaussian $D_{\mathbb{Z},\sigma}$ conditioned on the integers in $[-B_\chi, B_\chi]$, where $\sigma = n^2$ and $B_\chi = n\sigma$. Let $\bar{\chi}$ be uniform over the integers in $[-B_{\bar{\chi}}, B_{\bar{\chi}}]$. We choose parameters satisfying the following constraints:

- $n = \mathsf{poly}(\lambda)$, $q \leq 2^{n^\epsilon}$, $m = n\lceil \log q \rceil$ (lattice parameters)
- $B_{\bar{\chi}} \geq 2kmB_\chi \cdot 2^{\Omega(\lambda)}$ (for smudging)
- $B_{\bar{\chi}} + 2kmB_\chi + mB_\chi^2 < q/8$ (for correctness)

One possible parameter setting is

$$n = \left\lceil (4\lambda)^{1/\epsilon} \right\rceil, \quad \sigma = n^2, \quad B_\chi = n^3, \quad 2^{3\lambda} \leq q < 2^{3\lambda+1}, \quad m = n\lceil \log q \rceil, \quad B_{\bar{\chi}} = 2^\lambda.$$

For every polynomially bounded k, these choices satisfy the constraints for sufficiently large λ.

The tagged IPFE scheme has plaintext modulus q. Let $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$ be the gadget matrix. Our BIBE scheme is as follows.

$\mathsf{Setup}(1^\lambda)$: It samples $\mathbf{A}_0, \mathbf{A}_1, \mathbf{B} \leftarrow \mathbb{Z}_q^{n\times m}$ and $\mathbf{y}_\perp \leftarrow \mathbb{Z}_q^n$, and sets $\mathbf{A} \coloneqq [\mathbf{A}_0 \mid \mathbf{A}_1]$. It samples $\mathbf{x}^\star \leftarrow \chi^m$, sets $\mathbf{y}^\star \coloneqq \mathbf{B}\mathbf{x}^\star$, and generates

$$(\mathsf{mpk}_{\mathsf{IPFE}}, \mathsf{msk}_{\mathsf{IPFE}}) \leftarrow \mathsf{IPFE.Setup}(1^\lambda).$$

It outputs

$$\mathsf{pp} \coloneqq (\mathbf{A}, \mathbf{B}, \mathbf{G}, \mathbf{y}_\perp, \mathbf{y}^\star, \mathbf{x}^\star, \mathsf{mpk}_{\mathsf{IPFE}}), \qquad \mathsf{msk} \coloneqq \mathsf{msk}_{\mathsf{IPFE}}.$$

$\mathsf{Digest}(\mathsf{pp}, S)$: It takes as input a batch $S = (r_1, \ldots, r_\ell) \subseteq \mathcal{I}$ and outputs $\perp$ if the identities are not distinct. If $S = \emptyset$, it outputs $\mathbf{y}_\perp$. We identify each $r \in \{0,1\}^k$ with a leaf of a depth-k binary tree. The root is ϵ, the children of a node α are $\alpha 0$ and $\alpha 1$, and any child absent from the current tree has label $\mathbf{y}_\perp$. Parse pp, write $\mathbf{A} = [\mathbf{A}_0 \mid \mathbf{A}_1]$, and register $r_1, \ldots, r_\ell$ successively using the following tree-hashing procedure.

Tree hashing. For $\mathbf{y}_0, \mathbf{y}_1 \in \mathbb{Z}_q^n$, define

$$\mathsf{H}_{\mathbf{A}}(\mathbf{y}_0, \mathbf{y}_1) \coloneqq \mathbf{A}_0\big(-\mathbf{G}^{-1}(\mathbf{y}_0)\big) + \mathbf{A}_1\big(-\mathbf{G}^{-1}(\mathbf{y}_1)\big).$$

To register $r_i = b_1 \cdots b_k$, let $\alpha_j = b_1 \cdots b_j$, where $\alpha_0 = \epsilon$ and $\alpha_k = r_i$, and set $\mathbf{y}_{\alpha_k} \leftarrow \mathbf{y}^\star$. Starting with $j = k-1$ and proceeding down to $j = 0$, let $\widetilde{\mathbf{y}}_j$ be the label of the other child of α_j, namely, the child not on the path to r_i. Use $\widetilde{\mathbf{y}}_j = \mathbf{y}_\perp$ if that child is absent. Since $\mathbf{y}_{\alpha_{j+1}}$ is already known, set

$$\mathbf{y}_{\alpha_j} \leftarrow \begin{cases} \mathsf{H}_{\mathbf{A}}(\mathbf{y}_{\alpha_{j+1}}, \widetilde{\mathbf{y}}_j), & b_{j+1} = 0, \\ \mathsf{H}_{\mathbf{A}}(\widetilde{\mathbf{y}}_j, \mathbf{y}_{\alpha_{j+1}}), & b_{j+1} = 1. \end{cases}$$

All other labels remain unchanged. The updated tree has a root label $\mathbf{y}_\epsilon$.

After all registrations, output the root label $\mathbf{y}_S \coloneqq \mathbf{y}_\epsilon$. The algorithm evaluates at most $k\ell$ tree hashes.

$\mathsf{KeyGen}(\mathsf{msk}, \mathbf{y}_S, t)$: Parse msk as $\mathsf{msk}_{\mathsf{IPFE}}$ and set $\mathbf{f}_S \coloneqq (\mathbf{y}_S^\top, 1)^\top \in \mathbb{Z}_q^{n+1}$. It generates

$$\mathsf{sk}_{\mathsf{IPFE}} \leftarrow \mathsf{IPFE.KeyGen}(\mathsf{msk}_{\mathsf{IPFE}}, t, \mathbf{f}_S)$$

and outputs $\mathsf{sk}_{S,t} \coloneqq \mathsf{sk}_{\mathsf{IPFE}}$.

$\mathsf{Enc}(\mathsf{pp}, r, t, \mu)$: Parse $r = b_1 \cdots b_k \in \{0,1\}^k$. It samples $\mathbf{s}_0, \ldots, \mathbf{s}_k \leftarrow \mathbb{Z}_q^n$, $\mathbf{e}_j \leftarrow \chi^{2m}$ for every $j \in \{0, \ldots, k-1\}$, $\mathbf{e}_k \leftarrow \chi^m$, and $\bar{e} \leftarrow \bar{\chi}$. For every $j \in \{0, \ldots, k-1\}$, it sets

$$\mathbf{D}_j \coloneqq \big[(1 - b_{j+1})\mathbf{G} \mid b_{j+1}\mathbf{G}\big], \qquad \mathbf{c}_j^\top \coloneqq \mathbf{s}_j^\top \mathbf{A} + \mathbf{s}_{j+1}^\top \mathbf{D}_j + \mathbf{e}_j^\top,$$

and sets $\mathbf{c}_k^\top \coloneqq \mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top$. It then generates

$$C_{\mathsf{IPFE}} \leftarrow \mathsf{IPFE.Enc}\left(\mathsf{mpk}_{\mathsf{IPFE}}, t, \begin{bmatrix} \mathbf{s}_0 \\ \bar{e} + \lfloor q/2 \rfloor \mu \end{bmatrix}\right)$$

and outputs $\mathsf{ct} \coloneqq (r, t, \mathbf{c}_0, \ldots, \mathbf{c}_k, C_{\mathsf{IPFE}})$.

$\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}_{S,t}, \mathsf{ct}, S, r, t)$: Set $\mathsf{sk}_{\mathsf{IPFE}} \coloneqq \mathsf{sk}_{S,t}$. Parse pp to obtain $\mathbf{x}^\star$ and parse the ciphertext as $\mathsf{ct} = (r', t', \mathbf{c}_0, \ldots, \mathbf{c}_k, C_{\mathsf{IPFE}})$. If $r' \neq r$, $t' \neq t$, or $r \notin S$, it outputs $\perp$. It reconstructs the sparse-tree labels using the tree construction in $\mathsf{Digest}(\mathsf{pp}, S)$. Write $r = b_1 \cdots b_k$, let $\alpha_j = b_1 \cdots b_j$ for $j \in \{0, \ldots, k\}$, where $\alpha_0 = \epsilon$, and set

$$\mathbf{z}_j \coloneqq \begin{bmatrix} -\mathbf{G}^{-1}(\mathbf{y}_{\alpha_j 0}) \\ -\mathbf{G}^{-1}(\mathbf{y}_{\alpha_j 1}) \end{bmatrix} \quad \text{for every } j \in \{0, \ldots, k-1\}.$$

It computes $d \leftarrow \mathsf{IPFE.Dec}(\mathsf{sk}_{\mathsf{IPFE}}, C_{\mathsf{IPFE}})$ and outputs $\perp$ if $d = \perp$. It then sets

$$v \coloneqq d - \sum_{j=0}^{k-1} \mathbf{c}_j^\top \mathbf{z}_j - \mathbf{c}_k^\top \mathbf{x}^\star.$$

Viewing v as an integer in $(-q/2, q/2]$, it outputs 0 if $|v| < q/4$, and outputs 1 otherwise.

Correctness. Let $\mu \in \{0,1\}$ be the encrypted message and let $r \in S$. The hash equations give $\mathbf{A}\mathbf{z}_j = \mathbf{y}_{\alpha_j}$ and $\mathbf{D}_j \mathbf{z}_j = -\mathbf{y}_{\alpha_{j+1}}$ for every $j < k$, with $\mathbf{y}_{\alpha_k} = \mathbf{y}^\star = \mathbf{B}\mathbf{x}^\star$. Thus, whenever IPFE decryption is correct, the secret terms cancel and $v = \lfloor q/2 \rfloor \mu + \bar{e} - \sum_{j=0}^{k-1} \mathbf{e}_j^\top \mathbf{z}_j - \mathbf{e}_k^\top \mathbf{x}^\star \pmod q$. Since $\|\mathbf{z}_j\|_1 \leq 2m$ and χ is B_χ-bounded, the total error satisfies

$$\left| \bar{e} - \sum_{j=0}^{k-1} \mathbf{e}_j^\top \mathbf{z}_j - \mathbf{e}_k^\top \mathbf{x}^\star \right| \leq B_{\bar{\chi}} + 2kmB_\chi + mB_\chi^2 < q/8.$$

Efficiency analysis. Let $b = \lceil \log q \rceil$ and let $\ell_{\mathcal{T}} = \mathsf{poly}(\lambda)$ be the epoch length. Encoding arithmetic components modulo q and omitting the fixed gadget matrix, the bit lengths are $|\mathsf{pp}| = (3nm + 2n + m)b + |\mathsf{mpk}_{\mathsf{IPFE}}|$, $|\mathsf{msk}| = |\mathsf{msk}_{\mathsf{IPFE}}|$, $|\mathbf{y}_S| = nb$, $|\mathsf{sk}_{S,t}| = |\mathsf{sk}_{\mathsf{IPFE}}|$, $|\mathsf{ct}| = k + \ell_{\mathcal{T}} + (2k+1)mb + |C_{\mathsf{IPFE}}|$. Under the above parameter choice, each IPFE component has polynomial bit length in λ. Thus the public parameters, master secret key, digest, and batch key have size $O_\lambda(1)$, and the ciphertext has size $O_\lambda(k)$.

Let $\ell = |S| \geq 1$. Setup and key generation each invoke the corresponding IPFE algorithm and take $O_\lambda(1)$ time. Digest computation evaluates at most ℓk tree hashes and takes $O_\lambda(\ell k)$ time. Encryption computes $k+1$ lattice ciphertext components and performs one IPFE encryption, taking $O_\lambda(k)$ time in total. Decryption reconstructs the sparse tree, computes the k path vectors, and performs one IPFE decryption and $k+1$ inner products, taking $O_\lambda(\ell k)$ time in total.

Reusable public preprocessing. The sparse tree can be computed publicly from (pp, S) once and reused for all ciphertexts in the batch. Each ciphertext then takes $O_\lambda(k)$ time to decrypt, giving $O_\lambda(\ell k)$ total time for ℓ ciphertexts, including public preprocessing.

5.1 Security of the LE–IPFE Construction

Theorem 5.1. Assume that the tagged IPFE scheme is adaptively simulation secure with at most one functional key at the challenge tag and unrestricted functional keys at every non-challenge tag. Assume that the $\mathsf{LWE}_{n,2m+1,q,\chi}$ and $\mathsf{LWE}_{n,m+1,q,\chi}$ assumptions hold and that the error distributions satisfy the smudging condition in Section 5. Then the construction in Section 5 satisfies adaptive security as in Section 4.1.

Sequence of Hybrid Games. We start by describing the hybrid games, with changes from the preceding hybrid marked in red. Hybrid 0 is the original adaptive security experiment. In Hybrid 1, the challenger replaces the tagged IPFE setup, functional keys, and challenge ciphertext with the outputs of the IPFE simulator. For every $i \in \{0, \ldots, k\}$, Hybrid 2.i.0 enters the i-th layer, Hybrid 2.i.1 uses smudging to remove the small error from the function output, and Hybrid 2.i.2 makes $(\mathbf{c}_i, u_i)$ uniform and carries the function output to the next layer. For every $i < k$, Hybrid 2.i.2 is identically distributed to Hybrid 2.$(i+1)$.0. Finally, Hybrid 3 removes the challenge message from the function output.

Hybrid 0: This is the original adaptive security experiment in Section 4.1.

1. The challenger samples $\mathbf{A}_0, \mathbf{A}_1, \mathbf{B} \leftarrow \mathbb{Z}_q^{n \times m}$ and $\mathbf{y}_\perp \leftarrow \mathbb{Z}_q^n$, and sets $\mathbf{A} := [\mathbf{A}_0 \mid \mathbf{A}_1]$. It samples $\mathbf{x}^\star \leftarrow \chi^m$, sets $\mathbf{y}^\star := \mathbf{B}\mathbf{x}^\star$, and generates $(\mathsf{mpk}_{\mathsf{IPFE}}, \mathsf{msk}_{\mathsf{IPFE}}) \leftarrow \mathsf{IPFE.Setup}(1^\lambda)$. It sets $\mathsf{pp} := (\mathbf{A}, \mathbf{B}, \mathbf{G}, \mathbf{y}_\perp, \mathbf{y}^\star, \mathbf{x}^\star, \mathsf{mpk}_{\mathsf{IPFE}})$ and $\mathsf{msk} := \mathsf{msk}_{\mathsf{IPFE}}$. Finally, it samples $\beta \leftarrow \{0,1\}$ and gives pp to $\mathcal{A}$.

2. Before its challenge query, the adversary may make key queries. In each query, $\mathcal{A}$ outputs (S, t), where $S = (r_1, \ldots, r_\ell) \subseteq \mathcal{I}$. If $S = \emptyset$, the challenger sets $\mathbf{y}_S := \mathbf{y}_\perp$. Otherwise, it sets $\mathbf{y}_{r_i} := \mathbf{y}^\star$ for every $r_i \in S$ and evaluates the sparse tree bottom-up. For every internal node α whose subtree contains an identity in S, it uses $\mathbf{y}_\perp$ for an empty child and sets $\mathbf{y}_\alpha := \mathbf{A}_0(-\mathbf{G}^{-1}(\mathbf{y}_{\alpha 0})) + \mathbf{A}_1(-\mathbf{G}^{-1}(\mathbf{y}_{\alpha 1}))$. It sets $\mathbf{y}_S := \mathbf{y}_\epsilon$, $\mathbf{f}_S := (\mathbf{y}_S^\top, 1)^\top$, generates $\mathsf{sk}_{\mathsf{IPFE}} \leftarrow \mathsf{IPFE.KeyGen}(\mathsf{msk}_{\mathsf{IPFE}}, t, \mathbf{f}_S)$, and returns $\mathsf{sk}_{S,t} := \mathsf{sk}_{\mathsf{IPFE}}$.

3. During the challenge query, $\mathcal{A}$ outputs (r^*, t^*, m_0, m_1) and the challenger writes $r^* = b_1 \cdots b_k$. If more than one previous key query has tag t^*, or if the unique such query (S, t^*) satisfies $r^* \in S$, the game aborts and the adversary fails. The challenger samples $\mathbf{s}_0, \ldots, \mathbf{s}_k \leftarrow \mathbb{Z}_q^n$, $\mathbf{e}_j \leftarrow \chi^{2m}$ for every $j \in \{0, \ldots, k-1\}$, $\mathbf{e}_k \leftarrow \chi^m$, and $\bar{e} \leftarrow \bar{\chi}$. For every $j \in \{0, \ldots, k-1\}$, it sets $\mathbf{D}_j := [(1 - b_{j+1})\mathbf{G} \mid b_{j+1}\mathbf{G}]$ and $\mathbf{c}_j^\top := \mathbf{s}_j^\top \mathbf{A} + \mathbf{s}_{j+1}^\top \mathbf{D}_j + \mathbf{e}_j^\top$. It sets $\mathbf{c}_k^\top := \mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top$ and generates
$$C^*_{\mathsf{IPFE}} \leftarrow \mathsf{IPFE.Enc}\left(\mathsf{mpk}_{\mathsf{IPFE}}, t^*, \begin{bmatrix} \mathbf{s}_0 \\ \bar{e} + \lfloor q/2 \rfloor m_\beta \end{bmatrix}\right).$$
It returns $\mathsf{ct}^* := (r^*, t^*, \mathbf{c}_0, \ldots, \mathbf{c}_k, C^*_{\mathsf{IPFE}})$.

4. For each post-challenge key query (S, t), the challenger aborts if $t = t^*$ and either a key with tag t^* has already been queried or $r^* \in S$. Otherwise, it computes $\mathbf{y}_S$ using the above tree equations, sets $\mathbf{f}_S := (\mathbf{y}_S^\top, 1)^\top$, generates $\mathsf{sk}_{\mathsf{IPFE}} \leftarrow \mathsf{IPFE.KeyGen}(\mathsf{msk}_{\mathsf{IPFE}}, t, \mathbf{f}_S)$, and returns $\mathsf{sk}_{S,t} := \mathsf{sk}_{\mathsf{IPFE}}$. The adversary eventually outputs β' and wins if and only if $\beta' = \beta$.

Hybrid 1: It is the same as Hybrid 0, except that the challenger runs the ideal experiment of the tagged IPFE scheme. The lattice parameters, digests, and LWE ciphertext components remain unchanged.

1. The challenger samples $\mathbf{A}_0, \mathbf{A}_1, \mathbf{B} \leftarrow \mathbb{Z}_q^{n \times m}$ and $\mathbf{y}_\perp \leftarrow \mathbb{Z}_q^n$, sets $\mathbf{A} := [\mathbf{A}_0 \mid \mathbf{A}_1]$, samples $\mathbf{x}^\star \leftarrow \chi^m$, and sets $\mathbf{y}^\star := \mathbf{B}\mathbf{x}^\star$. It uses the setup algorithm of the tagged IPFE simulator to generate $\mathsf{mpk}_{\mathsf{IPFE}}$ and its internal state. It sets $\mathsf{pp} := (\mathbf{A}, \mathbf{B}, \mathbf{G}, \mathbf{y}_\perp, \mathbf{y}^\star, \mathbf{x}^\star, \mathsf{mpk}_{\mathsf{IPFE}})$, samples $\beta \leftarrow \{0,1\}$, and gives pp to $\mathcal{A}$.

2. For each pre-challenge key query (S, t), the challenger computes $\mathbf{y}_S$ using the same sparse-tree equations as Hybrid 0 and sets $\mathbf{f}_S := (\mathbf{y}_S^\top, 1)^\top$. It obtains $\mathsf{sk}_{\mathsf{IPFE}}$ from the pre-challenge key-generation algorithm of the tagged IPFE simulator on $(t, \mathbf{f}_S)$ and returns $\mathsf{sk}_{S,t} := \mathsf{sk}_{\mathsf{IPFE}}$.

3. During the challenge query, $\mathcal{A}$ outputs (r^*, t^*, m_0, m_1) and the challenger writes $r^* = b_1 \cdots b_k$. It samples $\mathbf{s}_0, \ldots, \mathbf{s}_k \leftarrow \mathbb{Z}_q^n$, $\mathbf{e}_j \leftarrow \chi^{2m}$ for every $j < k$, $\mathbf{e}_k \leftarrow \chi^m$, and $\bar{e} \leftarrow \bar{\chi}$. It sets $\mathbf{D}_j := [(1-b_{j+1})\mathbf{G} \mid b_{j+1}\mathbf{G}]$ and $\mathbf{c}_j^\top := \mathbf{s}_j^\top \mathbf{A} + \mathbf{s}_{j+1}^\top \mathbf{D}_j + \mathbf{e}_j^\top$ for every $j < k$, and sets $\mathbf{c}_k^\top := \mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top$. If a key for (S, t^*) was queried before the challenge, it sets $d_\beta := \mathbf{s}_0^\top \mathbf{y}_S + \bar{e} + \lfloor q/2 \rfloor m_\beta$ and gives $(\mathbf{f}_S, d_\beta)$ to the tagged IPFE ciphertext simulator. If no key with tag t^* was queried, the ciphertext simulator is given only the IPFE plaintext length. It obtains C^*_{IPFE} and returns $\mathsf{ct}^* := (r^*, t^*, \mathbf{c}_0, \ldots, \mathbf{c}_k, C^*_{\mathsf{IPFE}})$.

