

Batch Decryption from New Standard Assumptions

Nico Döttling¹, Bernardo Magri², Benjamin Marsh³, and Mahesh Sreekumar Rajasree^{id}*⁴

¹CISPA Helmholtz Center for Information Security , doettling@cispa.de

²University of Manchester , bernardo.magri@manchester.ac.uk

³Sei Labs and University of Portsmouth , ben@seinetwork.io

⁴University of Warsaw , srmahesh1994@gmail.com

September 28, 2026

Abstract

Batch decryption allows an authority to release a short key that enables public decryption of a selected batch of ciphertexts. Motivated by encrypted mempools, we study this capability through two constructions with different authorization semantics and assumptions.

First, we provide a simple and compact epochless construction in an RSA group based on Fiat’s Batch RSA [CRYPTO’89]. Its batch-decryption hint is just one group element, and its keys and ciphertexts have size independent of the batch size, with no batch bound fixed at setup. We prove adaptive security under the strong RSA assumption in the random-oracle model. The scheme permits repeated releases for overlapping batches, with each release authorizing decryption of the selected ciphertexts.

Second, we give an adaptively secure generic construction of identity-based encryption with batch decryption in the epoch-based model, which permits at most one batch-key query for the challenge epoch. The construction combines deferred encryption via garbled circuits, non-inclusion secure registration-based encryption, and weak receiver non-committing IBE. We construct the latter primitive from ordinary IBE and pseudorandom functions, obtaining an instantiation of the epoch-based construction from the computational Diffie–Hellman assumption.

batch decryption identity-based encryption encrypted mempools deferred encryption RSA

1 Introduction

Batch decryption, a notion recently formalized by Choudhuri *et al.* [CGPP24], allows ciphertexts generated independently by different senders to be decrypted using a single short public hint. The authority chooses the batch after encryption and releases a key whose size is independent of the number of ciphertexts. Anyone holding the batch and this key can then recover its messages. The central question is how to compress the information needed to authorize decryption of an adaptively selected collection while preserving the privacy of ciphertexts outside that collection.

*Funded by the European Union (ERC, LACONIC, 101041207). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Encrypted mempools are the application that motivates this notion: Transactions are encrypted before submission, their order is fixed while their contents remain hidden, and decryption follows after the ordering decision. This can mitigate forms of maximal extractable value (MEV) that exploit transaction contents during ordering. Batched threshold encryption addresses the communication cost of this process when a committee holds the decryption authority: instead of publishing a separate decryption share for every transaction, each party contributes to the release of a batch [CGPP24]. The cryptographic release mechanism is one component of such a system; the ordering protocol and the committee’s corruption model require their own guarantees.

Two choices distinguish the settings we consider. In ciphertext-specific batch decryption, a hint authorizes decryption of the selected ciphertexts. In batch IBE, a key authorizes a set of identities and enables decryption of ciphertexts under those identities [BLT25]. Independently, encryption may require an epoch label: batch keys then apply to ciphertexts for that epoch, whereas epochless encryption requires no such label. Our first construction provides ciphertext-specific releases without epochs; our second provides identity-based batch decryption with epochs.

In ciphertext-specific batch decryption, the adversary may request decryption hints for maliciously formed batches that exclude only the exact challenge ciphertext, including batches containing related ciphertexts. This is a natural analogue of chosen-ciphertext security for batch releases. In the identity-based setting, exclusion applies to the challenge identity; in our epoch-based model, this restriction concerns the challenge epoch.

Existing approaches include pairing-based batch threshold encryption [CGPP24, BNRT26, ADG⁺26] and lattice-based batch IBE [BLT25]. This raises a natural question: can compact batch decryption be constructed from other standard assumptions, such as strong RSA or CDH?

1.1 Our Contributions

In this work, we provide two novel approaches to construct batch decryption schemes:

- *Compact epochless batch decryption from strong RSA.* We provide a batch decryption scheme in the random oracle model from the strong RSA assumption (Section 3). In this scheme, the decryption hint is just one group element. The public key specifies an RSA modulus N and public sampling parameters, and the secret key additionally contains the group order $\varphi(N)$. Key and ciphertext sizes depend only on the security parameter. Unlike the pairing-based constructions of [CGPP24, BNRT26, ADG⁺26, Pol26], our scheme requires no public setup or decryption material growing with a predetermined maximum batch size, although some of these constructions already achieve constant-size encryption keys. No maximum batch size or lifetime is fixed at setup. To modularize our construction, we introduce a tag-based notion of batch decryption and formalize a simple security notion we call adaptive tag-exclusion privacy. In such schemes, every ciphertext is endowed with a tag, and batch-decryption queries are not allowed to include ciphertexts tagged with the challenge tag. This is a natural generalization of the notion of tag-based encryption [Kil06]. A CHK/Kiltz compiler [CHK04, Kil06] using strong one-time signatures then gives ordinary batch decryption against malicious batch-decryption queries that exclude only the exact challenge ciphertext.
- *Generic batch decryption with epochs.* As a second contribution, we construct an adaptively secure identity-based encryption scheme with batch decryption (Section 4). The critical component in this construction is a notion we call weak receiver non-committing identity-based encryption, which we can construct from the CDH assumption (Section 4.2). The