4. For a post-challenge query (S, t^*), the challenger computes $\mathbf{y}_S$, sets $\mathbf{f}_S := (\mathbf{y}_S^\top, 1)^\top$, and sets $d_\beta := \mathbf{s}_0^\top \mathbf{y}_S + \bar{e} + \lfloor q/2 \rfloor m_\beta$ and gives $(\mathbf{f}_S, d_\beta)$ to the post-challenge key-generation algorithm of the tagged IPFE simulator. It returns $\mathsf{sk}_{S,t^*} := \mathsf{sk}_{\mathsf{IPFE}}$. Every query with $t \neq t^*$ is answered using the corresponding non-challenge-tag algorithm. All admissibility checks remain the same as Hybrid 0.

Challenge-path notation. Suppose that (S, t^*) is the unique challenge-tag key query. Let $\mathbf{y}_\alpha$ denote the label of node α in the tree for S. Let $\alpha_j = b_1 \cdots b_j$ for $j \in \{0, \ldots, k\}$, where $\alpha_0 = \epsilon$, and let $S_\alpha := \{r \in S : \alpha \text{ is a prefix of } r\}$. Define $h := \min\{j \in \{0, \ldots, k\} : S_{\alpha_j} = \emptyset\}$. Such an h exists since $r^* \notin S$. In particular, $\mathbf{y}_{\alpha_h} = \mathbf{y}_\perp$. For every $j < h$, define $\mathbf{z}_j := [-\mathbf{G}^{-1}(\mathbf{y}_{\alpha_j 0})^\top, -\mathbf{G}^{-1}(\mathbf{y}_{\alpha_j 1})^\top]^\top$. Since α_j is an active node for every $j < h$, these vectors satisfy $\mathbf{A}\mathbf{z}_j = \mathbf{y}_{\alpha_j}$ and $\mathbf{D}_j \mathbf{z}_j = -\mathbf{y}_{\alpha_{j+1}}$. If $S = \emptyset$, then $h = 0$, $\mathbf{y}_S = \mathbf{y}_\perp$, and every sum over $j < h$ is empty. For every $i \in \{0, \ldots, k\}$, we define the following three hybrids.

Hybrid $2.i.0$**:** It is the same as Hybrid 1, except that steps 3 and 4 are changed as follows.

3. During the challenge query, $\mathcal{A}$ outputs (r^*, t^*, m_0, m_1). The challenger aborts and the adversary fails if more than one previous key query has tag t^*, or if the unique such query (S, t^*) satisfies $r^* \in S$. Otherwise, it writes $r^* = b_1 \cdots b_k$. For every $j < i$, it samples $(\mathbf{c}_j, u_j) \leftarrow \mathbb{Z}_q^{2m} \times \mathbb{Z}_q$ uniformly. It samples $\mathbf{s}_i, \ldots, \mathbf{s}_k \leftarrow \mathbb{Z}_q^n$, $\mathbf{e}_j \leftarrow \chi^{2m}$ for every $j \in \{i, \ldots, k-1\}$, $\mathbf{e}_k \leftarrow \chi^m$, $\widehat{e}_i \leftarrow \chi$, and $\bar{e} \leftarrow \bar{\chi}$. It defines $\mathbf{D}_j := [(1 - b_{j+1})\mathbf{G} \mid b_{j+1}\mathbf{G}]$ for every $j < k$ and

$$(\widehat{\mathbf{c}}_i^\top, u_i) := \begin{cases} (\mathbf{s}_i^\top \mathbf{A} + \mathbf{e}_i^\top, \mathbf{s}_i^\top \mathbf{y}_\perp + \widehat{e}_i), & i < k, \\ (\mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top, \mathbf{s}_k^\top \mathbf{y}_\perp + \widehat{e}_k), & i = k. \end{cases}$$

If $i < k$, it sets $\mathbf{c}_i^\top := \widehat{\mathbf{c}}_i^\top + \mathbf{s}_{i+1}^\top \mathbf{D}_i$ and $\mathbf{c}_j^\top := \mathbf{s}_j^\top \mathbf{A} + \mathbf{s}_{j+1}^\top \mathbf{D}_j + \mathbf{e}_j^\top$ for every $j \in \{i+1, \ldots, k-1\}$. It sets $\mathbf{c}_k^\top := \mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top$ if $i < k$, and $\mathbf{c}_k^\top := \widehat{\mathbf{c}}_k^\top$ if $i = k$. If a key for (S, t^*) was queried before the challenge, it computes $\mathbf{y}_S := \mathsf{Digest}(\mathsf{pp}, S)$, h, and the path vectors $\mathbf{z}_0, \ldots, \mathbf{z}_{h-1}$, sets $\mathbf{f}_S := (\mathbf{y}_S^\top, 1)^\top$, and invokes the tagged IPFE ciphertext simulator on t^*, the IPFE plaintext length, and the singleton list containing $(\mathbf{f}_S, d_\beta)$, where

$$d_\beta := \begin{cases} \displaystyle\sum_{j=0}^{h-1} \mathbf{c}_j^\top \mathbf{z}_j + u_h + \bar{e} + \lfloor q/2 \rfloor m_\beta, & h < i, \\ \displaystyle\sum_{j=0}^{i-1} \mathbf{c}_j^\top \mathbf{z}_j + (u_i - \widehat{e}_i) + \bar{e} + \lfloor q/2 \rfloor m_\beta, & h = i, \\ \displaystyle\sum_{j=0}^{i-1} \mathbf{c}_j^\top \mathbf{z}_j + (\widehat{\mathbf{c}}_i^\top - \mathbf{e}_i^\top)\mathbf{z}_i + \bar{e} + \lfloor q/2 \rfloor m_\beta, & h > i. \end{cases}$$

If no key with tag t^* was queried, it invokes the same simulator on t^*, the IPFE plaintext length, and the empty list. It obtains C^*_{IPFE}, retains its state, and returns $\mathsf{ct}^* := (r^*, t^*, \mathbf{c}_0, \ldots, \mathbf{c}_k, C^*_{\mathsf{IPFE}})$ to $\mathcal{A}$.

4. Post-challenge key queries are answered as in Hybrid 1, except that for (S, t^*), the challenger computes d_β by the formula in step 3 of this hybrid, using the path for S and r^* and the stored challenge values, and gives $(\mathbf{f}_S, d_\beta)$ to the tagged IPFE key simulator. All admissibility checks and the winning condition remain unchanged.

Hybrid $2.i.1$**:** It is the same as Hybrid $2.i.0$, except that the challenger uses smudging to remove the small error from the function output.

3. During the challenge query, $\mathcal{A}$ outputs (r^*, t^*, m_0, m_1). The challenger aborts and the adversary fails if more than one previous key query has tag t^*, or if the unique such query (S, t^*) satisfies $r^* \in S$. Otherwise, it writes $r^* = b_1 \cdots b_k$. For every $j < i$, it samples $(\mathbf{c}_j, u_j) \leftarrow \mathbb{Z}_q^{2m} \times \mathbb{Z}_q$ uniformly. It samples $\mathbf{s}_i, \ldots, \mathbf{s}_k \leftarrow \mathbb{Z}_q^n$, $\mathbf{e}_j \leftarrow \chi^{2m}$ for every $j \in \{i, \ldots, k-1\}$, $\mathbf{e}_k \leftarrow \chi^m$, $\widehat{e}_i \leftarrow \chi$, and $\bar{e} \leftarrow \bar{\chi}$. It defines $\mathbf{D}_j := [(1 - b_{j+1})\mathbf{G} \mid b_{j+1}\mathbf{G}]$ for every $j < k$ and

$$(\widehat{\mathbf{c}}_i^\top, u_i) := \begin{cases} (\mathbf{s}_i^\top \mathbf{A} + \mathbf{e}_i^\top, \mathbf{s}_i^\top \mathbf{y}_\perp + \widehat{e}_i), & i < k, \\ (\mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top, \mathbf{s}_k^\top \mathbf{y}_\perp + \widehat{e}_k), & i = k. \end{cases}$$

If $i < k$, it sets $\mathbf{c}_i^\top := \widehat{\mathbf{c}}_i^\top + \mathbf{s}_{i+1}^\top \mathbf{D}_i$ and $\mathbf{c}_j^\top := \mathbf{s}_j^\top \mathbf{A} + \mathbf{s}_{j+1}^\top \mathbf{D}_j + \mathbf{e}_j^\top$ for every $j \in \{i+1, \ldots, k-1\}$. It sets $\mathbf{c}_k^\top := \mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top$ if $i < k$, and $\mathbf{c}_k^\top := \widehat{\mathbf{c}}_k^\top$ if $i = k$. If a key for (S, t^*) was queried before the challenge, it computes $\mathbf{y}_S := \mathsf{Digest}(\mathsf{pp}, S)$, h, and the path vectors $\mathbf{z}_0, \ldots, \mathbf{z}_{h-1}$, sets $\mathbf{f}_S := (\mathbf{y}_S^\top, 1)^\top$, and invokes the tagged IPFE ciphertext simulator on t^*, the IPFE plaintext length, and the singleton list containing $(\mathbf{f}_S, d_\beta)$, where

$$d_\beta := \begin{cases} \displaystyle\sum_{j=0}^{h-1} \mathbf{c}_j^\top \mathbf{z}_j + u_h + \bar{e} + \lfloor q/2 \rfloor m_\beta, & h < i, \\ \displaystyle\sum_{j=0}^{i-1} \mathbf{c}_j^\top \mathbf{z}_j + u_i + \bar{e} + \lfloor q/2 \rfloor m_\beta, & h = i, \\ \displaystyle\sum_{j=0}^{i-1} \mathbf{c}_j^\top \mathbf{z}_j + \widehat{\mathbf{c}}_i^\top \mathbf{z}_i + \bar{e} + \lfloor q/2 \rfloor m_\beta, & h > i. \end{cases}$$

If no key with tag t^* was queried, it invokes the same simulator on t^*, the IPFE plaintext length, and the empty list. It obtains C^*_{IPFE}, retains its state, and returns $\mathsf{ct}^* := (r^*, t^*, \mathbf{c}_0, \ldots, \mathbf{c}_k, C^*_{\mathsf{IPFE}})$ to $\mathcal{A}$.

4. Post-challenge key queries are answered as in Hybrid 1, except that for (S, t^*), the challenger computes d_β by the formula in step 3 of this hybrid, using the path for S and r^* and the stored challenge values, and gives $(\mathbf{f}_S, d_\beta)$ to the tagged IPFE key simulator. All admissibility checks and the winning condition remain unchanged.

Hybrid $2.i.2$**:** It is the same as Hybrid $2.i.1$, except that the challenger samples the actual ciphertext component $\mathbf{c}_i$ together with u_i uniformly and carries the function output to the next layer. For every $i < k$, Hybrid $2.i.2$ is identically distributed to Hybrid $2.(i+1).0$.

3. During the challenge query, $\mathcal{A}$ outputs (r^*, t^*, m_0, m_1). The challenger aborts and the adversary fails if more than one previous key query has tag t^*, or if the unique such query (S, t^*) satisfies $r^* \in S$. Otherwise, it writes $r^* = b_1 \cdots b_k$. For every $j < i$, it samples $(\mathbf{c}_j, u_j) \leftarrow \mathbb{Z}_q^{2m} \times \mathbb{Z}_q$ uniformly. If $i < k$, it samples $(\mathbf{c}_i, u_i) \leftarrow \mathbb{Z}_q^{2m} \times \mathbb{Z}_q$ uniformly. If $i = k$, it samples $(\mathbf{c}_k, u_k) \leftarrow \mathbb{Z}_q^{m} \times \mathbb{Z}_q$ uniformly. It samples $\bar{e} \leftarrow \bar{\chi}$. If $i < k$, it samples $\mathbf{s}_{i+1}, \ldots, \mathbf{s}_k \leftarrow \mathbb{Z}_q^n$, $\mathbf{e}_j \leftarrow \chi^{2m}$ for every $i < j < k$, and $\mathbf{e}_k \leftarrow \chi^m$, sets $\mathbf{D}_j := [(1-b_{j+1})\mathbf{G} \mid b_{j+1}\mathbf{G}]$ and $\mathbf{c}_j^\top := \mathbf{s}_j^\top \mathbf{A} + \mathbf{s}_{j+1}^\top \mathbf{D}_j + \mathbf{e}_j^\top$ for every $i < j < k$, and sets $\mathbf{c}_k^\top := \mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top$. If a key for (S, t^*) was queried before the challenge, it computes $\mathbf{y}_S := \mathsf{Digest}(\mathsf{pp}, S)$, h, and the path vectors $\mathbf{z}_0, \ldots, \mathbf{z}_{h-1}$, sets $\mathbf{f}_S := (\mathbf{y}_S^\top, 1)^\top$, and invokes the tagged IPFE ciphertext simulator on t^*, the IPFE plaintext length, and the singleton list containing $(\mathbf{f}_S, d_\beta)$, where

$$d_\beta := \begin{cases} \sum_{j=0}^{h-1} \mathbf{c}_j^\top \mathbf{z}_j + u_h + \bar{e} + \lfloor q/2 \rfloor m_\beta, & h \le i, \\ \sum_{j=0}^{i} \mathbf{c}_j^\top \mathbf{z}_j + \mathbf{s}_{i+1}^\top \mathbf{y}_{\alpha_{i+1}} + \bar{e} + \lfloor q/2 \rfloor m_\beta, & h > i. \end{cases}$$

   If no key with tag t^* was queried, it invokes the same simulator on t^*, the IPFE plaintext length, and the empty list. It obtains C^*_{IPFE}, retains its state, and returns $\mathsf{ct}^* := (r^*, t^*, \mathbf{c}_0, \ldots, \mathbf{c}_k, C^*_{\mathsf{IPFE}})$ to $\mathcal{A}$.

4. Post-challenge key queries are answered as in Hybrid 1, except that for (S, t^*), the challenger computes d_β by the formula in step 3 of this hybrid, using the path for S and r^* and the stored challenge values, and gives $(\mathbf{f}_S, d_\beta)$ to the tagged IPFE key simulator. All admissibility checks and the winning condition remain unchanged.

Hybrid 3: It is the same as Hybrid 2.k.2, except that the challenger removes the challenge message from the function output supplied to the tagged IPFE simulator.

3. If (S, t^*) was queried before the challenge, the tagged IPFE ciphertext simulator is given $\mathbf{f}_S$ and $d := \sum_{j=0}^{h-1} \mathbf{c}_j^\top \mathbf{z}_j + u_h + \bar{e}$. If no key with tag t^* was queried, the ciphertext simulator is given only the IPFE plaintext length.

4. If (S, t^*) is queried after the challenge, the post-challenge key-generation algorithm is given $\mathbf{f}_S$ and $d := \sum_{j=0}^{h-1} \mathbf{c}_j^\top \mathbf{z}_j + u_h + \bar{e}$.

5.1.1 Indistinguishability of Hybrids

For any admissible adversary $\mathcal{A}$, let $\mathsf{Adv}^i_{\mathcal{A}}$ denote the probability that $\mathcal{A}$ wins in Hybrid i.

Lemma 5.2. Assume that the tagged IPFE scheme satisfies 1-adaptive simulation security as in Definition 3.6. Then, for every admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $|\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}}| \le \mathsf{negl}(\lambda)$.

Proof. Assume that there exists an admissible PPT adversary $\mathcal{A}$ such that $|\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}}| \ge \epsilon(\lambda)$ for some non-negligible function $\epsilon(\cdot)$. We construct a reduction algorithm $\mathcal{B}$ against the adaptive simulation security of the tagged IPFE scheme.

The tagged IPFE challenger begins by giving $\mathsf{mpk}_{\mathsf{IPFE}}$ to $\mathcal{B}$. The reduction samples $\mathbf{A}_0, \mathbf{A}_1, \mathbf{B} \leftarrow \mathbb{Z}_q^{n \times m}$ and $\mathbf{y}_\perp \leftarrow \mathbb{Z}_q^n$, sets $\mathbf{A} := [\mathbf{A}_0 \mid \mathbf{A}_1]$, samples $\mathbf{x}^\star \leftarrow \chi^m$, and sets $\mathbf{y}^\star := \mathbf{B}\mathbf{x}^\star$. It samples $\beta \leftarrow \{0, 1\}$, sets $\mathsf{pp} := (\mathbf{A}, \mathbf{B}, \mathbf{G}, \mathbf{y}_\perp, \mathbf{y}^\star, \mathbf{x}^\star, \mathsf{mpk}_{\mathsf{IPFE}})$, and gives pp to $\mathcal{A}$. For every pre-challenge

key query (S,t), $\mathcal{B}$ computes $\mathbf{y}_S \coloneqq \mathsf{Digest}(\mathsf{pp}, S)$ and sets $\mathbf{f}_S \coloneqq (\mathbf{y}_S^\top, 1)^\top$. It submits $(t, \mathbf{f}_S)$ as a functional-key query to the tagged IPFE challenger. Upon receiving $\mathsf{sk}_{\mathsf{IPFE}}$, it returns $\mathsf{sk}_{S,t} \coloneqq \mathsf{sk}_{\mathsf{IPFE}}$ to $\mathcal{A}$ and records the query. During the challenge query, $\mathcal{A}$ outputs (r^*, t^*, m_0, m_1). The reduction aborts and outputs 0 if more than one previous key query has tag t^*, or if the unique such query (S, t^*) satisfies $r^* \in S$. Otherwise, it writes $r^* = b_1 \cdots b_k$ and samples $\mathbf{s}_0, \ldots, \mathbf{s}_k \leftarrow \mathbb{Z}_q^n$, $\mathbf{e}_j \leftarrow \chi^{2m}$ for every $j < k$, $\mathbf{e}_k \leftarrow \chi^m$, and $\bar{e} \leftarrow \bar{\chi}$. For every $j < k$, it sets $\mathbf{D}_j \coloneqq [(1 - b_{j+1})\mathbf{G} \mid b_{j+1}\mathbf{G}]$ and $\mathbf{c}_j^\top \coloneqq \mathbf{s}_j^\top \mathbf{A} + \mathbf{s}_{j+1}^\top \mathbf{D}_j + \mathbf{e}_j^\top$. It sets $\mathbf{c}_k^\top \coloneqq \mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top$ and submits $(t^*, [\mathbf{s}_0^\top, \bar{e} + \lfloor q/2 \rfloor m_\beta]^\top)$ as its tagged IPFE challenge. Upon receiving C^*_{IPFE}, it returns $\mathsf{ct}^* \coloneqq (r^*, t^*, \mathbf{c}_0, \ldots, \mathbf{c}_k, C^*_{\mathsf{IPFE}})$ to $\mathcal{A}$. For every post-challenge key query (S,t), $\mathcal{B}$ aborts and outputs 0 if $t = t^*$ and either a key with tag t^* has already been queried or $r^* \in S$. Otherwise, it computes $\mathbf{y}_S \coloneqq \mathsf{Digest}(\mathsf{pp}, S)$, sets $\mathbf{f}_S \coloneqq (\mathbf{y}_S^\top, 1)^\top$, and submits $(t, \mathbf{f}_S)$ to the tagged IPFE challenger. Upon receiving $\mathsf{sk}_{\mathsf{IPFE}}$, it returns $\mathsf{sk}_{S,t} \coloneqq \mathsf{sk}_{\mathsf{IPFE}}$ to $\mathcal{A}$ and records the query. When $\mathcal{A}$ outputs β', the reduction outputs 1 if and only if $\beta' = \beta$.

If the tagged IPFE challenger runs the real experiment, then $\mathsf{mpk}_{\mathsf{IPFE}}$, every functional key, and C^*_{IPFE} are generated by the real tagged IPFE algorithms. Therefore, the view of $\mathcal{A}$ is identically distributed to Hybrid 0. Suppose that the tagged IPFE challenger runs the ideal experiment. If a key for (S, t^*) was queried before the challenge, the tagged IPFE ciphertext simulator is given t^*, the prescribed IPFE plaintext length, and the singleton function-value list containing $(\mathbf{f}_S, d_\beta)$, where $d_\beta = \mathbf{f}_S^\top \begin{bmatrix} \mathbf{s}_0 \\ \bar{e} + \lfloor q/2 \rfloor m_\beta \end{bmatrix} = \mathbf{s}_0^\top \mathbf{y}_S + \bar{e} + \lfloor q/2 \rfloor m_\beta$. If no key with tag t^* was queried before the challenge, it is given t^*, the prescribed IPFE plaintext length, and the empty list. For every post-challenge query (S, t^*), the tagged IPFE key simulator is given $(t^*, \mathbf{f}_S, d_\beta)$, where $d_\beta = \mathbf{s}_0^\top \mathbf{y}_S + \bar{e} + \lfloor q/2 \rfloor m_\beta$. Every query with tag different from t^* is answered by the corresponding non-challenge-tag simulator algorithm without a function output. Thus the view of $\mathcal{A}$ is identically distributed to Hybrid 1. The admissibility of $\mathcal{A}$ guarantees that $\mathcal{B}$ makes at most one functional-key query at the challenge tag, while an arbitrary polynomial number of queries may be made at every other tag. Consequently, the distinguishing advantage of $\mathcal{B}$ is $|\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}}|$, which contradicts the adaptive simulation security of the tagged IPFE scheme. □

Lemma 5.3. For every admissible adversary $\mathcal{A}$ of unbounded running time, $\mathsf{Adv}^1_{\mathcal{A}} = \mathsf{Adv}^{2.0.0}_{\mathcal{A}}$.

Proof. Hybrid 2.0.0 samples no uniform pair. If $h = 0$, then $\mathbf{y}_S = \mathbf{y}_\perp$ and $u_0 - \widehat{e}_0 = \mathbf{s}_0^\top \mathbf{y}_S$. If $h > 0$, then $\mathbf{A}\mathbf{z}_0 = \mathbf{y}_S$ and $(\widehat{\mathbf{c}}_0^\top - \mathbf{e}_0^\top)\mathbf{z}_0 = \mathbf{s}_0^\top \mathbf{y}_S$. Thus its function output is exactly $\mathbf{s}_0^\top \mathbf{y}_S + \bar{e} + \lfloor q/2 \rfloor m_\beta$, as in Hybrid 1. □

Lemma 5.4. For every $i \in \{0, \ldots, k\}$ and every admissible adversary $\mathcal{A}$ of unbounded running time, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $|\mathsf{Adv}^{2.i.0}_{\mathcal{A}} - \mathsf{Adv}^{2.i.1}_{\mathcal{A}}| \le \mathsf{negl}(\lambda)$.

Proof. If $h < i$, the function outputs in the two hybrids are identical. If $h = i$, Hybrid $2.i.0$ uses the error $\bar{e} - \widehat{e}_i$, whereas Hybrid $2.i.1$ uses $\bar{e}$. If $h > i$, Hybrid $2.i.0$ uses the error $\bar{e} - \mathbf{e}_i^\top \mathbf{z}_i$, whereas Hybrid $2.i.1$ uses $\bar{e}$. The canonical decomposition gives $\|\mathbf{z}_i\|_1 \le 2m$, so $|\mathbf{e}_i^\top \mathbf{z}_i| \le 2mB_\chi$, while $|\widehat{e}_i| \le B_\chi$. The claim follows from Lemma 3.4. □

Lemma 5.5. Assume that the $\mathsf{LWE}_{n,2m+1,q,\chi}$ and $\mathsf{LWE}_{n,m+1,q,\chi}$ assumptions hold. Then, for every $i \in \{0, \ldots, k\}$ and every admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $|\mathsf{Adv}^{2.i.1}_{\mathcal{A}} - \mathsf{Adv}^{2.i.2}_{\mathcal{A}}| \le \mathsf{negl}(\lambda)$.

Proof. First suppose that $i < k$. Assume that there exists an admissible PPT adversary $\mathcal{A}$ such that $|\mathsf{Adv}^{2.i.1}_{\mathcal{A}} - \mathsf{Adv}^{2.i.2}_{\mathcal{A}}| \ge \epsilon(\lambda)$ for some non-negligible function $\epsilon(\cdot)$. We construct a reduction algorithm $\mathcal{B}$ against the $\mathsf{LWE}_{n,2m+1,q,\chi}$ assumption.

The LWE challenger begins by giving $\mathcal{B}$ a matrix $\mathbf{Q} \in \mathbb{Z}_q^{n\times(2m+1)}$ and a vector $(\mathbf{v}^\top, v) \in \mathbb{Z}_q^{2m} \times \mathbb{Z}_q$. The reduction parses $\mathbf{Q} = [\mathbf{A} \mid \mathbf{y}_\perp]$ and $\mathbf{A} = [\mathbf{A}_0 \mid \mathbf{A}_1]$. It samples $\mathbf{B} \leftarrow \mathbb{Z}_q^{n\times m}$ and $\mathbf{x}^\star \leftarrow \chi^m$, sets $\mathbf{y}^\star \coloneqq \mathbf{B}\mathbf{x}^\star$, and runs the setup algorithm of the tagged IPFE simulator to obtain $\mathsf{mpk}_{\mathsf{IPFE}}$ and its internal state. It samples $\beta \leftarrow \{0,1\}$, sets $\mathsf{pp} \coloneqq (\mathbf{A}, \mathbf{B}, \mathbf{G}, \mathbf{y}_\perp, \mathbf{y}^\star, \mathbf{x}^\star, \mathsf{mpk}_{\mathsf{IPFE}})$, and gives pp to $\mathcal{A}$. For every pre-challenge key query (S,t), $\mathcal{B}$ computes $\mathbf{y}_S \coloneqq \mathsf{Digest}(\mathsf{pp}, S)$ and sets $\mathbf{f}_S \coloneqq (\mathbf{y}_S^\top, 1)^\top$. It obtains $\mathsf{sk}_{\mathsf{IPFE}}$ by invoking the pre-challenge key-generation algorithm of the tagged IPFE simulator on $(t, \mathbf{f}_S)$, returns $\mathsf{sk}_{S,t} \coloneqq \mathsf{sk}_{\mathsf{IPFE}}$ to $\mathcal{A}$, and records the query. During the challenge query, $\mathcal{A}$ outputs (r^*, t^*, m_0, m_1). The reduction aborts and outputs 0 if more than one previous key query has tag t^*, or if the unique such query (S, t^*) satisfies $r^* \in S$. Otherwise, it writes $r^* = b_1 \cdots b_k$ and sets $\mathbf{D}_j \coloneqq [(1-b_{j+1})\mathbf{G} \mid b_{j+1}\mathbf{G}]$ for every $j < k$. For every $j < i$, it samples $(\mathbf{c}_j, u_j) \leftarrow \mathbb{Z}_q^{2m} \times \mathbb{Z}_q$ uniformly. It samples $\mathbf{s}_{i+1}, \ldots, \mathbf{s}_k \leftarrow \mathbb{Z}_q^n$, $\mathbf{e}_j \leftarrow \chi^{2m}$ for every $j \in \{i+1, \ldots, k-1\}$, $\mathbf{e}_k \leftarrow \chi^m$, and $\bar{e} \leftarrow \bar{\chi}$. It sets $u_i \coloneqq v, \mathbf{c}_i^\top \coloneqq \mathbf{v}^\top + \mathbf{s}_{i+1}^\top \mathbf{D}_i$, sets $\mathbf{c}_j^\top \coloneqq \mathbf{s}_j^\top \mathbf{A} + \mathbf{s}_{j+1}^\top \mathbf{D}_j + \mathbf{e}_j^\top$ for every $j \in \{i+1, \ldots, k-1\}$, and sets $\mathbf{c}_k^\top \coloneqq \mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top$. If a key for (S, t^*) was queried before the challenge, $\mathcal{B}$ computes h and $\mathbf{z}_0, \ldots, \mathbf{z}_{h-1}$ from S and r^*. It sets

$$d_\beta \coloneqq \begin{cases} \displaystyle\sum_{j=0}^{h-1} \mathbf{c}_j^\top \mathbf{z}_j + u_h + \bar{e} + \lfloor q/2 \rfloor m_\beta, & h < i, \\ \displaystyle\sum_{j=0}^{i-1} \mathbf{c}_j^\top \mathbf{z}_j + u_i + \bar{e} + \lfloor q/2 \rfloor m_\beta, & h = i, \\ \displaystyle\sum_{j=0}^{i-1} \mathbf{c}_j^\top \mathbf{z}_j + \mathbf{v}^\top \mathbf{z}_i + \bar{e} + \lfloor q/2 \rfloor m_\beta, & h > i \end{cases}$$

and invokes the tagged IPFE ciphertext simulator on t^*, the prescribed IPFE plaintext length, and the singleton function-value list containing $(\mathbf{f}_S, d_\beta)$. If no key with tag t^* was queried before the challenge, it invokes the ciphertext simulator on t^*, the prescribed IPFE plaintext length, and the empty function-value list. It obtains C^*_{IPFE} and returns $\mathsf{ct}^* \coloneqq (r^*, t^*, \mathbf{c}_0, \ldots, \mathbf{c}_k, C^*_{\mathsf{IPFE}})$ to $\mathcal{A}$. For every post-challenge key query (S,t), $\mathcal{B}$ aborts and outputs 0 if $t = t^*$ and either a key with tag t^* has already been queried or $r^* \in S$. Otherwise, it computes $\mathbf{y}_S \coloneqq \mathsf{Digest}(\mathsf{pp}, S)$ and sets $\mathbf{f}_S \coloneqq (\mathbf{y}_S^\top, 1)^\top$. If $t \neq t^*$, it invokes the corresponding non-challenge-tag key-generation algorithm of the tagged IPFE simulator on $(t, \mathbf{f}_S)$ and returns the resulting $\mathsf{sk}_{S,t} \coloneqq \mathsf{sk}_{\mathsf{IPFE}}$ to $\mathcal{A}$. If $t = t^*$, it computes h and $\mathbf{z}_0, \ldots, \mathbf{z}_{h-1}$ from S and r^*, and sets

$$d_\beta \coloneqq \begin{cases} \displaystyle\sum_{j=0}^{h-1} \mathbf{c}_j^\top \mathbf{z}_j + u_h + \bar{e} + \lfloor q/2 \rfloor m_\beta, & h < i, \\ \displaystyle\sum_{j=0}^{i-1} \mathbf{c}_j^\top \mathbf{z}_j + u_i + \bar{e} + \lfloor q/2 \rfloor m_\beta, & h = i, . \\ \displaystyle\sum_{j=0}^{i-1} \mathbf{c}_j^\top \mathbf{z}_j + \mathbf{v}^\top \mathbf{z}_i + \bar{e} + \lfloor q/2 \rfloor m_\beta, & h > i \end{cases}$$

It invokes the post-challenge key-generation algorithm of the tagged IPFE simulator on $(t^*, \mathbf{f}_S, d_\beta)$ and returns the resulting $\mathsf{sk}_{S,t^*} \coloneqq \mathsf{sk}_{\mathsf{IPFE}}$ to $\mathcal{A}$. The reduction records every answered post-challenge query. The adversary eventually outputs β'. The reduction outputs 1 if and only if $\beta' = \beta$.

If the LWE challenger samples $\mathbf{s}_i \leftarrow \mathbb{Z}_q^n$, $\mathbf{e}_i \leftarrow \chi^{2m}$, and $\widehat{e}_i \leftarrow \chi$ and sets $(\mathbf{v}^\top, v) = (\mathbf{s}_i^\top \mathbf{A} + \mathbf{e}_i^\top, \mathbf{s}_i^\top \mathbf{y}_\perp + \widehat{e}_i)$, then $u_i = \mathbf{s}_i^\top \mathbf{y}_\perp + \widehat{e}_i$ and $\mathbf{c}_i^\top = \mathbf{s}_i^\top \mathbf{A} + \mathbf{s}_{i+1}^\top \mathbf{D}_i + \mathbf{e}_i^\top$. The function output uses u_h when $h < i$, uses u_i when $h = i$, and uses $(\mathbf{s}_i^\top \mathbf{A} + \mathbf{e}_i^\top)\mathbf{z}_i$ when $h > i$. Therefore, the entire experiment is

identically distributed to Hybrid $2.i.1$. Otherwise, $(\mathbf{v}^\top, v)$ is uniformly distributed over $\mathbb{Z}_q^{2m} \times \mathbb{Z}_q$. For every fixed $\mathbf{s}_{i+1}$, translation by $\mathbf{s}_{i+1}^\top \mathbf{D}_i$ preserves uniformity. Hence $(\mathbf{c}_i, u_i)$ is uniform and independent of $\mathbf{s}_{i+1}$ and all later ciphertext components. When $h \leq i$, the function output is already the one used in Hybrid $2.i.2$. When $h > i$, the equation $\mathbf{D}_i \mathbf{z}_i = -\mathbf{y}_{\alpha_{i+1}}$ gives $\mathbf{v}^\top \mathbf{z}_i = (\mathbf{c}_i^\top - \mathbf{s}_{i+1}^\top \mathbf{D}_i)\mathbf{z}_i = \mathbf{c}_i^\top \mathbf{z}_i + \mathbf{s}_{i+1}^\top \mathbf{y}_{\alpha_{i+1}}$. Thus, in this case, $d_\beta = \sum_{j=0}^{i} \mathbf{c}_j^\top \mathbf{z}_j + \mathbf{s}_{i+1}^\top \mathbf{y}_{\alpha_{i+1}} + \bar{e} + \lfloor q/2 \rfloor m_\beta$, which is the function output used in Hybrid $2.i.2$. If the challenge-tag key is queried after the challenge, $\mathcal{B}$ computes the same value from the stored LWE vector when it answers that query. The adaptive choice of S, h, and the path vectors is efficient post-processing of this stored vector, so $\mathcal{B}$ does not guess them before receiving its LWE challenge. Hence the entire experiment is identically distributed to Hybrid $2.i.2$. It follows that $\mathcal{B}$ distinguishes the two LWE distributions with advantage $|\mathsf{Adv}_{\mathcal{A}}^{2.i.1} - \mathsf{Adv}_{\mathcal{A}}^{2.i.2}|$, contradicting the $\mathsf{LWE}_{n,2m+1,q,\chi}$ assumption.

Now suppose that $i = k$. Assume that there exists an admissible PPT adversary $\mathcal{A}$ such that $|\mathsf{Adv}_{\mathcal{A}}^{2.k.1} - \mathsf{Adv}_{\mathcal{A}}^{2.k.2}| \geq \epsilon(\lambda)$ for some non-negligible function $\epsilon(\cdot)$. We construct a reduction algorithm $\mathcal{B}$ against the $\mathsf{LWE}_{n,m+1,q,\chi}$ assumption.

The LWE challenger begins by giving $\mathcal{B}$ a matrix $\mathbf{Q} \in \mathbb{Z}_q^{n \times (m+1)}$ and a vector $(\mathbf{v}^\top, v) \in \mathbb{Z}_q^m \times \mathbb{Z}_q$. The reduction parses $\mathbf{Q} = [\mathbf{B} \mid \mathbf{y}_\perp]$. It samples $\mathbf{A}_0, \mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n \times m}$ and $\mathbf{x}^\star \leftarrow \chi^m$, sets $\mathbf{A} \coloneqq [\mathbf{A}_0 \mid \mathbf{A}_1]$ and $\mathbf{y}^\star \coloneqq \mathbf{B}\mathbf{x}^\star$, and runs the setup algorithm of the tagged IPFE simulator to obtain $\mathsf{mpk}_{\mathsf{IPFE}}$ and its internal state. It samples $\beta \leftarrow \{0,1\}$, sets $\mathsf{pp} \coloneqq (\mathbf{A}, \mathbf{B}, \mathbf{G}, \mathbf{y}_\perp, \mathbf{y}^\star, \mathbf{x}^\star, \mathsf{mpk}_{\mathsf{IPFE}})$, and gives pp to $\mathcal{A}$. For every pre-challenge key query (S, t), $\mathcal{B}$ computes $\mathbf{y}_S \coloneqq \mathsf{Digest}(\mathsf{pp}, S)$ and sets $\mathbf{f}_S \coloneqq (\mathbf{y}_S^\top, 1)^\top$. It obtains $\mathsf{sk}_{\mathsf{IPFE}}$ by invoking the pre-challenge key-generation algorithm of the tagged IPFE simulator on $(t, \mathbf{f}_S)$, returns $\mathsf{sk}_{S,t} \coloneqq \mathsf{sk}_{\mathsf{IPFE}}$ to $\mathcal{A}$, and records the query. During the challenge query, $\mathcal{A}$ outputs (r^*, t^*, m_0, m_1). The reduction aborts and outputs 0 if more than one previous key query has tag t^*, or if the unique such query (S, t^*) satisfies $r^* \in S$. Otherwise, it writes $r^* = b_1 \cdots b_k$ and sets $\mathbf{D}_j \coloneqq [(1 - b_{j+1})\mathbf{G} \mid b_{j+1}\mathbf{G}]$ for every $j < k$. It samples $(\mathbf{c}_j, u_j) \leftarrow \mathbb{Z}_q^{2m} \times \mathbb{Z}_q$ uniformly for every $j < k$ and $\bar{e} \leftarrow \bar{\chi}$, and sets $(\mathbf{c}_k^\top, u_k) \coloneqq (\mathbf{v}^\top, v)$. If a key for (S, t^*) was queried before the challenge, $\mathcal{B}$ computes h and $\mathbf{z}_0, \ldots, \mathbf{z}_{h-1}$ from S and r^*. It sets $d_\beta \coloneqq \sum_{j=0}^{h-1} \mathbf{c}_j^\top \mathbf{z}_j + u_h + \bar{e} + \lfloor q/2 \rfloor m_\beta$ and invokes the tagged IPFE ciphertext simulator on t^*, the prescribed IPFE plaintext length, and the singleton function-value list containing $(\mathbf{f}_S, d_\beta)$. If no key with tag t^* was queried before the challenge, it invokes the ciphertext simulator on t^*, the prescribed IPFE plaintext length, and the empty function-value list. It obtains C^*_{IPFE} and returns $\mathsf{ct}^* \coloneqq (r^*, t^*, \mathbf{c}_0, \ldots, \mathbf{c}_k, C^*_{\mathsf{IPFE}})$ to $\mathcal{A}$. For every post-challenge key query (S, t), $\mathcal{B}$ aborts and outputs 0 if $t = t^*$ and either a key with tag t^* has already been queried or $r^* \in S$. Otherwise, it computes $\mathbf{y}_S \coloneqq \mathsf{Digest}(\mathsf{pp}, S)$ and sets $\mathbf{f}_S \coloneqq (\mathbf{y}_S^\top, 1)^\top$. If $t \neq t^*$, it invokes the corresponding non-challenge-tag key-generation algorithm of the tagged IPFE simulator on $(t, \mathbf{f}_S)$ and returns the resulting $\mathsf{sk}_{S,t} \coloneqq \mathsf{sk}_{\mathsf{IPFE}}$ to $\mathcal{A}$. If $t = t^*$, it computes h and $\mathbf{z}_0, \ldots, \mathbf{z}_{h-1}$ from S and r^*, and sets $d_\beta \coloneqq \sum_{j=0}^{h-1} \mathbf{c}_j^\top \mathbf{z}_j + u_h + \bar{e} + \lfloor q/2 \rfloor m_\beta$. It invokes the post-challenge key-generation algorithm of the tagged IPFE simulator on $(t^*, \mathbf{f}_S, d_\beta)$ and returns the resulting $\mathsf{sk}_{S,t^*} \coloneqq \mathsf{sk}_{\mathsf{IPFE}}$ to $\mathcal{A}$. The reduction records every answered post-challenge query. The adversary eventually outputs β'. The reduction outputs 1 if and only if $\beta' = \beta$.

If the LWE challenger samples $\mathbf{s}_k \leftarrow \mathbb{Z}_q^n$, $\mathbf{e}_k \leftarrow \chi^m$, and $\widehat{e}_k \leftarrow \chi$ and sets $(\mathbf{v}^\top, v) = (\mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top, \mathbf{s}_k^\top \mathbf{y}_\perp + \widehat{e}_k)$, then $(\mathbf{c}_k^\top, u_k) = (\mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top, \mathbf{s}_k^\top \mathbf{y}_\perp + \widehat{e}_k)$. Together with the first k uniform pairs and the function output above, the entire experiment is identically distributed to Hybrid $2.k.1$. The public values $\mathbf{x}^\star$ and $\mathbf{y}^\star = \mathbf{B}\mathbf{x}^\star$ are computed from the public LWE matrix and fresh independent randomness and do not affect this distribution. Otherwise, $(\mathbf{v}^\top, v)$ is uniformly distributed over $\mathbb{Z}_q^m \times \mathbb{Z}_q$. In this case, $(\mathbf{c}_k, u_k)$ is uniform, and $d_\beta = \sum_{j=0}^{h-1} \mathbf{c}_j^\top \mathbf{z}_j + u_h + \bar{e} + \lfloor q/2 \rfloor m_\beta$ is exactly the function output used in Hybrid $2.k.2$. The same computation is used if the challenge-tag key is queried after the challenge. Hence the entire experiment is identically distributed to Hybrid $2.k.2$. It follows that $\mathcal{B}$ distinguishes the two LWE distributions with advantage $|\mathsf{Adv}_{\mathcal{A}}^{2.k.1} - \mathsf{Adv}_{\mathcal{A}}^{2.k.2}|$,

contradicting the $\mathsf{LWE}_{n,m+1,q,\chi}$ assumption. □

Lemma 5.6. For every $i \in \{0,\dots,k-1\}$ and every admissible adversary $\mathcal{A}$ of unbounded running time, $\mathsf{Adv}^{2.i.2}_{\mathcal{A}} = \mathsf{Adv}^{2.(i+1).0}_{\mathcal{A}}$.