construction further relies on an adaptive garbling scheme [BHR12a, HJO⁺16] and a weak form of registration-based encryption [GHMR18, GHM⁺19]. In essence, the principle of deferred encryption [DG17b, BLSV18] lets a sender prepare an encryption before the hash of the eventual identity set is known, and allows to later *install* the desired hash via a short decryption hint. We prove security of our construction under the standard notion that permits at most one partial-decryption query for the challenge epoch, whose identity set must exclude the challenge identity. This restriction ensures that a challenge garbled circuit is opened on at most one input. Consequently, instantiating our weak receiver non-committing IBE scheme with CDH-based IBE [DG17b] and combining it with CDH-based RBE [GHM⁺19] and adaptive garbling from one-way functions [HJO⁺16] gives the epoch-based feasibility result from CDH in Section 4.4.

1.2 Related Work

Batch and threshold encryption. Choudhuri *et al.* [CGPP24] study batched threshold encryption for mempool privacy, reducing the communication needed to decrypt a batch relative to applying threshold encryption separately to every ciphertext. Boneh, Laufer, and Tas [BLT25] study epochless batch IBE, observe that key-policy ABE with short policy keys gives batch IBE, and develop a lattice-based threshold construction. The security and efficiency of the ABE route depend on the particular ABE instantiation. For example, the compact-key ABE of Boneh *et al.* [BGG⁺14] has a selective proof and obtains adaptive security by complexity leveraging under subexponential LWE. Our generic result instead uses deferred encryption and establishes a CDH-based construction under the epoch restriction stated above.

For ciphertext-specific releases, the partial-fraction construction of Boneh *et al.* [BNRT26] and BTX [ADG⁺26] give epochless batch threshold encryption using pairings. TrX [FPTX25] studies the integration of batched threshold encryption into a high-performance BFT protocol. These works address threshold operation and concrete performance, whereas our RSA construction studies compact single-authority release under a different assumption. Its one-element hint does not by itself imply that a distributed protocol has communication independent of the batch size.

Root batching. The arithmetic behind our epochless construction goes back to Fiat’s Batch RSA [Fia97]: roots for pairwise coprime exponents can be recovered from their product and the public group elements. We use this mechanism to compress a selected batch’s decryption information. The privacy proof additionally needs to handle malicious ciphertexts and retagging. Our root-dependent check addresses these issues in the random-oracle model. For ordinary exact-ciphertext privacy we apply the signature transformation of Canetti, Halevi, and Katz [CHK04], in its tag-based formulation due to Kiltz [Kil06]; Appendix B proves the required whole-batch version. The strong RSA assumption allows the solver to choose the extracted exponent; it is not fixed-exponent search RSA.

Non-Committing Encryption. The study of non-committing encryption (NCE) includes a substantial line of work on constructing schemes that remain secure even when the adversary learns the randomness used in both the setup and encryption procedures. These works achieve NCE from a range of assumptions, with a particular emphasis on obtaining high ciphertext rate [CFGN96, Bea97, DN00, CDSMW09, HOR15, HORR15, CPR17, YKT19, YKXT20, BBD⁺20].

A separate line of work investigates weaker variants of NCE, aiming to improve parameters when the adversary is allowed to learn the randomness used in only one of the two phases: either setup, corresponding to receiver randomness, or encryption, corresponding to sender randomness. Jarecki and Lysyanskaya [JL00] gave a construction that is non-committing only with respect to the receiver. Canetti, Halevi, and Katz [CHK05] constructed a constant-rate NCE scheme with erasures, in which the adversary obtains only the secret key and the ciphertext.

Hiroka *et al.* [HMNY21] proposed non-committing attribute-based encryption (NC-ABE) from indistinguishability obfuscation. Their work was motivated by attribute-based encryption with certified deletion in the quantum setting, and considers an adversary that receives the master secret key together with the ciphertext. More recently, Goyal *et al.* [GKK⁺25] constructed non-committing identity-based encryption from bilinear pairings and used it to obtain strong incompressible identity-based encryption [GWZ22, BDD22, GWZ23, GKR25].

1.3 Technical Overview of the Epochless Construction

We begin with an approach to ordinary batch decryption without epochs, based on the RSA cryptosystem. Our goal is to keep the ciphertexts, keys, and batch-decryption hints compact. The roadmap is simple: we start with independently generated RSA ciphertexts and show how their decryption information can be compressed after the batch has been selected.