Proof. Hybrid $2.i.2$ samples the first $i+1$ pairs $(\mathbf{c}_j, u_j)$ uniformly, as does Hybrid $2.(i+1).0$. If $h \le i$, their function outputs are identical. If $h = i+1$, then $\mathbf{y}_{\alpha_{i+1}} = \mathbf{y}_\perp$ and $\mathbf{s}_{i+1}^\top \mathbf{y}_{\alpha_{i+1}} = u_{i+1} - \widehat{e}_{i+1}$. If $h > i+1$, then $\mathbf{A}\mathbf{z}_{i+1} = \mathbf{y}_{\alpha_{i+1}}$ and $\mathbf{s}_{i+1}^\top \mathbf{y}_{\alpha_{i+1}} = (\widehat{\mathbf{c}}_{i+1}^\top - \mathbf{e}_{i+1}^\top)\mathbf{z}_{i+1}$. Hence their function outputs are identical in every case. □

Lemma 5.7. For every admissible adversary $\mathcal{A}$ of unbounded running time, $\mathsf{Adv}^{2.k.2}_{\mathcal{A}} = \mathsf{Adv}^{3}_{\mathcal{A}}$.

Proof. In Hybrid $2.k.2$, all pairs $(\mathbf{c}_i, u_i)$ are uniformly and independently distributed. If the challenge-tag key is queried before the challenge, then h is fixed before these pairs are sampled. If the challenge occurs first, the simulated IPFE ciphertext depends only on the plaintext length, so S and h may depend on $\mathbf{c}_0,\dots,\mathbf{c}_k$ but not on $u_0,\dots,u_k$. Consequently, conditioned on the transcript that determines h, the value u_h remains uniform and independent. Hence $u_h + \bar{e} + \lfloor q/2 \rfloor m_\beta$ and $u_h + \bar{e}$ are identically distributed. If no key with tag t^* is queried, neither value is used. □

Lemma 5.8. For every admissible adversary $\mathcal{A}$ of unbounded running time, $\mathsf{Adv}^{3}_{\mathcal{A}} = 1/2$.

Proof. In Hybrid 3, the public parameters, all key-query responses, the components $\mathbf{c}_0,\dots,\mathbf{c}_k$, and every function output supplied to the tagged IPFE simulator are independent of β. Thus the entire view of $\mathcal{A}$ is independent of β, and the adversary wins with probability exactly $1/2$. □

Proof of Theorem 5.1. By Lemma 5.2 and Lemma 5.3, Hybrid 0 and Hybrid 2.0.0 are computationally indistinguishable. Apply Lemma 5.4 and Lemma 5.5 for every $i \in \{0,\dots,k\}$, and apply Lemma 5.6 for every $i \in \{0,\dots,k-1\}$. It follows that Hybrid 2.0.0 and Hybrid $2.k.2$ are computationally indistinguishable. By Lemma 5.7, Hybrid $2.k.2$ and Hybrid 3 are identically distributed. Finally, Lemma 5.8 gives $\mathsf{Adv}^{3}_{\mathcal{A}} = 1/2$, and the theorem follows. □

6 Batched IBE from Laconic Function Evaluation and Tagged Functional Encryption

We construct a batched IBE scheme from laconic function evaluation and tagged functional encryption. The identity and message spaces are $\mathcal{I} = \{0,1\}^k$ and $\mathcal{M} = \{0,1\}^\mu$, where $k = k(\lambda)$ and $\mu = \mu(\lambda)$ are polynomially bounded. We use an LFE scheme for circuits from $\mathcal{I} \times \mathcal{M}$ to $\mathcal{M} \cup \{\perp\}$ and a tagged functional encryption scheme for the circuits defined below whose tag space contains the epoch space $\mathcal{T}$.

Let $r_{\mathsf{LFE}} = r_{\mathsf{LFE}}(\lambda)$ denote the number of random bits used by the LFE encoding algorithm. We write $\mathsf{LFE.Enc}(\mathsf{pk}, h, x; R)$ for the deterministic computation using random tape $R \in \{0,1\}^{r_{\mathsf{LFE}}}$. For every set $S \subseteq \mathcal{I}$, define the circuit $F_S : \mathcal{I} \times \mathcal{M} \to \mathcal{M} \cup \{\perp\}$ as

$$F_S(r,m) := \begin{cases} m, & \text{if } r \in S, \\ \perp, & \text{if } r \notin S. \end{cases}$$

A balanced membership circuit has depth $O(k)$ for every $S \subseteq \mathcal{I}$, since $|S| \le 2^k$. We use LFE for this depth bound, without fixing a maximum batch size. Our BIBE scheme for these identity and message spaces is as follows.

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

$$\mathsf{pk}_{\mathsf{LFE}} \leftarrow \mathsf{LFE.Setup}(1^\lambda), \qquad (\mathsf{mpk}_{\mathsf{FE}}, \mathsf{msk}_{\mathsf{FE}}) \leftarrow \mathsf{FE.Setup}(1^\lambda).$$

It outputs $\mathsf{pp} := (\mathsf{pk}_{\mathsf{LFE}}, \mathsf{mpk}_{\mathsf{FE}})$ and $\mathsf{msk} := (\mathsf{pk}_{\mathsf{LFE}}, \mathsf{msk}_{\mathsf{FE}})$.

$\mathsf{Digest}(\mathsf{pp}, S)$: It takes as input public parameters pp and a set $S \subseteq \mathcal{I}$. Parse pp as $(\mathsf{pk}_{\mathsf{LFE}}, \mathsf{mpk}_{\mathsf{FE}})$. It constructs the circuit F_S defined above, computes $(h_S, \psi_S) := \mathsf{LFE.Hash}(\mathsf{pk}_{\mathsf{LFE}}, F_S)$, and outputs the digest $\mathbf{d}_S := h_S$. The value ψ_S is not included in the digest and is recomputed during decryption.

$\mathsf{KeyGen}(\mathsf{msk}, \mathbf{d}_S, t)$: Parse msk as $(\mathsf{pk}_{\mathsf{LFE}}, \mathsf{msk}_{\mathsf{FE}})$ and $\mathbf{d}_S$ as h_S. Define the deterministic circuit G_{h_S} that takes as input $(r, m, R) \in \mathcal{I} \times \mathcal{M} \times \{0,1\}^{r_{\mathsf{LFE}}}$ and computes

$$G_{h_S}(r, m, R) := \big(r,\, \mathsf{LFE.Enc}\big(\mathsf{pk}_{\mathsf{LFE}}, h_S, (r, m); R\big)\big)\,.$$

It generates $\widetilde{\mathsf{sk}}_{S,t} \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}_{\mathsf{FE}}, t, G_{h_S})$ and outputs $\mathsf{sk}_{S,t} := \widetilde{\mathsf{sk}}_{S,t}$. The tagged FE circuit hardwires only the LFE public key and the short digest h_S, rather than the set S itself.

$\mathsf{Enc}(\mathsf{pp}, r, t, m)$: Parse pp as $(\mathsf{pk}_{\mathsf{LFE}}, \mathsf{mpk}_{\mathsf{FE}})$. It samples $R \leftarrow \{0,1\}^{r_{\mathsf{LFE}}}$ and generates

$$C_{\mathsf{FE}} \leftarrow \mathsf{FE.Enc}\big(\mathsf{mpk}_{\mathsf{FE}}, t, (r, m, R)\big), \qquad \mathsf{ct} := (r, t, C_{\mathsf{FE}}).$$

$\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}_{S,t}, \mathsf{ct}, S, r, t)$: Parse pp as $(\mathsf{pk}_{\mathsf{LFE}}, \mathsf{mpk}_{\mathsf{FE}})$, set $\widetilde{\mathsf{sk}}_{S,t} := \mathsf{sk}_{S,t}$, and parse ct as $(r', t', C_{\mathsf{FE}})$. If $r' \neq r$ or $t' \neq t$, it outputs $\perp$. It reconstructs F_S from S and recomputes $(h_S, \psi_S) := \mathsf{LFE.Hash}(\mathsf{pk}_{\mathsf{LFE}}, F_S)$. It computes $(\widehat{r}, E) \leftarrow \mathsf{FE.Dec}(\widetilde{\mathsf{sk}}_{S,t}, C_{\mathsf{FE}})$ and outputs $\perp$ if the result cannot be parsed in this form or $\widehat{r} \neq r$. It then computes $z \leftarrow \mathsf{LFE.Dec}(\mathsf{pk}_{\mathsf{LFE}}, E, F_S, \psi_S)$. If $z \in \mathcal{M}$, it outputs z. Otherwise, it outputs $\perp$.

Correctness. Correctness follows directly from the correctness of the tagged functional encryption scheme and the LFE scheme.

Efficiency analysis. Let $\ell = |S| \geq 1$. A balanced circuit for F_S has depth $O(\log k + \log \ell) = O(k)$, since $\ell \leq 2^k$. We therefore instantiate Corollary 3.5 at depth $O(k)$. Together with Corollary 3.7, this gives polynomial-size public parameters, keys, digests, and ciphertexts for polynomially bounded k, μ, and epoch length. Digest computation and decryption still process the full set S.

6.1 Security of the LFE–Tagged FE Construction

Theorem 6.1. Assume that LFE satisfies adaptive encoder privacy, and that FE satisfies 1-adaptive simulation security as in Definition 3.6. Then the construction in Section 6 satisfies adaptive security as in Section 4.1.

6.1.1 Sequence of Hybrid Games

We prove Theorem 6.1 using the following sequence of hybrid games. Hybrid 0 is the real adaptive security experiment. Hybrid 1 replaces the tagged FE public parameters, functional keys, and challenge ciphertext with the outputs of the tagged FE simulator. Hybrid 2 replaces the LFE encoding supplied as the unique challenge-tag function output with a simulated encoding. The last hybrid is independent of the challenge bit.

Hybrid 0: This is the real adaptive security experiment in Section 4.1.

1. The challenger generates $\mathsf{pk}_{\mathsf{LFE}} \leftarrow \mathsf{LFE.Setup}(1^\lambda)$ and $(\mathsf{mpk}_{\mathsf{FE}}, \mathsf{msk}_{\mathsf{FE}}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$. It sets $\mathsf{pp} := (\mathsf{pk}_{\mathsf{LFE}}, \mathsf{mpk}_{\mathsf{FE}})$ and $\mathsf{msk} := (\mathsf{pk}_{\mathsf{LFE}}, \mathsf{msk}_{\mathsf{FE}})$, samples $\beta \leftarrow \{0,1\}$, and gives pp to $\mathcal{A}$.

2. For every pre-challenge key query (S,t), the challenger constructs F_S, computes $(h_S, \psi_S) := \mathsf{LFE.Hash}(\mathsf{pk}_{\mathsf{LFE}}, F_S)$, and defines G_{h_S} as in Section 6. It runs $\mathsf{FE.KeyGen}(\mathsf{msk}_{\mathsf{FE}}, t, G_{h_S})$ to obtain $\widetilde{\mathsf{sk}}_{S,t}$ and returns $\mathsf{sk}_{S,t} := \widetilde{\mathsf{sk}}_{S,t}$.

3. During the challenge query, $\mathcal{A}$ outputs (r^*, t^*, m_0, m_1). The challenger samples $R^* \leftarrow \{0,1\}^{r_{\mathsf{LFE}}}$ and sets $X^*_\beta := (r^*, m_\beta, R^*)$. It runs $\mathsf{FE.Enc}(\mathsf{mpk}_{\mathsf{FE}}, t^*, X^*_\beta)$ to obtain C^*_{FE} and returns $\mathsf{ct}^* := (r^*, t^*, C^*_{\mathsf{FE}})$.

4. Every post-challenge key query (S,t) is answered as in step 2. The adversary eventually outputs β' and wins if and only if $\beta' = \beta$.

All queries satisfy the admissibility conditions in Section 4.1. In particular, at most one key query has epoch t^*, and its set S satisfies $r^* \notin S$.

Hybrid 1: It is the same as Hybrid 0, except that the challenger runs the ideal experiment of the tagged FE scheme. The LFE setup and all digest computations remain unchanged.

1. The challenger generates $\mathsf{pk}_{\mathsf{LFE}} \leftarrow \mathsf{LFE.Setup}(1^\lambda)$ and uses the setup algorithm of the tagged FE simulator to generate $\mathsf{mpk}_{\mathsf{FE}}$ and its internal state. It sets $\mathsf{pp} := (\mathsf{pk}_{\mathsf{LFE}}, \mathsf{mpk}_{\mathsf{FE}})$, samples $\beta \leftarrow \{0,1\}$, and gives pp to $\mathcal{A}$.

2. For every pre-challenge key query (S,t), the challenger computes h_S, defines G_{h_S}, and obtains $\widetilde{\mathsf{sk}}_{S,t}$ from the pre-challenge key-generation algorithm of the tagged FE simulator on (t, G_{h_S}). It returns $\mathsf{sk}_{S,t} := \widetilde{\mathsf{sk}}_{S,t}$ to $\mathcal{A}$.

3. During the challenge query, $\mathcal{A}$ outputs (r^*, t^*, m_0, m_1), the challenger samples $R^* \leftarrow \{0,1\}^{r_{\mathsf{LFE}}}$, and it sets $X^*_\beta := (r^*, m_\beta, R^*)$. If a key for (S, t^*) was queried before the challenge, it sets $E^*_\beta := \mathsf{LFE.Enc}(\mathsf{pk}_{\mathsf{LFE}}, h_S, (r^*, m_\beta); R^*)$ and gives the tagged FE ciphertext simulator t^*, $1^{|X^*_\beta|}$, and the singleton function-value list containing $(G_{h_S}, (r^*, E^*_\beta))$. If no key with tag t^* was queried, the ciphertext simulator is given only t^* and $1^{|X^*_\beta|}$. It obtains C^*_{FE} and returns $\mathsf{ct}^* := (r^*, t^*, C^*_{\mathsf{FE}})$.

4. For every post-challenge query (S,t), the challenger computes h_S and defines G_{h_S}. If $t = t^*$, it sets $E^*_\beta := \mathsf{LFE.Enc}(\mathsf{pk}_{\mathsf{LFE}}, h_S, (r^*, m_\beta); R^*)$ and gives $(t^*, G_{h_S}, (r^*, E^*_\beta))$ to the post-challenge key-generation algorithm of the tagged FE simulator. If $t \neq t^*$, it invokes the corresponding non-challenge-tag key-generation algorithm on (t, G_{h_S}) without a function output. In either case, let $\widetilde{\mathsf{sk}}_{S,t}$ be the generated key. The challenger returns $\mathsf{sk}_{S,t} := \widetilde{\mathsf{sk}}_{S,t}$ to $\mathcal{A}$.

Hybrid 2: It is the same as Hybrid 1, except that the challenger replaces the LFE encoding supplied as the challenge-tag function output. Steps 3 and 4 are changed as follows.

3. If (S, t^*) was queried before the challenge, then $r^* \notin S$ and $F_S(r^*, m_\beta) = \perp$. The challenger invokes $\mathsf{LFE.Sim}$ on $(1^\lambda, \mathsf{pk}_{\mathsf{LFE}}, F_S, \perp)$ and receives a simulated LFE encoding E^{sim}_S. It gives the tagged FE ciphertext simulator the singleton function-value list containing $(G_{h_S}, (r^*, E^{\mathsf{sim}}_S))$.

4. If (S, t^*) is queried after the challenge, then $r^* \notin S$ and $F_S(r^*, m_\beta) = \perp$. The challenger invokes $\mathsf{LFE.Sim}$ on $(1^\lambda, \mathsf{pk}_{\mathsf{LFE}}, F_S, \perp)$ and receives a simulated LFE encoding E_S^{sim}. It gives $(t^*, G_{h_S}, (r^*, E_S^{\mathsf{sim}}))$ to the post-challenge key-generation algorithm.

If no key with tag t^* is queried, Hybrids 1 and 2 are identical.

6.1.2 Indistinguishability of Hybrids

For every adversary $\mathcal{A}$, let $\mathsf{Adv}^i_{\mathcal{A}}$ denote the probability that $\mathcal{A}$ wins in Hybrid i.

Lemma 6.2. Assume that the tagged FE scheme satisfies 1-adaptive simulation security as in Definition 3.6. Then, for every admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $|\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}}| \leq \mathsf{negl}(\lambda)$.

Proof. Suppose that a PPT adversary $\mathcal{A}$ distinguishes the two hybrids. We construct a reduction algorithm $\mathcal{B}$ against the 1-adaptive simulation security of the tagged FE scheme.

The tagged FE challenger gives $\mathcal{B}$ a master public key $\mathsf{mpk}_{\mathsf{FE}}$. The reduction generates $\mathsf{pk}_{\mathsf{LFE}} \leftarrow \mathsf{LFE.Setup}(1^\lambda)$, samples $\beta \leftarrow \{0,1\}$, sets $\mathsf{pp} := (\mathsf{pk}_{\mathsf{LFE}}, \mathsf{mpk}_{\mathsf{FE}})$, and gives pp to $\mathcal{A}$. For every key query (S, t), it constructs F_S, computes $(h_S, \psi_S) := \mathsf{LFE.Hash}(\mathsf{pk}_{\mathsf{LFE}}, F_S)$, defines G_{h_S}, and submits (t, G_{h_S}) as its tagged FE functional-key query. Upon receiving $\widetilde{\mathsf{sk}}_{S,t}$, it returns $\widetilde{\mathsf{sk}}_{S,t}$ to $\mathcal{A}$. When $\mathcal{A}$ submits (r^*, t^*, m_0, m_1), the reduction samples $R^* \leftarrow \{0,1\}^{r_{\mathsf{LFE}}}$, sets $X^*_\beta := (r^*, m_\beta, R^*)$, and submits (t^*, X^*_β) as its tagged FE challenge. It returns $\mathsf{ct}^* := (r^*, t^*, C^*_{\mathsf{FE}})$ to $\mathcal{A}$ and continues to forward all post-challenge key queries. When $\mathcal{A}$ outputs β', the reduction outputs 1 if and only if $\beta' = \beta$.

The admissibility of $\mathcal{A}$ guarantees that $\mathcal{B}$ makes at most one functional-key query at the challenge tag and an arbitrary polynomial number of queries at every other tag. The real tagged FE experiment gives exactly Hybrid 0, whereas the ideal tagged FE experiment gives exactly Hybrid 1. Hence a non-negligible difference between the two hybrids contradicts the adaptive simulation security of the tagged FE scheme. □

Lemma 6.3. Assume that LFE satisfies adaptive encoder privacy as defined in Section 3.2. Then, for every admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $|\mathsf{Adv}^1_{\mathcal{A}} - \mathsf{Adv}^2_{\mathcal{A}}| \leq \mathsf{negl}(\lambda)$.

Proof. Suppose that a PPT adversary $\mathcal{A}$ distinguishes the two hybrids. We construct a reduction algorithm $\mathcal{B}$ against adaptive encoder privacy of LFE. The reduction runs the ideal tagged FE experiment and uses its LFE challenge encoding as the unique function output associated with the challenge tag.

The LFE challenger gives $\mathcal{B}$ a public key $\mathsf{pk}_{\mathsf{LFE}}$. The reduction uses the setup algorithm of the tagged FE simulator to generate $\mathsf{mpk}_{\mathsf{FE}}$ and its internal state, samples $\beta \leftarrow \{0,1\}$, and gives $\mathsf{pp} := (\mathsf{pk}_{\mathsf{LFE}}, \mathsf{mpk}_{\mathsf{FE}})$ to $\mathcal{A}$. For every key query (S, t), it can compute $(h_S, \psi_S) := \mathsf{LFE.Hash}(\mathsf{pk}_{\mathsf{LFE}}, F_S)$ locally because the hashing algorithm is public and deterministic. It uses h_S to define G_{h_S}, invokes the appropriate tagged FE key-generation algorithm, and returns the resulting key $\widetilde{\mathsf{sk}}_{S,t}$. Suppose first that a key for (S, t^*) was queried before the challenge. When $\mathcal{A}$ later submits (r^*, t^*, m_0, m_1), the reduction selects the already fixed circuit F_S as its LFE challenge function. The digest returned by the LFE challenger is the same h_S that was computed locally. After receiving this digest, $\mathcal{B}$ submits (r^*, m_β) as its LFE challenge input and uses the returned encoding E in the singleton function-value list $(G_{h_S}, (r^*, E))$ supplied to the tagged FE ciphertext simulator. Suppose instead that no key with tag t^* was queried before the challenge. The reduction invokes the tagged FE ciphertext simulator using only t^* and the plaintext length. If $\mathcal{A}$ later queries (S, t^*), the reduction

selects F_S as its LFE challenge function, submits the already fixed (r^*, m_β) as its challenge input after receiving h_S, and uses the returned encoding in the post-challenge tagged FE key-generation algorithm. If no key with tag t^* is ever queried, the two hybrids are identical. In this case, $\mathcal{B}$ completes the LFE experiment using an arbitrary function and input and ignores the returned encoding.

In either query order, admissibility gives $F_S(r^*, m_\beta) = \perp$. When the LFE challenger returns a real encoding, its random tape is fresh and uniform. Since this tape has no other visible occurrence in Hybrid 1, the induced function output has exactly the distribution specified there. A simulated LFE challenge encoding gives Hybrid 2. The reduction guesses that the encoding is real if and only if $\beta' = \beta$. Its advantage is $|\mathsf{Adv}^1_{\mathcal{A}} - \mathsf{Adv}^2_{\mathcal{A}}|/2$, contradicting adaptive encoder privacy of LFE. □

Lemma 6.4. For every adversary $\mathcal{A}$ of unbounded running time, $\mathsf{Adv}^2_{\mathcal{A}} = 1/2$.

Proof. In Hybrid 2, the simulated LFE encoding is generated from $F_S(r^*, m_\beta) = \perp$ and is therefore independent of β. If the challenge-tag key is queried before the challenge, this encoding and the fixed plaintext length are the only challenge-dependent values given to the tagged FE ciphertext simulator. If the key is queried after the challenge, the simulated ciphertext depends only on the challenge tag and the plaintext length, and the later key simulation is again given an encoding independent of β. Every non-challenge-tag key is simulated without a function output. Thus the entire view of $\mathcal{A}$ is independent of β, and $\mathsf{Adv}^2_{\mathcal{A}} = 1/2$. □

Proof of Theorem 6.1. By Lemmas 6.2 and 6.3, the winning probabilities in Hybrids 0 and 2 differ by at most a negligible function. Lemma 6.4 gives $\mathsf{Adv}^2_{\mathcal{A}} = 1/2$. Since Hybrid 0 is the real adaptive security experiment, the theorem follows. □

7 Batched IBE from RNCE and Error-Leakage LWE

We modify the LE–IPFE construction in Section 5, using the tagged IPFE scheme of [ET25] and tagged RNCE from Corollary 3.10. Their internal lattice parameters are chosen separately. Except for the changes described below, each algorithm is the corresponding algorithm of that construction. Fix a polynomial identity-length bound $K = K(\lambda)$. For any $1 \le k \le K$, take $\mathcal{I} = \{0,1\}^k$ and $\mathcal{M} = \{0,1\}$. For an LE–IPFE ciphertext $E = (r, t, \mathbf{c}_0, \ldots, \mathbf{c}_k, C_{\mathsf{IPFE}})$, write $\langle E \rangle$ for its canonical bit representation of public length L_E, and instantiate tagged RNCE with message space $\{0,1\}^{L_E}$.

Let q be prime. Let $\chi = D_{\mathbb{Z},\sigma}$ and $\bar{\chi} = D_{\mathbb{Z},\bar{\sigma}}$ be discrete Gaussians, and set $B_\chi = \lceil \sqrt{\lambda}\,\sigma \rceil$ and $B_{\bar{\chi}} = \lceil \sqrt{\lambda}\,\bar{\sigma} \rceil$. We choose parameters satisfying the following constraints:

- $n = \mathsf{poly}(\lambda) \ge \lambda$, $q = \mathsf{poly}(\lambda)$, $m = n\lceil \log q \rceil$ (lattice parameters)
- $\sigma = n^2$, $\bar{\sigma} = 2\sqrt{2m+1}\,\sigma$ (for error-leakage LWE)
- $B_{\bar{\chi}} + (2K+1)mB_\chi < q/8$ (for correctness)

One possible parameter setting is

$$n = \lambda, \quad (K+1)n^5 \le q < 2(K+1)n^5, \quad m = n\lceil \log q \rceil, \quad \sigma = n^2, \quad \bar{\sigma} = 2\sqrt{2m+1}\,\sigma.$$

For every fixed polynomial K, these choices satisfy the constraints for sufficiently large λ. The tagged IPFE scheme has plaintext modulus q and vector dimension $n+1$. Let $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ be the gadget matrix. Our BIBE scheme is as follows.

$\mathsf{Setup}(1^\lambda)$: It is the same as the Setup algorithm in Section 5, except that it uses the above parameters, samples $\mathbf{x}^\star \leftarrow \{0,1\}^m$ uniformly, sets $\mathbf{y}^\star := \mathbf{B}\mathbf{x}^\star$, and additionally generates $(\mathsf{npk}, \mathsf{nmsk}) \leftarrow \mathsf{RNCE.Setup}(1^\lambda)$. It outputs $\mathsf{pp} := (\mathbf{A}, \mathbf{B}, \mathbf{G}, \mathbf{y}_\perp, \mathbf{y}^\star, \mathbf{x}^\star, \mathsf{mpk}_{\mathsf{IPFE}}, \mathsf{npk})$ and $\mathsf{msk} := (\mathsf{msk}_{\mathsf{IPFE}}, \mathsf{nmsk})$.

$\mathsf{Digest}(\mathsf{pp}, S)$: Parse pp as $(\mathbf{A}, \mathbf{B}, \mathbf{G}, \mathbf{y}_\perp, \mathbf{y}^\star, \mathbf{x}^\star, \mathsf{mpk}_{\mathsf{IPFE}}, \mathsf{npk})$ and ignore npk. It then runs the Digest algorithm in Section 5 and outputs the root label $\mathbf{y}_S$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathbf{y}_S, t)$: It is the same as the KeyGen algorithm in Section 5, except that it also generates $\mathsf{dk}_t \leftarrow \mathsf{RNCE.KeyGen}(\mathsf{nmsk}, t)$. Namely, it sets $\mathbf{f}_S := (\mathbf{y}_S^\top, 1)^\top$, generates $\mathsf{sk}_{\mathsf{IPFE}} \leftarrow \mathsf{IPFE.KeyGen}(\mathsf{msk}_{\mathsf{IPFE}}, t, \mathbf{f}_S)$, and outputs $\mathsf{sk}_{S,t} := (\mathsf{sk}_{\mathsf{IPFE}}, \mathsf{dk}_t)$.

$\mathsf{Enc}(\mathsf{pp}, r, t, \mu)$: It computes $\mathbf{c}_0, \ldots, \mathbf{c}_k$ and C_{IPFE} exactly as the Enc algorithm in Section 5, using $\bar{e} \leftarrow \bar{\chi}$, and sets $E := (r, t, \mathbf{c}_0, \ldots, \mathbf{c}_k, C_{\mathsf{IPFE}})$. It generates $C_{\mathsf{RNCE}} \leftarrow \mathsf{RNCE.Enc}(\mathsf{npk}, t, \langle E \rangle)$ and outputs $\mathsf{ct} := (r, t, C_{\mathsf{RNCE}})$.

$\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}_{S,t}, \mathsf{ct}, S, r, t)$: It is the same as the Dec algorithm in Section 5, except that it first recovers the LE–IPFE ciphertext through RNCE. Parse $\mathsf{sk}_{S,t} = (\mathsf{sk}_{\mathsf{IPFE}}, \mathsf{dk})$ and $\mathsf{ct} = (r', t', C_{\mathsf{RNCE}})$. It outputs $\perp$ unless $r' = r$ and $t' = t$, and computes $\widehat{E} \leftarrow \mathsf{RNCE.Dec}(\mathsf{dk}, C_{\mathsf{RNCE}})$. It parses $\widehat{E}$ as $\langle E \rangle$ for $E = (r, t, \mathbf{c}_0, \ldots, \mathbf{c}_k, C_{\mathsf{IPFE}})$. It outputs $\perp$ if parsing fails, and otherwise runs the Dec algorithm in Section 5 on $(\mathsf{pp}, \mathsf{sk}_{\mathsf{IPFE}}, E, S, r, t)$, ignoring the additional RNCE public key in pp.

Correctness. Let $\mu \in \{0, 1\}$ be the encrypted message and let $r \in S$. Whenever RNCE and IPFE decryption are correct, the same cancellation as in Section 5 gives $v = \lfloor q/2 \rfloor \mu + \bar{e} - \sum_{j=0}^{k-1} \mathbf{e}_j^\top \mathbf{z}_j - \mathbf{e}_k^\top \mathbf{x}^\star \pmod q$. The Gaussian tail bounds hold for all error samples except with negligible probability. Since $\|\mathbf{z}_j\|_1 \le 2m$ and $\|\mathbf{x}^\star\|_1 \le m$, the total error satisfies

$$\left| \bar{e} - \sum_{j=0}^{k-1} \mathbf{e}_j^\top \mathbf{z}_j - \mathbf{e}_k^\top \mathbf{x}^\star \right| \le B_{\bar{\chi}} + (2k+1) m B_\chi \le B_{\bar{\chi}} + (2K+1) m B_\chi < q/8.$$

Efficiency analysis. Let $b = \lceil \log q \rceil$ and let $\ell_{\mathcal{T}} = \mathsf{poly}(\lambda)$ be the epoch length. Encoding $\mathbf{x}^\star$ in m bits and the other arithmetic components modulo q, and omitting the fixed gadget matrix, the bit lengths are $|\mathsf{pp}| = (3nm + 2n)b + m + |\mathsf{mpk}_{\mathsf{IPFE}}| + |\mathsf{npk}|$, $|\mathsf{msk}| = |\mathsf{msk}_{\mathsf{IPFE}}| + |\mathsf{nmsk}|$, $|\mathbf{y}_S| = nb$, $|\mathsf{sk}_{S,t}| = |\mathsf{sk}_{\mathsf{IPFE}}| + |\mathsf{dk}_t|$, $L_E = k + \ell_{\mathcal{T}} + (2k+1)mb + |C_{\mathsf{IPFE}}|$, $|\mathsf{ct}| = k + \ell_{\mathcal{T}} + L_E + \mathsf{poly}(\lambda, \ell_{\mathcal{T}})$. For fixed $K(\lambda)$ and $\ell_{\mathcal{T}}(\lambda)$, the IPFE components and RNCE keys have polynomial bit length in λ. Thus the public parameters, master secret key, digest, and batch key have size $O_\lambda(1)$, and the ciphertext has size $O_\lambda(k)$.

Let $\ell = |S| \ge 1$. Including the IPFE and RNCE calls, setup and key generation take $O_\lambda(1)$ time, and encryption takes $O_\lambda(k)$ time. Digest computation and decryption take $O_\lambda(\ell k)$ time, including the sparse-tree computation as in Section 5.

Reusable public preprocessing. The sparse tree can be retained and reused exactly as in Section 5.

7.1 Security

Error distributions. Set $D = 2m + 1$ and $\chi^\star = D_{\mathbb{Z}, 2\sigma/\sqrt{17}}$. For $d \in \{2m+1, m+1\}$ and $\rho = 1$, apply Corollary 3.3 with $a = 4$, $b = \sqrt{68D - 1}$, $\sigma_0 = \sigma/\sqrt{17}$, and $\epsilon = 2^{-\lambda}$. These choices give the widths σ and $\bar{\sigma}$ above. Moreover, $b^2 \ge (306/16)d$, and σ_0 exceeds the required smoothing parameters for sufficiently large λ. Thus $\mathsf{LWE}_{n, 2m+1, q, \chi^\star}$ implies both error-leakage LWE assumptions used below.

Theorem 7.1. Assume that the tagged RNCE scheme satisfies post-challenge receiver simulatability, and that the tagged IPFE scheme is adaptively simulation secure with at most one functional key at the challenge tag and unrestricted functional keys at every non-challenge tag. Assume that $\mathsf{LWE}_{n,2m+1,q,\chi^\star}$ holds. Then the construction in Section 7 satisfies adaptive security following Section 4.1.

Sequence of Hybrid Games. We mark changes from the preceding hybrid in red. We separate executions according to when the challenge-epoch key is queried. In a Type 1 execution, no key for t^* is queried before the challenge. In a Type 2 execution, a key for t^* has already been queried, so its set S is known when the challenge is generated. For each type, the filtered game outputs an independent uniform bit on executions of the other type. Let $p_{\mathcal{A}}^{g,i}$ denote the success probability in the Type-g filtered Hybrid i. The Type 1 sequence is Hybrid 0, Hybrid 1, Hybrid 2, Hybrid 3, and Hybrids 4.0 through 4.$(k+1)$. The Type 2 sequence starts with Hybrid 0, which is also Hybrid 2, and then follows the common sequence from Hybrid 3. It suffices to bound the two filtered advantages, since their sum is the advantage in the real experiment.

Hybrid 0: This is the adaptive security experiment for the construction in Section 7.

1. The challenger samples $\mathbf{A}_0, \mathbf{A}_1, \mathbf{B} \leftarrow \mathbb{Z}_q^{n\times m}$ and $\mathbf{y}_\perp \leftarrow \mathbb{Z}_q^n$, sets $\mathbf{A} := [\mathbf{A}_0 \mid \mathbf{A}_1]$, samples $\mathbf{x}^\star \leftarrow \{0,1\}^m$, and sets $\mathbf{y}^\star := \mathbf{B}\mathbf{x}^\star$. It generates $(\mathsf{mpk}_{\mathsf{IPFE}}, \mathsf{msk}_{\mathsf{IPFE}}) \leftarrow \mathsf{IPFE.Setup}(1^\lambda)$ and $(\mathsf{npk}, \mathsf{nmsk}) \leftarrow \mathsf{RNCE.Setup}(1^\lambda)$. It sets $\mathsf{pp} := (\mathbf{A}, \mathbf{B}, \mathbf{G}, \mathbf{y}_\perp, \mathbf{y}^\star, \mathbf{x}^\star, \mathsf{mpk}_{\mathsf{IPFE}}, \mathsf{npk})$, samples $\beta \leftarrow \{0,1\}$, and gives pp to $\mathcal{A}$.

2. For every pre-challenge key query (S,t), the challenger sets $\mathbf{y}_S := \mathbf{y}_\perp$ if $S = \emptyset$. Otherwise, it computes the sparse-tree labels by assigning $\mathbf{y}^\star$ to the leaves in S, assigning $\mathbf{y}_\perp$ to an empty child, and setting $\mathbf{y}_\alpha := \mathbf{A}_0(-\mathbf{G}^{-1}(\mathbf{y}_{\alpha 0})) + \mathbf{A}_1(-\mathbf{G}^{-1}(\mathbf{y}_{\alpha 1}))$ at every active internal node, and sets $\mathbf{y}_S := \mathbf{y}_\epsilon$. It sets $\mathbf{f}_S := (\mathbf{y}_S^\top, 1)^\top$, generates $\mathsf{sk}_{\mathsf{IPFE}} \leftarrow \mathsf{IPFE.KeyGen}(\mathsf{msk}_{\mathsf{IPFE}}, t, \mathbf{f}_S)$ and $\mathsf{dk}_t \leftarrow \mathsf{RNCE.KeyGen}(\mathsf{nmsk}, t)$, and returns $\mathsf{sk}_{S,t} := (\mathsf{sk}_{\mathsf{IPFE}}, \mathsf{dk}_t)$.

3. On challenge (r^*, t^*, m_0, m_1), write $r^* = b_1 \cdots b_k$. If more than one previous key query has epoch t^*, or if the unique such query (S, t^*) satisfies $r^* \in S$, the game aborts. Otherwise, the challenger samples $\mathbf{s}_0, \ldots, \mathbf{s}_k \leftarrow \mathbb{Z}_q^n$, $\mathbf{e}_j \leftarrow \chi^{2m}$ for every $j < k$, $\mathbf{e}_k \leftarrow \chi^m$, and $\bar{e} \leftarrow \bar{\chi}$. For every $j < k$, it sets $\mathbf{D}_j := [(1-b_{j+1})\mathbf{G} \mid b_{j+1}\mathbf{G}]$ and $\mathbf{c}_j^\top := \mathbf{s}_j^\top \mathbf{A} + \mathbf{s}_{j+1}^\top \mathbf{D}_j + \mathbf{e}_j^\top$, and it sets $\mathbf{c}_k^\top := \mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top$. It generates

$$C^*_{\mathsf{IPFE}} \leftarrow \mathsf{IPFE.Enc}\left(\mathsf{mpk}_{\mathsf{IPFE}}, t^*, \begin{bmatrix} \mathbf{s}_0 \\ \bar{e} + \lfloor q/2 \rfloor m_\beta \end{bmatrix}\right),$$

   sets $E_\beta := (r^*, t^*, \mathbf{c}_0, \ldots, \mathbf{c}_k, C^*_{\mathsf{IPFE}})$, generates $C^*_{\mathsf{RNCE}} \leftarrow \mathsf{RNCE.Enc}(\mathsf{npk}, t^*, \langle E_\beta \rangle)$, and returns $\mathsf{ct}^* := (r^*, t^*, C^*_{\mathsf{RNCE}})$.

4. For every post-challenge key query (S,t), the challenger aborts if $t = t^*$ and either a key with epoch t^* has already been queried or $r^* \in S$. Otherwise, it computes $\mathbf{y}_S$ using the above tree equations and sets $\mathbf{f}_S := (\mathbf{y}_S^\top, 1)^\top$. It generates $\mathsf{sk}_{\mathsf{IPFE}} \leftarrow \mathsf{IPFE.KeyGen}(\mathsf{msk}_{\mathsf{IPFE}}, t, \mathbf{f}_S)$ and $\mathsf{dk}_t \leftarrow \mathsf{RNCE.KeyGen}(\mathsf{nmsk}, t)$, and returns $(\mathsf{sk}_{\mathsf{IPFE}}, \mathsf{dk}_t)$. The adversary eventually outputs β' and wins if and only if $\beta' = \beta$.

Hybrid 1: In the Type 1 filtered game, it is the same as Hybrid 0, except that the challenger uses the tagged RNCE simulator at the challenge epoch.

1. The challenger samples $\mathbf{A}_0, \mathbf{A}_1, \mathbf{B} \leftarrow \mathbb{Z}_q^{n \times m}$ and $\mathbf{y}_\perp \leftarrow \mathbb{Z}_q^n$, sets $\mathbf{A} := [\mathbf{A}_0 \mid \mathbf{A}_1]$, samples $\mathbf{x}^\star \leftarrow \{0,1\}^m$, and sets $\mathbf{y}^\star := \mathbf{B}\mathbf{x}^\star$. It generates $(\mathsf{mpk}_{\mathsf{IPFE}}, \mathsf{msk}_{\mathsf{IPFE}}) \leftarrow \mathsf{IPFE.Setup}(1^\lambda)$ and $(\mathsf{npk}, \mathsf{st}_{\mathsf{RNCE}}) \leftarrow \mathsf{RNCE.SimSetup}(1^\lambda)$. It sets $\mathsf{pp} := (\mathbf{A}, \mathbf{B}, \mathbf{G}, \mathbf{y}_\perp, \mathbf{y}^\star, \mathbf{x}^\star, \mathsf{mpk}_{\mathsf{IPFE}}, \mathsf{npk})$, samples $\beta \leftarrow \{0,1\}$, and gives pp to $\mathcal{A}$.

2. For every pre-challenge key query (S, t), the challenger computes $\mathbf{y}_S \leftarrow \mathsf{Digest}(\mathsf{pp}, S)$ and sets $\mathbf{f}_S := (\mathbf{y}_S^\top, 1)^\top$. It generates $\mathsf{sk}_{\mathsf{IPFE}} \leftarrow \mathsf{IPFE.KeyGen}(\mathsf{msk}_{\mathsf{IPFE}}, t, \mathbf{f}_S)$. It then obtains $\mathsf{dk}_t \leftarrow \mathsf{RNCE.SimKeyGen}(\mathsf{st}_{\mathsf{RNCE}}, t)$ and returns $(\mathsf{sk}_{\mathsf{IPFE}}, \mathsf{dk}_t)$.

3. On challenge (r^*, t^*, m_0, m_1), the filtered game outputs an independent uniform bit if a key with epoch t^* was previously queried. Otherwise, write $r^* = b_1 \cdots b_k$. The challenger samples $\mathbf{s}_0, \ldots, \mathbf{s}_k \leftarrow \mathbb{Z}_q^n$, $\mathbf{e}_j \leftarrow \chi^{2m}$ for every $j < k$, $\mathbf{e}_k \leftarrow \chi^m$, and $\bar{e} \leftarrow \bar{\chi}$. For every $j < k$, it sets $\mathbf{D}_j := [(1 - b_{j+1})\mathbf{G} \mid b_{j+1}\mathbf{G}]$ and $\mathbf{c}_j^\top := \mathbf{s}_j^\top \mathbf{A} + \mathbf{s}_{j+1}^\top \mathbf{D}_j + \mathbf{e}_j^\top$, and sets $\mathbf{c}_k^\top := \mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top$. It generates
$$C^*_{\mathsf{IPFE}} \leftarrow \mathsf{IPFE.Enc}\left(\mathsf{mpk}_{\mathsf{IPFE}}, t^*, \begin{bmatrix} \mathbf{s}_0 \\ \bar{e} + \lfloor q/2 \rfloor m_\beta \end{bmatrix}\right),$$
stores $E_\beta := (r^*, t^*, \mathbf{c}_0, \ldots, \mathbf{c}_k, C^*_{\mathsf{IPFE}})$, and runs $(C^*_{\mathsf{RNCE}}, \tau) \leftarrow \mathsf{RNCE.SimEnc}(\mathsf{st}_{\mathsf{RNCE}}, t^*, 1^{L_E})$. It returns $(r^*, t^*, C^*_{\mathsf{RNCE}})$.

4. Post-challenge key queries are answered as in Hybrid 0, except that the challenger generates $\mathsf{dk}_t \leftarrow \mathsf{RNCE.SimKeyGen}(\mathsf{st}_{\mathsf{RNCE}}, t)$ for $t \neq t^*$ and $\mathsf{dk}_{t^*} \leftarrow \mathsf{RNCE.SimOpen}(\mathsf{st}_{\mathsf{RNCE}}, \tau, t^*, \langle E_\beta \rangle)$ for $t = t^*$. All admissibility checks and the winning condition remain unchanged.

Hybrid 2: In the Type 1 filtered game, it is the same as Hybrid 1, except that the inner challenge is generated only after the challenge-epoch key query. In the Type 2 filtered game, Hybrid 2 is Hybrid 0. *Type 1.*

3. On challenge (r^*, t^*, m_0, m_1), the filtered game outputs an independent uniform bit if a key with epoch t^* was previously queried. Otherwise, the challenger stores (r^*, t^*, m_0, m_1), runs $(C^*_{\mathsf{RNCE}}, \tau) \leftarrow \mathsf{RNCE.SimEnc}(\mathsf{st}_{\mathsf{RNCE}}, t^*, 1^{L_E})$, and returns $(r^*, t^*, C^*_{\mathsf{RNCE}})$ without generating E_β.

4. Post-challenge key queries are answered as in Hybrid 1. On the valid query (S, t^*), the challenger first generates and withholds $\mathsf{sk}_{\mathsf{IPFE}} \leftarrow \mathsf{IPFE.KeyGen}(\mathsf{msk}_{\mathsf{IPFE}}, t^*, \mathbf{f}_S)$, then generates the inner ciphertext E_β from the stored challenge tuple as in step 3 of Hybrid 1. It obtains $\mathsf{dk}_{t^*} \leftarrow \mathsf{RNCE.SimOpen}(\mathsf{st}_{\mathsf{RNCE}}, \tau, t^*, \langle E_\beta \rangle)$ and returns $(\mathsf{sk}_{\mathsf{IPFE}}, \mathsf{dk}_{t^*})$. If no key at t^* is queried, E_β is never generated. All admissibility checks and the winning condition remain unchanged.

Hybrid 3: It is the same as Hybrid 2, except that the challenger runs the ideal tagged IPFE experiment. The challenge-epoch functional key is generated before the inner challenge in both types. Whenever the inner challenge is generated for a fixed set S, the challenger computes $\mathbf{y}_S := \mathsf{Digest}(\mathsf{pp}, S)$, sets $\mathbf{f}_S := (\mathbf{y}_S^\top, 1)^\top$, and writes $r^* = b_1 \cdots b_k$. It samples $\mathbf{s}_0, \ldots, \mathbf{s}_k \leftarrow \mathbb{Z}_q^n$, $\mathbf{e}_j \leftarrow \chi^{2m}$ for every $j < k$, $\mathbf{e}_k \leftarrow \chi^m$, and $\bar{e} \leftarrow \bar{\chi}$. It sets $\mathbf{D}_j := [(1-b_{j+1})\mathbf{G} \mid b_{j+1}\mathbf{G}]$ and $\mathbf{c}_j^\top := \mathbf{s}_j^\top \mathbf{A} + \mathbf{s}_{j+1}^\top \mathbf{D}_j + \mathbf{e}_j^\top$ for every $j < k$, and $\mathbf{c}_k^\top := \mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top$. It sets $d_0 := \mathbf{s}_0^\top \mathbf{y}_S + \bar{e} + \lfloor q/2 \rfloor m_\beta$, invokes the tagged IPFE ciphertext simulator on t^*, the IPFE plaintext length, and the singleton list containing $(\mathbf{f}_S, d_0)$ to obtain C^*_{IPFE}, and sets $E_0 := (r^*, t^*, \mathbf{c}_0, \ldots, \mathbf{c}_k, C^*_{\mathsf{IPFE}})$.

1. The challenger samples $\mathbf{A}_0, \mathbf{A}_1, \mathbf{B} \leftarrow \mathbb{Z}_q^{n \times m}$ and $\mathbf{y}_\perp \leftarrow \mathbb{Z}_q^n$, sets $\mathbf{A} := [\mathbf{A}_0 \mid \mathbf{A}_1]$, samples $\mathbf{x}^\star \leftarrow \{0,1\}^m$, and sets $\mathbf{y}^\star := \mathbf{B}\mathbf{x}^\star$. It uses the setup algorithm of the tagged IPFE simulator to obtain $(\mathsf{mpk}_{\mathsf{IPFE}}, \mathsf{st}_{\mathsf{IPFE}})$. In Type 1 it generates $(\mathsf{npk}, \mathsf{st}_{\mathsf{RNCE}}) \leftarrow \mathsf{RNCE.SimSetup}(1^\lambda)$, while in Type 2 it generates $(\mathsf{npk}, \mathsf{nmsk}) \leftarrow \mathsf{RNCE.Setup}(1^\lambda)$. It sets $\mathsf{pp} := (\mathbf{A}, \mathbf{B}, \mathbf{G}, \mathbf{y}_\perp, \mathbf{y}^\star, \mathbf{x}^\star, \mathsf{mpk}_{\mathsf{IPFE}}, \mathsf{npk})$, samples $\beta \leftarrow \{0,1\}$, and gives pp to $\mathcal{A}$.

2. For every key query (S, t) made before the outer challenge, the challenger computes $\mathbf{y}_S \leftarrow \mathsf{Digest}(\mathsf{pp}, S)$ and $\mathbf{f}_S := (\mathbf{y}_S^\top, 1)^\top$. It obtains $\mathsf{sk}_{\mathsf{IPFE}}$ from the tagged IPFE pre-challenge key simulator on $(t, \mathbf{f}_S)$. In Type 1 it generates $\mathsf{dk}_t \leftarrow \mathsf{RNCE.SimKeyGen}(\mathsf{st}_{\mathsf{RNCE}}, t)$, while in Type 2 it generates $\mathsf{dk}_t \leftarrow \mathsf{RNCE.KeyGen}(\mathsf{nmsk}, t)$. It returns $(\mathsf{sk}_{\mathsf{IPFE}}, \mathsf{dk}_t)$.

3. On challenge (r^*, t^*, m_0, m_1), the filtered game outputs an independent uniform bit unless the execution has the designated type. It aborts if more than one previous key query has epoch t^*, or if the unique such query (S, t^*) satisfies $r^* \in S$. In Type 1, it stores the challenge tuple, runs $(C^*_{\mathsf{RNCE}}, \tau) \leftarrow \mathsf{RNCE.SimEnc}(\mathsf{st}_{\mathsf{RNCE}}, t^*, 1^{L_E})$, and returns $(r^*, t^*, C^*_{\mathsf{RNCE}})$ without generating the inner challenge. In Type 2, it generates E_0 by the procedure above for the set S fixed by the unique challenge-epoch key, generates $C^*_{\mathsf{RNCE}} \leftarrow \mathsf{RNCE.Enc}(\mathsf{npk}, t^*, \langle E_0 \rangle)$, and returns $(r^*, t^*, C^*_{\mathsf{RNCE}})$.

4. Post-challenge key queries follow Hybrid 2, except that the challenger obtains each $\mathsf{sk}_{\mathsf{IPFE}}$ from the tagged IPFE simulator on $(t, \mathbf{f}_S, \perp)$, using its pre-challenge algorithm before the inner challenge and its post-challenge algorithm afterwards. On the valid Type 1 query (S, t^*), it first obtains and withholds the target IPFE key, generates E_0 by the procedure above, obtains $\mathsf{dk}_{t^*} \leftarrow \mathsf{RNCE.SimOpen}(\mathsf{st}_{\mathsf{RNCE}}, \tau, t^*, \langle E_0 \rangle)$, and returns $(\mathsf{sk}_{\mathsf{IPFE}}, \mathsf{dk}_{t^*})$. All other query handling and the winning condition remain unchanged.

For every $i \in \{0, \ldots, k+1\}$, define Hybrid 4.i as follows.

Hybrid 4.i**:** It is the same as Hybrid 3, except for the generation of the inner challenge. Hybrid 4.0 is identically distributed to Hybrid 3. Whenever $h < i \le k$, Hybrid 4.i and Hybrid 4.$(i+1)$ are identical. When the inner challenge is required for a fixed set S, compute $\mathbf{y}_S := \mathsf{Digest}(\mathsf{pp}, S)$, $\mathbf{f}_S := (\mathbf{y}_S^\top, 1)^\top$, and h and the path vectors using the challenge-path notation in Section 5.1. Write $r^* = b_1 \cdots b_k$, set $\mathbf{D}_j := [(1 - b_{j+1})\mathbf{G} \mid b_{j+1}\mathbf{G}]$ for every $j < k$, and generate E_i as follows. If $i \le h$, sample $\mathbf{c}_j \leftarrow \mathbb{Z}_q^{2m}$ and an independent auxiliary error $\mathbf{e}_j \leftarrow \chi^{2m}$ for every $j < i$. In this case, sample $\mathbf{s}_i, \ldots, \mathbf{s}_k \leftarrow \mathbb{Z}_q^n$, $\mathbf{e}_j \leftarrow \chi^{2m}$ for every $i \le j < k$, $\mathbf{e}_k \leftarrow \chi^m$, and $\bar{e} \leftarrow \bar{\chi}$. Set $\mathbf{c}_j^\top := \mathbf{s}_j^\top \mathbf{A} + \mathbf{s}_{j+1}^\top \mathbf{D}_j + \mathbf{e}_j^\top$ for every $i \le j < k$ and $\mathbf{c}_k^\top := \mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top$, and define

$$d_i := \sum_{j=0}^{i-1} (\mathbf{c}_j - \mathbf{e}_j)^\top \mathbf{z}_j + \mathbf{s}_i^\top \mathbf{y}_{\alpha_i} + \bar{e} + \lfloor q/2 \rfloor m_\beta.$$

If $i > h$ and $h < k$, instead sample $\mathbf{c}_0, \ldots, \mathbf{c}_h \leftarrow \mathbb{Z}_q^{2m}$ and $d_i \leftarrow \mathbb{Z}_q$ uniformly. In this case, sample $\mathbf{s}_{h+1}, \ldots, \mathbf{s}_k \leftarrow \mathbb{Z}_q^n$, $\mathbf{e}_j \leftarrow \chi^{2m}$ for every $h < j < k$, and $\mathbf{e}_k \leftarrow \chi^m$, and set $\mathbf{c}_j^\top := \mathbf{s}_j^\top \mathbf{A} + \mathbf{s}_{j+1}^\top \mathbf{D}_j + \mathbf{e}_j^\top$ for every $h < j < k$ and $\mathbf{c}_k^\top := \mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top$. If $i > h$ and $h = k$, sample $\mathbf{c}_0, \ldots, \mathbf{c}_{k-1} \leftarrow \mathbb{Z}_q^{2m}$, $\mathbf{c}_k \leftarrow \mathbb{Z}_q^m$, and $d_i \leftarrow \mathbb{Z}_q$ uniformly. In every case, invoke the tagged IPFE ciphertext simulator on t^*, the IPFE plaintext length, and the singleton list containing $(\mathbf{f}_S, d_i)$, obtain C^*_{IPFE}, and set $E_i := (r^*, t^*, \mathbf{c}_0, \ldots, \mathbf{c}_k, C^*_{\mathsf{IPFE}})$.

3. On challenge (r^*, t^*, m_0, m_1), the filtered game outputs an independent uniform bit unless the execution has the designated type. It aborts if more than one previous key query has epoch

t^*, or if the unique such query (S, t^*) satisfies $r^* \in S$. In Type 1, the challenger stores the challenge tuple, runs $(C^*_{\mathsf{RNCE}}, \tau) \leftarrow \mathsf{RNCE.SimEnc}(\mathsf{st}_{\mathsf{RNCE}}, t^*, 1^{L_E})$, and returns $(r^*, t^*, C^*_{\mathsf{RNCE}})$ without generating E_i. In Type 2, it generates E_i by the procedure above for the set S fixed by the unique challenge-epoch key, generates $C^*_{\mathsf{RNCE}} \leftarrow \mathsf{RNCE.Enc}(\mathsf{npk}, t^*, \langle E_i \rangle)$, and returns $(r^*, t^*, C^*_{\mathsf{RNCE}})$.

4. Post-challenge key queries are answered as in Hybrid 3. On the valid Type 1 query (S, t^*), the challenger first obtains and withholds $\mathsf{sk}_{\mathsf{IPFE}}$, generates E_i by the procedure above and opens the stored RNCE ciphertext to $\langle E_i \rangle$, and returns $(\mathsf{sk}_{\mathsf{IPFE}}, \mathsf{dk}_{t^*})$. All admissibility checks and the winning condition remain unchanged.

Hybrid 4.$(k+1)$ is independent of the challenge bit.

Indistinguishability of Hybrids.

Lemma 7.2. Assume that the tagged RNCE scheme satisfies post-challenge receiver simulatability as defined in Section 3.5. Then, for every admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $|p^{1,0}_{\mathcal{A}} - p^{1,1}_{\mathcal{A}}| \leq \mathsf{negl}(\lambda)$.

Proof. Assume that there exists an admissible PPT adversary $\mathcal{A}$ such that $|p^{1,0}_{\mathcal{A}} - p^{1,1}_{\mathcal{A}}| \geq \epsilon(\lambda)$ for some non-negligible function $\epsilon(\cdot)$. We construct a reduction algorithm $\mathcal{B}$ against post-challenge receiver simulatability of tagged RNCE.

The tagged RNCE challenger gives npk to $\mathcal{B}$. The reduction samples $\mathbf{A}_0, \mathbf{A}_1, \mathbf{B} \leftarrow \mathbb{Z}_q^{n \times m}$ and $\mathbf{y}_\perp \leftarrow \mathbb{Z}_q^n$, sets $\mathbf{A} := [\mathbf{A}_0 \mid \mathbf{A}_1]$, samples $\mathbf{x}^\star \leftarrow \{0,1\}^m$, and sets $\mathbf{y}^\star := \mathbf{B}\mathbf{x}^\star$. It generates $(\mathsf{mpk}_{\mathsf{IPFE}}, \mathsf{msk}_{\mathsf{IPFE}}) \leftarrow \mathsf{IPFE.Setup}(1^\lambda)$, samples $\beta \leftarrow \{0,1\}$, and sets $\mathsf{pp} := (\mathbf{A}, \mathbf{B}, \mathbf{G}, \mathbf{y}_\perp, \mathbf{y}^\star, \mathbf{x}^\star, \mathsf{mpk}_{\mathsf{IPFE}}, \mathsf{npk})$. It gives pp to $\mathcal{A}$. For every key query (S, t), it computes $\mathbf{y}_S$ using the tree-hashing equations, sets $\mathbf{f}_S := (\mathbf{y}_S^\top, 1)^\top$, generates $\mathsf{sk}_{\mathsf{IPFE}} \leftarrow \mathsf{IPFE.KeyGen}(\mathsf{msk}_{\mathsf{IPFE}}, t, \mathbf{f}_S)$, forwards t as a receiver-key query, and returns $(\mathsf{sk}_{\mathsf{IPFE}}, \mathsf{dk}_t)$. When $\mathcal{A}$ submits (r^*, t^*, m_0, m_1), the reduction returns an independent uniform output unless the execution is Type 1. Otherwise, write $r^* = b_1 \cdots b_k$. It samples $\mathbf{s}_0, \ldots, \mathbf{s}_k \leftarrow \mathbb{Z}_q^n$, $\mathbf{e}_j \leftarrow \chi^{2m}$ for every $j < k$, $\mathbf{e}_k \leftarrow \chi^m$, and $\bar{e} \leftarrow \bar{\chi}$. It sets $\mathbf{D}_j := [(1-b_{j+1})\mathbf{G} \mid b_{j+1}\mathbf{G}]$ and $\mathbf{c}_j^\top := \mathbf{s}_j^\top \mathbf{A} + \mathbf{s}_{j+1}^\top \mathbf{D}_j + \mathbf{e}_j^\top$ for every $j < k$, sets $\mathbf{c}_k^\top := \mathbf{s}_k^\top \mathbf{B} + \mathbf{e}_k^\top$, and generates $C^*_{\mathsf{IPFE}} \leftarrow \mathsf{IPFE.Enc}(\mathsf{mpk}_{\mathsf{IPFE}}, t^*, (\mathbf{s}_0^\top, \bar{e} + \lfloor q/2 \rfloor m_\beta)^\top)$. It sets $E_\beta := (r^*, t^*, \mathbf{c}_0, \ldots, \mathbf{c}_k, C^*_{\mathsf{IPFE}})$, submits $(t^*, \langle E_\beta \rangle)$ as its tagged RNCE challenge, and returns $(r^*, t^*, C^*_{\mathsf{RNCE}})$ to $\mathcal{A}$. It continues to answer key queries as above and outputs 1 if and only if $\mathcal{A}$ outputs β.

If the tagged RNCE challenger runs the real experiment, the reduction produces the Type 1 filtered Hybrid 0. In the ideal experiment, the challenge ciphertext is generated before E_β is used, and the later receiver key for t^* is opened to $\langle E_\beta \rangle$, which gives the Type 1 filtered Hybrid 1. No receiver key for t^* was queried before the RNCE challenge. The reduction is therefore admissible and has advantage $|p^{1,0}_{\mathcal{A}} - p^{1,1}_{\mathcal{A}}|$, contradicting post-challenge receiver simulatability. □

Lemma 7.3. For every adversary $\mathcal{A}$ of unbounded running time, $p^{1,1}_{\mathcal{A}} = p^{1,2}_{\mathcal{A}}$.

Proof. In Hybrid 1, the simulated RNCE ciphertext depends only on L_E, and the coins used to generate E_β are independent of key generation. They may therefore be sampled when the challenge-epoch key is queried, which gives Hybrid 2. □

Lemma 7.4. Assume that the tagged IPFE scheme satisfies 1-adaptive simulation security as in Definition 3.6. Then, for every $g \in \{1,2\}$ and every admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $|p^{g,2}_{\mathcal{A}} - p^{g,3}_{\mathcal{A}}| \leq \mathsf{negl}(\lambda)$.

Proof. Assume that an admissible PPT adversary $\mathcal{A}$ distinguishes the two hybrids with non-negligible advantage. We construct a reduction $\mathcal{B}$ against the adaptive simulation security of tagged IPFE.

The reduction receives $\mathsf{mpk}_{\mathsf{IPFE}}$ from its challenger and generates the remaining public parameters as in Hybrid 2, including $\mathbf{x}^\star \leftarrow \{0,1\}^m$. It runs the RNCE algorithms according to the designated type and forwards every functional-key query $(t, \mathbf{f}_S)$ to its IPFE challenger. When the inner challenge is required, it generates the lattice components as in Hybrid 2 and submits $(t^*, (\mathbf{s}_0^\top, \bar{e} + \lfloor q/2 \rfloor m_\beta)^\top)$ as its IPFE challenge. In Type 1, it obtains and withholds the target IPFE key before submitting this challenge. It delivers the resulting inner ciphertext through RNCE as in Hybrid 2, continues to answer key queries, and outputs 1 if and only if $\mathcal{A}$ outputs β. If no inner challenge is required, it runs the common experiment.

The real IPFE experiment gives Hybrid 2. In the ideal experiment, the unique challenge-tag key precedes the inner challenge, and its function value is $\mathbf{s}_0^\top \mathbf{y}_S + \bar{e} + \lfloor q/2 \rfloor m_\beta$, which gives Hybrid 3. Every non-challenge-tag key is answered by the corresponding simulator without a function value. The reduction makes at most one functional-key query at the challenge tag, so its advantage is $|p_{\mathcal{A}}^{g,2} - p_{\mathcal{A}}^{g,3}|$, contradicting adaptive simulation security. □

Lemma 7.5. For every $g \in \{1,2\}$ and every admissible adversary $\mathcal{A}$ of unbounded running time, $p_{\mathcal{A}}^{g,3} = p_{\mathcal{A}}^{g,4.0}$.

Proof. At $i = 0$, no ciphertext component is sampled uniformly, and the sum in d_i is empty. Since $\alpha_0 = \epsilon$ and $\mathbf{y}_{\alpha_0} = \mathbf{y}_S$, the function value is $d_0 = \mathbf{s}_0^\top \mathbf{y}_S + \bar{e} + \lfloor q/2 \rfloor m_\beta$. Thus the inner challenge and all query answers have exactly the distribution in Hybrid 3. □

Lemma 7.6. Assume that $\mathsf{elLWE}_{n,2m+1,q,\chi,\bar{\chi},1}$ and $\mathsf{elLWE}_{n,m+1,q,\chi,\bar{\chi},1}$ hold. For every $g \in \{1,2\}$, every $i \in \{0,\ldots,k\}$, and every admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $|p_{\mathcal{A}}^{g,4.i} - p_{\mathcal{A}}^{g,4.(i+1)}| \le \mathsf{negl}(\lambda)$.

Proof. Fix $g \in \{1,2\}$. Assume that there exists an admissible PPT adversary $\mathcal{A}$ for which the difference is non-negligible. We construct a two-stage reduction algorithm $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against error-leakage LWE. On an execution of the other type, or on a Type 1 execution with no challenge-epoch key query, it submits the zero leakage vector, ignores the elLWE sample, and returns the output prescribed by the filtered game.

First suppose that $i < k$. The elLWE challenger gives $\mathbf{Q} \leftarrow \mathbb{Z}_q^{n \times (2m+1)}$ to $\mathcal{B}_1$, which parses $\mathbf{Q} = [\mathbf{A} \mid \mathbf{y}_\perp]$ and $\mathbf{A} = [\mathbf{A}_0 \mid \mathbf{A}_1]$. It samples $\mathbf{B} \leftarrow \mathbb{Z}_q^{n \times m}$ and $\mathbf{x}^\star \leftarrow \{0,1\}^m$, sets $\mathbf{y}^\star := \mathbf{B}\mathbf{x}^\star$, and runs the setup algorithm of the tagged IPFE simulator. If $g = 1$, it uses $\mathsf{RNCE.SimSetup}$, returns the output of $\mathsf{RNCE.SimEnc}$ at the outer challenge, and retains the opening until the target key query. If $g = 2$, it runs the honest RNCE algorithms. It sets the remaining public parameters exactly as in the construction, samples $\beta \leftarrow \{0,1\}$, and gives pp to $\mathcal{A}$. It answers every key query using the tagged IPFE simulator and the RNCE scheduling specified for the corresponding type. The reduction runs $\mathcal{A}$ until both the challenge tuple (r^*, t^*, m_0, m_1) and the challenge-epoch set S are fixed. In a Type 1 execution, it generates and stores the target inner key and pauses the key query before producing the inner challenge. It computes $\mathbf{f}_S := (\mathbf{y}_S^\top, 1)^\top$, h, and the path vectors. If $h < i$, the adjacent hybrids are identical, so $\mathcal{B}_1$ submits the zero leakage vector and $\mathcal{B}_2$ ignores its elLWE sample and runs their common experiment. Suppose that $h \ge i$. If $h > i$, $\mathcal{B}_1$ submits $\mathbf{z}_i^{\mathsf{leak}} := [-\mathbf{z}_i^\top, 0]^\top$. If $h = i$, it submits $\mathbf{z}_i^{\mathsf{leak}} := (0,\ldots,0,-1)^\top$. Both vectors have infinity norm at most 1. The challenger returns $(\mathbf{v}^\top, u, l)$, where $(\mathbf{v}^\top, u) \in \mathbb{Z}_q^{2m} \times \mathbb{Z}_q$. The reduction samples $\mathbf{c}_0, \ldots, \mathbf{c}_{i-1}$ uniformly with independent auxiliary errors and samples all secrets and errors after layer i as in Hybrid 4.i. It does not separately sample the layer-i error or $\bar{e}$, whose required combination is the elLWE leakage

l. If $h > i$, it sets $\mathbf{c}_i^\top := \mathbf{v}^\top + \mathbf{s}_{i+1}^\top \mathbf{D}_i, d := \sum_{j=0}^{i-1}(\mathbf{c}_j - \mathbf{e}_j)^\top \mathbf{z}_j + \mathbf{v}^\top \mathbf{z}_i + l + \lfloor q/2 \rfloor m_\beta$. If $h = i$, it uses the same definition of $\mathbf{c}_i$ and sets $d := \sum_{j=0}^{i-1}(\mathbf{c}_j - \mathbf{e}_j)^\top \mathbf{z}_j + u + l + \lfloor q/2 \rfloor m_\beta$. It generates every later component honestly, gives $(\mathbf{f}_S, d)$ to the tagged IPFE ciphertext simulator, and sets $E := (r^*, t^*, \mathbf{c}_0, \ldots, \mathbf{c}_k, C^*_{\mathsf{IPFE}})$. In Type 1 it opens the stored RNCE ciphertext to $\langle E \rangle$. In Type 2 it encrypts $\langle E \rangle$ under tagged RNCE. It then completes all remaining key queries and outputs 1 if and only if $\mathcal{A}$ outputs β. Now let $i = k$. The reduction receives $\mathbf{Q} \leftarrow \mathbb{Z}_q^{n \times (m+1)}$, parses $\mathbf{Q} = [\mathbf{B} \mid \mathbf{y}_\perp]$, samples $\mathbf{A}_0, \mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n \times m}$ and $\mathbf{x}^\star \leftarrow \{0,1\}^m$, sets $\mathbf{A} := [\mathbf{A}_0 \mid \mathbf{A}_1]$, and sets $\mathbf{y}^\star := \mathbf{B}\mathbf{x}^\star$. It runs the IPFE and RNCE algorithms as above until the challenge tuple and S are fixed, withholding the Type 1 target key when necessary, and computes h. If $h < k$, it submits the zero leakage vector and runs the common experiment since Hybrids 4.k and 4.$(k+1)$ are identical. Suppose that $h = k$. It submits $(0, \ldots, 0, -1)^\top$, receives $(\mathbf{v}^\top, u, l) \in \mathbb{Z}_q^m \times \mathbb{Z}_q \times \mathbb{Z}$, samples $\mathbf{c}_j \leftarrow \mathbb{Z}_q^{2m}$ and $\mathbf{e}_j \leftarrow \chi^{2m}$ independently for every $j < k$, sets $\mathbf{c}_k^\top := \mathbf{v}^\top$, and uses $d := \sum_{j=0}^{k-1}(\mathbf{c}_j - \mathbf{e}_j)^\top \mathbf{z}_j + u + l + \lfloor q/2 \rfloor m_\beta$. All remaining key queries and the outer delivery are generated as above.

If the elLWE challenger uses an LWE sample and $h > i$, then $\mathbf{v}^\top = \mathbf{s}_i^\top \mathbf{A} + \mathbf{e}_i^\top$ and $l = -\mathbf{e}_i^\top \mathbf{z}_i + \bar{e}$. Hence $\mathbf{v}^\top \mathbf{z}_i + l = \mathbf{s}_i^\top \mathbf{y}_{\alpha_i} + \bar{e}$, and the experiment is Hybrid 4.i. If it uses a uniform sample, then $\mathbf{c}_i$ is uniform and $\mathbf{v}^\top \mathbf{z}_i + l = (\mathbf{c}_i - \mathbf{e}_i)^\top \mathbf{z}_i + \mathbf{s}_{i+1}^\top \mathbf{y}_{\alpha_{i+1}} + \bar{e}$, where $\mathbf{e}_i$ and $\bar{e}$ are the errors defining l, and we used $\mathbf{D}_i \mathbf{z}_i = -\mathbf{y}_{\alpha_{i+1}}$. This is Hybrid 4.$(i+1)$. If $h = i$, the real elLWE sample satisfies $u = \mathbf{s}_i^\top \mathbf{y}_\perp + \widehat{e}_i$ and $l = -\widehat{e}_i + \bar{e}$, so $u + l = \mathbf{s}_i^\top \mathbf{y}_{\alpha_i} + \bar{e}$ and the experiment is Hybrid 4.i. In the uniform case, $\mathbf{c}_i$ and d are uniform, which gives Hybrid 4.$(i+1)$. For $i = k$ and $h = k$, the same argument applies. In the real case, $u + l = \mathbf{s}_k^\top \mathbf{y}_\perp + \bar{e}$, which gives Hybrid 4.$k$. In the uniform case, both $\mathbf{c}_k$ and d are uniform, which gives Hybrid 4.$(k+1)$. In every nontrivial case, the real elLWE sample gives Hybrid 4.i and the uniform sample gives Hybrid 4.$(i+1)$. Possibly complementing its output, $\mathcal{B}$ has elLWE advantage one half of the adjacent-hybrid gap, which contradicts the corresponding elLWE assumption. □

Lemma 7.7. For every $g \in \{1, 2\}$ and every admissible adversary $\mathcal{A}$ of unbounded running time, $p_{\mathcal{A}}^{g,4.(k+1)} = 1/2$.

Proof. Since $k + 1 > h$, the value d_{k+1} given to the IPFE ciphertext simulator is uniform over $\mathbb{Z}_q$, and the remaining inner-challenge components do not depend on β. The RNCE ciphertext and receiver keys therefore do not depend on β either. In Type 1 with no challenge-epoch key query, no inner challenge is generated and the simulated RNCE ciphertext depends only on L_E. In every case the adversary's view is independent of β, giving success probability 1/2. □

Proof of Theorem 7.1. The error distributions above satisfy both elLWE assumptions by Corollary 3.3. For the Type 1 filtered game, apply Lemma 7.2 and Lemma 7.3. For the Type 2 filtered game, Hybrid 2 is the real experiment. For each $g \in \{1, 2\}$, apply Lemma 7.4 and Lemma 7.5, followed by Lemma 7.6 for every $i \in \{0, \ldots, k\}$. Lemma 7.7 gives $p_{\mathcal{A}}^{g,4.(k+1)} = 1/2$, so $|p_{\mathcal{A}}^{g,0} - 1/2|$ is negligible for both g. The decomposition at the beginning of the proof gives $|p_{\mathcal{A}}^{\mathsf{real}} - 1/2| \leq \mathsf{negl}(\lambda)$. □

Corollary 7.8. Assuming standard LWE with a polynomial modulus-to-noise ratio, there exists an adaptively secure batched IBE scheme in the random oracle model.

Proof. For the stated choice of q, the outer LWE modulus-to-noise ratio is $q/(2\sigma/\sqrt{17}) = \Theta((K+1)n^3) = \mathsf{poly}(\lambda)$. Instantiate tagged IPFE using Corollary 3.9 and tagged RNCE using Corollary 3.10, with their internal lattice parameters chosen independently. The claim follows from Theorem 7.1. □

7.2 Ring-Based Variant

We describe how to use tagged IPFE for the ring inner product needed by a ring-based LE instantiation [DKL+23, Section 12]. Let $R_q = \mathbb{Z}_q[X]/(X^N+1)$, where N is a power of two and polynomially bounded in λ. Choose a prime q such that the defining polynomial splits into distinct linear factors over $\mathbb{Z}_q$. The number theoretic transform evaluates a ring element at these roots. For plaintext and function vectors $\mathbf{x}, \mathbf{f} \in R_q^{n+1}$, we have $\langle \mathbf{x}, \mathbf{f} \rangle(\omega) = \sum_{j=1}^{n+1} x_j(\omega) f_j(\omega)$ at each root ω. Thus a ring inner product becomes an ordinary inner product at each evaluation point.

We therefore use N independent tagged-IPFE instances over $\mathbb{Z}_q$, each of dimension $n+1$, to encrypt the evaluated plaintext vectors and issue keys for the evaluated function vectors. All instances use the same epoch as their tag, and the ciphertext and key consist of the corresponding tuples. Whenever the IPFE decryptions are correct, an inverse NTT recovers the ring inner product exactly. Each instance receives at most one function key at the challenge epoch. Simulation security follows by a hybrid over the instances, since the permitted ring output determines all of its evaluations.

8 Threshold Batched IBE from Threshold FHE

Let BIBE be a BIBE scheme as in Section 4.1, let TFHE be a threshold FHE scheme as in Section 3.6, and let F be a pseudorandom function. Let R be the random-tape length of $\mathsf{BIBE.KeyGen}$, and let F_s map encoded pairs $(\mathbf{d}, t)$ to $\{0,1\}^R$. For each $(\mathbf{d}, t)$, define

$$C_{\mathbf{d},t}(\mathsf{msk}, s) := \left\langle \mathsf{BIBE.KeyGen}\big(\mathsf{msk}, \mathbf{d}, t; F_s(\langle \mathbf{d}, t \rangle)\big) \right\rangle .$$

Let D bound the depth of these circuits, including the PRF computation, and let L be the length of $\langle \mathsf{msk}, s \rangle$. We apply TFHE componentwise to the output bits. Our threshold BIBE scheme is as follows.

$\mathsf{TBIBE.Setup}(1^\lambda, 1^N, \theta)$: It generates $(\mathsf{pp}_0, \mathsf{msk}) \leftarrow \mathsf{BIBE.Setup}(1^\lambda)$ and samples a PRF seed s. It runs $\mathsf{TFHE.Setup}(1^\lambda, 1^D, 1^N, \theta)$ to obtain pk_F and shares $\mathsf{sk}_{F,1}, \ldots, \mathsf{sk}_{F,N}$. It computes $\mathbf{c} \leftarrow \mathsf{TFHE.Enc}(\mathsf{pk}_F, \langle \mathsf{msk}, s \rangle)$ and outputs $\mathsf{pp} := (\mathsf{pp}_0, \mathsf{pk}_F, \mathbf{c}, N, \theta)$ and $\mathsf{msk}_i := \mathsf{sk}_{F,i}$ for $i \in [N]$.

$\mathsf{TBIBE.Digest}(\mathsf{pp}, S)$: It returns $\mathsf{BIBE.Digest}(\mathsf{pp}_0, S)$.

$\mathsf{TBIBE.KeyShare}(\mathsf{pp}, \mathsf{msk}_i, \mathbf{d}, t)$: It constructs $C_{\mathbf{d},t}$, computes $\widehat{\mathbf{c}} := \mathsf{TFHE.Eval}(\mathsf{pk}_F, C_{\mathbf{d},t}, \mathbf{c})$, and returns $\sigma_i \leftarrow \mathsf{TFHE.PartDec}(\mathsf{pk}_F, \widehat{\mathbf{c}}, \mathsf{msk}_i)$.

$\mathsf{TBIBE.Combine}(\mathsf{pp}, \mathbf{d}, t, \{(i, \sigma_i)\}_{i \in J})$: It outputs $\perp$ if $|J| < \theta$. Otherwise, it computes $z \leftarrow \mathsf{TFHE.FinDec}(\mathsf{pk}_F, \{(i, \sigma_i)\}_{i \in J})$ and returns the batch key encoded by z, or $\perp$ if decoding fails.

$\mathsf{TBIBE.Enc}(\mathsf{pp}, r, t, m)$: It returns $\mathsf{BIBE.Enc}(\mathsf{pp}_0, r, t, m)$.

$\mathsf{TBIBE.Dec}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct}, S, r, t)$: It returns $\mathsf{BIBE.Dec}(\mathsf{pp}_0, \mathsf{sk}, \mathsf{ct}, S, r, t)$.

Correctness. TFHE correctness implies that honestly generated shares from any J with $|J| \geq \theta$ combine to $\mathsf{BIBE.KeyGen}(\mathsf{msk}, \mathbf{d}, t; F_s(\langle \mathbf{d}, t \rangle))$, except with negligible probability. Replacing the PRF value by a uniform random tape changes the decryption success probability negligibly. Correctness then follows from that of BIBE.

Security. We give the security analysis as follows.

Theorem 8.1. Assume that BIBE satisfies adaptive security, that TFHE satisfies the security property in Section 3.6, and that F is a secure pseudorandom function. Then the construction above satisfies adaptive security against static corruptions as in Section 4.2.

Proof. We prove the theorem through the following hybrids. Let $(\mathsf{Sim}_0, \mathsf{Sim}_1)$ be the TFHE simulator. Changes from the preceding hybrid are marked in red. All games use the admissibility conditions in Section 4.2.

Hybrid 0: This is the real adaptive security experiment in Section 4.2. The adversary first outputs (N, θ, I). The challenger runs $\mathsf{TBIBE.Setup}(1^\lambda, 1^N, \theta)$, samples $\beta \leftarrow \{0,1\}$, and gives pp and $\{\mathsf{sk}_{F,i}\}_{i\in I}$ to $\mathcal{A}$. For every pre-challenge share query (S,t), it computes $\mathbf{d} \coloneqq \mathsf{BIBE.Digest}(\mathsf{pp}_0, S)$ and returns $\{\sigma_i \leftarrow \mathsf{TBIBE.KeyShare}(\mathsf{pp}, \mathsf{sk}_{F,i}, \mathbf{d}, t)\}_{i\in[N]\setminus I}$. On challenge (r^*, t^*, m_0, m_1), it returns $\mathsf{ct}^* \leftarrow \mathsf{BIBE.Enc}(\mathsf{pp}_0, r^*, t^*, m_\beta)$. Post-challenge share queries are answered in the same way. The adversary outputs β' and wins if and only if $\beta' = \beta$.

Hybrid 1: It is the same as Hybrid 0, except that the challenger uses the TFHE simulator. After receiving (N, θ, I), it generates $(\mathsf{pp}_0, \mathsf{msk}) \leftarrow \mathsf{BIBE.Setup}(1^\lambda)$ and samples s as before. It runs $\mathsf{Sim}_0(1^\lambda, 1^D, 1^N, \theta, 1^L)$ to obtain $(\mathsf{pk}_F, \mathbf{c}, \mathsf{sk}_{F,1}, \ldots, \mathsf{sk}_{F,N}, \mathsf{st})$. It sets $\mathsf{pp} \coloneqq (\mathsf{pp}_0, \mathsf{pk}_F, \mathbf{c}, N, \theta)$, samples $\beta \leftarrow \{0,1\}$, and gives pp and $\{\mathsf{sk}_{F,i}\}_{i\in I}$ to $\mathcal{A}$. For every share query (S,t), before or after the challenge, it computes $\mathbf{d} \coloneqq \mathsf{BIBE.Digest}(\mathsf{pp}_0, S)$ and $z \coloneqq C_{\mathbf{d},t}(\mathsf{msk}, s)$, and returns the shares generated by $\mathsf{Sim}_1(\mathsf{st}, C_{\mathbf{d},t}, [N] \setminus I, z)$. The simulator is applied componentwise to the output bits. The challenge ciphertext and winning condition remain unchanged.

Hybrid 2: It is the same as Hybrid 1, except for the value supplied to Sim_1. On the first query with a given $(\mathbf{d}, t)$, the challenger samples $\rho_{\mathbf{d},t} \leftarrow \{0,1\}^R$ uniformly and stores $\mathsf{sk}_{\mathbf{d},t} \coloneqq \mathsf{BIBE.KeyGen}(\mathsf{msk}, \mathbf{d}, t; \rho_{\mathbf{d},t})$. On this and every later query with the same $(\mathbf{d}, t)$, it sets $z \coloneqq \langle \mathsf{sk}_{\mathbf{d},t} \rangle$ and invokes Sim_1 as in Hybrid 1. The circuit $C_{\mathbf{d},t}$ passed to the simulator is unchanged.

Lemma 8.2. Assume that TFHE satisfies the security property in Section 3.6. Then, for every admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $|\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}}| \le \mathsf{negl}(\lambda)$.

Proof. Suppose that an admissible PPT adversary $\mathcal{A}$ distinguishes the two hybrids with non-negligible advantage. We construct a reduction $\mathcal{B}$ against TFHE security.

The reduction obtains (N, θ, I) from $\mathcal{A}$, generates $(\mathsf{pp}_0, \mathsf{msk}) \leftarrow \mathsf{BIBE.Setup}(1^\lambda)$, and samples a PRF seed s and $\beta \leftarrow \{0,1\}$. It submits (N, D, θ, x, I) to its TFHE challenger, where $x \coloneqq \langle \mathsf{msk}, s \rangle$. Upon receiving $(\mathsf{pk}_F, \mathbf{c}, \{\mathsf{sk}_{F,i}\}_{i\in I})$, it gives $\mathsf{pp} \coloneqq (\mathsf{pp}_0, \mathsf{pk}_F, \mathbf{c}, N, \theta)$ and the corrupted shares to $\mathcal{A}$. For every share query (S,t), it computes $\mathbf{d} \coloneqq \mathsf{BIBE.Digest}(\mathsf{pp}_0, S)$, submits $(C_{\mathbf{d},t}, [N] \setminus I)$ componentwise to the TFHE challenger, and returns the resulting shares. It generates the challenge ciphertext using $\mathsf{BIBE.Enc}(\mathsf{pp}_0, r^*, t^*, m_\beta)$, continues to answer share queries in the same way, and outputs 1 if and only if $\beta' = \beta$.

The real TFHE experiment gives Hybrid 0. In the ideal experiment, setup is simulated by Sim_0 and each query is answered by Sim_1 using $C_{\mathbf{d},t}(\mathsf{msk}, s)$, which gives Hybrid 1. Since $|I| < \theta$ and all queried circuits have depth at most D, the reduction is admissible. Its distinguishing advantage is $|\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}}|$, contradicting TFHE security. □

Lemma 8.3. Assume that F is a secure pseudorandom function. Then, for every admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $|\mathsf{Adv}^1_{\mathcal{A}} - \mathsf{Adv}^2_{\mathcal{A}}| \le \mathsf{negl}(\lambda)$.

Proof. Suppose that an admissible PPT adversary $\mathcal{A}$ distinguishes the two hybrids with non-negligible advantage. We construct a reduction $\mathcal{B}$ against the pseudorandomness of F, with oracle $\mathcal{O}$ that is either F_s for a random seed s or a random function with the same domain and range.

The reduction generates the BIBE keys and runs Sim_0 as in Hybrid 1. It samples $\beta \leftarrow \{0,1\}$ and gives the public parameters and corrupted shares to $\mathcal{A}$. On a share query (S,t), it computes $\mathbf{d} := \mathsf{BIBE.Digest}(\mathsf{pp}_0, S)$. If no key is stored for $(\mathbf{d},t)$, it queries $\rho \leftarrow \mathcal{O}(\langle \mathbf{d},t\rangle)$ and stores $\mathsf{sk}_{\mathbf{d},t} := \mathsf{BIBE.KeyGen}(\mathsf{msk},\mathbf{d},t;\rho)$. It returns the shares generated by $\mathsf{Sim}_1(\mathsf{st}, C_{\mathbf{d},t}, [N]\setminus I, \langle \mathsf{sk}_{\mathbf{d},t}\rangle)$. It generates the challenge ciphertext honestly, answers subsequent share queries in the same way, and outputs 1 if and only if $\beta' = \beta$.

The simulated setup and the descriptions of $C_{\mathbf{d},t}$ do not require the seed s. If $\mathcal{O} = F_s$, the stored key is the value encoded by $C_{\mathbf{d},t}(\mathsf{msk}, s)$, so the view is Hybrid 1. If $\mathcal{O}$ is a random function, each new $(\mathbf{d},t)$ uses an independent uniform random tape and repeated queries use the stored key, giving Hybrid 2. Thus the reduction has advantage $|\mathsf{Adv}^1_{\mathcal{A}} - \mathsf{Adv}^2_{\mathcal{A}}|$, contradicting the pseudorandomness of F. □

Lemma 8.4. Assume that BIBE satisfies adaptive security as in Section 4.1. Then, for every admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\mathsf{Adv}^2_{\mathcal{A}} \le 1/2 + \mathsf{negl}(\lambda)$.

Proof. Suppose that an admissible PPT adversary $\mathcal{A}$ wins Hybrid 2 with probability at least $1/2 + \epsilon(\lambda)$ for some non-negligible function $\epsilon(\cdot)$. We construct an adversary $\mathcal{B}$ against the adaptive security of BIBE.

The reduction receives pp_0 from its BIBE challenger and obtains (N,θ,I) from $\mathcal{A}$. It runs Sim_0 and gives the resulting public parameters and corrupted shares to $\mathcal{A}$ as in Hybrid 1. On a share query (S,t), it computes $\mathbf{d} := \mathsf{BIBE.Digest}(\mathsf{pp}_0, S)$. If no key is stored for $(\mathbf{d},t)$, it submits (S,t) to its key oracle and stores the returned key $\mathsf{sk}_{\mathbf{d},t}$. It answers with $\mathsf{Sim}_1(\mathsf{st}, C_{\mathbf{d},t}, [N]\setminus I, \langle \mathsf{sk}_{\mathbf{d},t}\rangle)$. It forwards $\mathcal{A}$'s challenge tuple to its challenger, returns the challenge ciphertext, and answers subsequent share queries in the same way. Finally, it outputs $\mathcal{A}$'s guess.

Each first occurrence of $(\mathbf{d},t)$ is answered by the BIBE key oracle using fresh uniform randomness, and later occurrences use the stored key. The view is therefore identically distributed to Hybrid 2. Every key query made by $\mathcal{B}$ comes from a share query of $\mathcal{A}$, so at most one query has epoch t^*, and its batch excludes r^*. Thus $\mathcal{B}$ is admissible and wins with probability at least $1/2 + \epsilon(\lambda)$, contradicting the adaptive security of BIBE. □

By Lemma 8.2 and Lemma 8.3, the winning probabilities in Hybrids 0 and 2 differ negligibly. By Lemma 8.4, $\mathsf{Adv}^2_{\mathcal{A}} \le 1/2 + \mathsf{negl}(\lambda)$. Since Hybrid 0 is the real security experiment, the theorem follows. □

Remark 8.5 (Random-oracle calls)**.** The TFHE evaluator takes ordinary circuits without random-oracle gates. In the construction of Section 7, the random-oracle queries made during IPFE key generation depend only on the public epoch t, not on msk or the secret random tape [ET25, Section 4.2]. Each party can evaluate these queries in advance and hardwire the answers into $C_{\mathbf{d},t}$. The random oracle in the RNCE instantiation [GSW21, Appendix A.1] is used for encryption and decryption, which remain outside the TFHE evaluation. Thus the construction also applies to Section 7, with an oracle-free key-generation circuit.

9 Implementation and Evaluation

Implementation. We implement a non-threshold prototype combining ordinary-LWE laconic encryption [DKL+23], ring IPFE, and tagged RNCE [GSW21]. The IPFE design is adapted from [BS25]. We use OpenFHE for LE and PALISADE for the IBE component. We ran the

benchmarks on a Mac with an Apple M5 chip and 24 GB of RAM. IPFE key generation uses eight worker threads. LE and IBE operations are sequential.

Parameters. We consider 32-bit identity for batched IBE. The parameters target approximately 128-bit classical security. LE uses $p = 1048573$, $n = 864$, and $m = 17280$, with public matrices uniform modulo p. The IPFE inputs have length 865 over $\mathbb{Z}_p$. Its internal ring is $\mathbb{Z}_Q[X]/(X^{8192}+1)$, with $Q = p^8$ and message scale $\Delta = p^7$.

Benchmarks. Table 2 reports the costs of the combined LE–IPFE–RNCE prototype. The experiment uses a batch of 16 identities and a single batch key. Each ciphertext encrypts one bit. We generate one fresh ciphertext for each identity and report mean encryption and decryption times over these 16 ciphertexts. Decryption excludes constructing the hash tree, which is reported separately.

Operation	Time
IPFE key generation	134.38 s
RNCE receiver-key generation	5.89 s
Encryption	13.13 s
Tree construction, 16 IDs	18.23 s
Decryption, including witness preparation	2.65 s

Table 2: Running times for a batch of 16 identities. Encryption and decryption are per-ciphertext averages.

Sizes. Table 3 reports sizes in decimal units. The batch key includes both the IPFE functional key and the RNCE receiver key.

Object	Size
Digest	2.59 KB
Batch key	$\approx$ 79.11 MB
Ciphertext	303.89 MB

Table 3: Space costs of the prototype. Digest and batch-key sizes are encoding estimates. The ciphertext size is a measured file size.

Threshold construction. We also implement a threshold prototype using EMP garbling and the threshold IBE of [BdLP26], with a threshold of three out of five parties. We also consider the following optimizations for our implementation:

For the FE component, the encryptor garbles an inner-product circuit. The function-input labels are encrypted using threshold identity-based encryption. The IBE identities encode the epoch, group position, and candidate value. Once the batch digest is known, the authorities jointly issue IBE keys for the FE input values. The recipient uses these keys to recover the selected labels and evaluate the inner-product circuit. Key issuance only requires threshold IBE, which replaces the threshold homomorphic encryption algorithm.

We keep the FE ciphertext outside the RNCE wrapper to reduce the amount of data processed by RNCE. The wrapper includes only the ciphertext components over the path of the laconic encryption

tree. The recipient recovers these components through RNCE and obtains the inner-product value directly from FE. The remaining LE decryption steps are unchanged.

We use the same LE parameters and a length-865 inner-product circuit over $\mathbb{Z}_p$. The IBE ring has degree 4096 and modulus 576460752303439873. Setup uses a trusted dealer, and the three participating parties run sequentially in one process. In one run with a batch of 16 identities and a one-bit message, encryption takes 312.38 s and key generation takes 97.56 s. Decryption takes 7.01 s, including witness preparation and excluding tree construction, which takes 21.27 s. Encryption and decryption include the LE, FE, and RNCE stages. The ciphertext and batch key occupy 4.62 GB and 874.80 MB, respectively.

Comparison with BLT25. We tested a third-party implementation of BLT25[2] using its smallest parameter set, with $n = 4$, 4-bit identities and a batch of two. On the same machine, encryption takes about 0.06 s per ciphertext and batch decryption takes about 6 s, including the public sampling computation. These parameters are intended only for functional testing. Using the repository's parameter generator with $\lambda = 128$, 32-bit identities and a batch of 16, we estimate that one dense matrix in its sampler would require hundreds of terabytes of memory. This exceeds the memory available on our test platform, so we do not report running times for this configuration.

References

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 553–572, French Riviera, May 30 – June 3, 2010. Springer Berlin Heidelberg, Germany.

[AFP25] Amit Agarwal, Rex Fernando, and Benny Pinkas. Efficiently-thresholdizable batched identity based encryption, with applications. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part III*, volume 16002 of *Lecture Notes in Computer Science*, pages 69–100, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland.

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501, Cambridge, UK, April 15–19, 2012. Springer Berlin Heidelberg, Germany.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th Annual ACM Symposium on Theory of Computing*, pages 99–108, Philadephia, PA, USA, May 22–24, 1996. ACM Press.

[ALS16] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 333–362, Santa Barbara, CA, USA, August 14–18, 2016. Springer Berlin Heidelberg, Germany.

[2] https://github.com/pepae/blt25-implementation

[AMR25] Damiano Abram, Giulio Malavolta, and Lawrence Roy. Succinct oblivious tensor evaluation and applications: Adaptively-secure laconic function evaluation and trapdoor hashing for all circuits. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, pages 1875–1886, 2025.

[BCF+25] Jan Bormet, Arka Rai Choudhuri, Sebastian Faust, Sanjam Garg, Hussien Othman, Guru-Vamsi Policharla, Ziyan Qu, and Mingyuan Wang. BEAST-MEV: Batched threshold encryption with silent setup for MEV prevention. Cryptology ePrint Archive, Report 2025/1419, 2025.

[BdLP26] Katharina Boudgoust, Rafael del Pino, Oleksandra Lapiha, and Thomas Prest. IND-CCA lattice threshold KEM under 30 KiB. In Shi Bai and Edoardo Persichetti, editors, *PKC 2026: 29th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 16551 of *Lecture Notes in Computer Science*, pages 38–69, West Palm Beach, FL, USA, May 25–28, 2026. Springer, Cham, Switzerland.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229, Santa Barbara, CA, USA, August 19–23, 2001. Springer Berlin Heidelberg, Germany.

[BFOQ25] Jan Bormet, Sebastian Faust, Hussien Othman, and Ziyan Qu. BEAT-MEV: Epochless approach to batched threshold encryption for MEV prevention. In Lujo Bauer and Giancarlo Pellegrino, editors, *USENIX Security 2025: 34th USENIX Security Symposium*, pages 3457–3476, Seattle, WA, USA, August 13–15, 2025. USENIX Association.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 533–556, Copenhagen, Denmark, May 11–15, 2014. Springer Berlin Heidelberg, Germany.

[BGG+18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 565–596, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

[BKP13] Rikke Bendlin, Sara Krehbiel, and Chris Peikert. How to share a lattice trapdoor: Threshold protocols for signatures and (H)IBE. In Michael J. Jacobson, Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 2013: 11th International Conference on Applied Cryptography and Network Security*, volume 7954 of *Lecture Notes in Computer Science*, pages 218–236, Banff, AB, Canada, June 25–28, 2013. Springer Berlin Heidelberg, Germany.

[BLT25] Dan Boneh, Evan Laufer, and Ertem Nusret Tas. Batch decryption without epochs and its application to encrypted mempools. Cryptology ePrint Archive, Report 2025/1254, 2025.

[BO22] Joseph Bebel and Dev Ojha. Ferveo: Threshold decryption for mempool privacy in BFT networks. Cryptology ePrint Archive, Report 2022/898, 2022.

[BS25] Anushree Belel and Junji Shikata. Efficient identity-based inner product functional encryption from RLWE. In Ratna Dutta, Luca De Feo, and Sugata Gangopadhyay, editors, *Progress in Cryptology - INDOCRYPT 2025: 26th International Conference in Cryptology in India*, volume 16372 of *Lecture Notes in Computer Science*, pages 285–311, Bhubaneshwar, India, December 14–17, 2025. Springer, Cham, Switzerland.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273, Providence, RI, USA, March 28–30, 2011. Springer Berlin Heidelberg, Germany.

[CGPP24] Arka Rai Choudhuri, Sanjam Garg, Julien Piet, and Guru-Vamsi Policharla. Mempool privacy via batched threshold encryption: Attacks and defenses. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024: 33rd USENIX Security Symposium*, Philadelphia, PA, USA, August 14–16, 2024. USENIX Association.

[CGPW25] Arka Rai Choudhuri, Sanjam Garg, Guru-Vamsi Policharla, and Mingyuan Wang. Practical mempool privacy via one-time setup batched threshold encryption. In Lujo Bauer and Giancarlo Pellegrino, editors, *USENIX Security 2025: 34th USENIX Security Symposium*, pages 3477–3495, Seattle, WA, USA, August 13–15, 2025. USENIX Association.

[CW23] Valerio Cini and Hoeteck Wee. ABE for circuits with poly (λ)-sized keys from LWE. In *64th Annual Symposium on Foundations of Computer Science*, pages 435–446, Santa Cruz, CA, USA, November 6–9, 2023. IEEE Computer Society Press.

[CW24] Valerio Cini and Hoeteck Wee. Unbounded ABE for circuits from LWE, revisited. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024, Part IV*, volume 15487 of *Lecture Notes in Computer Science*, pages 238–267, Kolkata, India, December 9–13, 2024. Springer, Singapore, Singapore.

[DGK+20] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy*, pages 910–927, San Francisco, CA, USA, May 18–21, 2020. IEEE Computer Society Press.

[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 Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part III*, volume 14006 of *Lecture Notes in Computer Science*, pages 417–446, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[ET25] Tenma Edamura and Atsushi Takayasu. Tight adaptive simulation security for identity-based inner-product FE in the (quantum) random oracle model. In Tibor Jager and Jiaxin Pan, editors, *PKC 2025: 28th International Conference on Theory and Practice of Public Key Cryptography, Part III*, volume 15676 of *Lecture Notes in Computer Science*, pages 3–36, Røros, Norway, May 12–15, 2025. Springer, Cham, Switzerland.

[FPTX25] Rex Fernando, Guru-Vamsi Policharla, Andrei Tonkikh, and Zhuolun Xiang. TrX: Encrypted mempools in high performance BFT protocols. Cryptology ePrint Archive, Report 2025/2032, 2025.

[GGH11] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Collision-free hashing from lattice problems. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation: In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, pages 30–39. Springer, 2011.

[GGL24] Rachit Garg, Rishab Goyal, and George Lu. Dynamic collusion functional encryption and multi-authority attribute-based encryption. In Qiang Tang and Vanessa Teague, editors, *PKC 2024: 27th International Conference on Theory and Practice of Public Key Cryptography, Part IV*, volume 14604 of *Lecture Notes in Computer Science*, pages 69–104, Sydney, NSW, Australia, April 15–17, 2024. Springer, Cham, Switzerland.

[GGLW22] Rachit Garg, Rishab Goyal, George Lu, and Brent Waters. Dynamic collusion bounded functional encryption from identity-based encryption. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 736–763, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

[GSW21] Rishab Goyal, Ridwan Syed, and Brent Waters. Bounded collusion ABE for TMs from IBE. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part IV*, volume 13093 of *Lecture Notes in Computer Science*, pages 371–402, Singapore, December 6–10, 2021. Springer, Cham, Switzerland.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 162–179, Santa Barbara, CA, USA, August 19–23, 2012. Springer Berlin Heidelberg, Germany.

[GWWW26] Junqing Gong, Brent Waters, Hoeteck Wee, and David J. Wu. Threshold batched identity-based encryption from pairings in the plain model. In Joan Daemen and Emmanuel Thomé, editors, *Advances in Cryptology – EUROCRYPT 2026, Part V*, volume 16545 of *Lecture Notes in Computer Science*, pages 548–578, Rome, Italy, May 10–14, 2026. Springer, Cham, Switzerland.

[KLJD23] Alireza Kavousi, Duc V. Le, Philipp Jovanovic, and George Danezis. BlindPerm: Efficient MEV mitigation with an encrypted mempool and permutation. Cryptology ePrint Archive, Report 2023/1061, 2023.

[Lee26] Kwangsu Lee. Verifiable outsourced bte with silent setup: Achieving constant-rate batched delegation. *Cryptology ePrint Archive*, 2026.

[LLNW16] Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 1–31, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany.

[LLW21] Qiqi Lai, Feng-Hao Liu, and Zhedong Wang. New lattice two-stage sampling technique and its applications to functional encryption - stronger security and smaller ciphertexts. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 498–527, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland.

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378, August 1988.

[Pol26] Guru-Vamsi Policharla. Labeled multi-key batched ibe. *Cryptology ePrint Archive*, 2026.

[QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th Annual Symposium on Foundations of Computer Science*, pages 859–870, Paris, France, October 7–9, 2018. IEEE Computer Society Press.

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

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53, Santa Barbara, CA, USA, August 19–23, 1984. Springer Berlin Heidelberg, Germany.

[SMM26] Ramprasad Sarkar, Shayeef Murshid, and Mriganka Mandal. Batched attribute-based encryption from bilinear pairings. *Cryptology ePrint Archive*, 2026.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010: 17th Conference on Computer and Communications Security*, pages 463–472, Chicago, Illinois, USA, October 4–8, 2010. ACM Press.

[Wee24] Hoeteck Wee. Circuit ABE with $\mathsf{poly}(\mathsf{depth}, \lambda)$-sized ciphertexts and keys from lattices. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part III*, volume 14922 of *Lecture Notes in Computer Science*, pages 178–209, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

[Wee25] Hoeteck Wee. Almost optimal KP and CP-ABE for circuits from succinct LWE. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025, Part III*, volume 15603 of *Lecture Notes in Computer Science*, pages 34–62, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland.

[WWW25] Brent Waters, Hoeteck Wee, and David J. Wu. New techniques for preimage sampling: Improved NIZKs and more from LWE. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025, Part IV*, volume 15604 of *Lecture Notes in Computer Science*, pages 3–33, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland.

[XZC26] Saisi Xiong, Yijian Zhang, and Jie Chen. Efficient batched IBE from lattices in the standard model. *IACR Communications in Cryptology*, 3(2), 2026.