Batch Decryption by Releasing Roots. First consider the basic textbook RSA encryption mechanism. Let $N = pq$ be an RSA modulus and write $G = \mathbb{Z}_N^\times$. For notational convenience, products, quotients, and powers of group elements are evaluated modulo N . For each admissible prime exponent $N/2 < e < N$, we have $\gcd(e, \varphi(N)) = 1$ because $p, q \leq N/3 < e$, so the map $x \mapsto x^e$ is a permutation of G . The authority retains $\varphi(N)$ as a trapdoor that allows it to invert these maps, whereas extracting roots without the trapdoor is hard under the strong RSA assumption. A sender samples $x \in G$ and an admissible exponent e , and publishes

$$C = x^e.$$

Hashing x together with the public ciphertext data gives a symmetric key with which to encrypt the message. The authority can recover x from C using its trapdoor and thereby decrypt.

A trivial and non-compact batch decryption mechanism would, for a batch of ciphertexts with group components $C_1, \dots, C_\ell$, release their roots $x_1, \dots, x_\ell$. This allows anyone to decrypt the corresponding messages. However, such a hint contains ℓ group elements, and is therefore highly non-compact. We would like to compress these roots into a single group element without losing the ability to recover each of them.

One Product, Many Roots. A natural candidate for the compressed hint is simply the product $W = \prod_{i=1}^\ell x_i$. At first sight, multiplying the roots seems to discard the information needed to recover them individually. However, note that the ciphertexts already provide powers of these same roots, i.e. there is already useful information about the individual roots present. The key is to choose the exponents e_1 and e_2 so that this public information complements the product.

Consider just two ciphertexts, $C_1 = x_1^{e_1}$ and $C_2 = x_2^{e_2}$, with coprime exponents. By Bézout's identity, we can efficiently find integers a, b such that $ae_1 + be_2 = 1$. Given $W = x_1x_2$, we can

compute

$$\frac{W^{e_2}}{C_2} = x_1^{e_2}, \quad C_1^a \left(\frac{W^{e_2}}{C_2} \right)^b = x_1^{ae_1} \cdot \left(\frac{x_1^{e_2} x_2^{e_2}}{x_2^{e_2}} \right)^b = x_1^{ae_1 + be_2} = x_1.$$

The other root is then $x_2 = W/x_1$. All exponents here are integers; negative exponents only require group inversion. In particular, public recovery does not require knowing the group order or extracting a root without the hint.

This recovery identity is the algebraic ingredient behind Fiat’s Batch RSA [Fia97], where batching amortizes the cost of RSA private-key operations. In our setting, the product is published as a compact decryption hint for a batch chosen after encryption, and security must hold even when an adversary obtains hints for adaptively chosen batches of malicious ciphertexts. The algebraic identity alone does not provide this guarantee: such queries can expose a challenge root, as we show below. We address this problem through a root-dependent consistency check, a signature transformation, and a security analysis that accounts for adaptive releases.

For an arbitrary batch, the Chinese remainder theorem gives the analogous recovery identities whenever the exponents are pairwise coprime. A convenient way to arrange this is for every sender to sample a prime independently from a sufficiently large admissible family. Honest exponent collisions are then negligible over any polynomial number of encryptions; batches with repeated exponents are rejected. Neither setup nor encryption needs to know the eventual batch size or an upper bound on the number of future ciphertexts. The prime lengths depend only on the security parameter, and the successful hint is still just W .

Why Root Batching Alone Is Not Enough. So far, this argument explains how to decrypt a batch of honest ciphertexts. Security, however, must also account for adversarially formed release queries. Suppose that a challenge ciphertext contains (e^*, C^*) . We first construct a tag-based batch-decryption scheme, in which each ciphertext carries an externally supplied tag t . In the security experiment, every admitted batch-decryption query must exclude ciphertexts carrying the challenge tag t^* . If the adversary copies the challenge pair into a ciphertext under another tag, releasing its root in a singleton batch still reveals x^* and hence the challenge message key.

Including the tag in the key-derivation hash does not prevent this attack: once x^* is known, the adversary can evaluate that hash for the original tag as well. Nor does a signature alone suffice, since the adversary can sign the copied group component under a fresh verification key. The missing property is a binding between the root and the context under which its release is requested.

We obtain this binding through a root-dependent random-oracle check. For an externally chosen tag t , the sender publishes

$$\rho = H_{\text{check}}(\text{pk}, t, e, C, x), \quad X = (t, e, C, \rho),$$

and derives a message key $K = H_{\text{key}}(\text{pk}, X, x)$. It encrypts the message as $c = m \oplus H_{\text{pad}}(\text{pk}, t, X, K)$. Before releasing a product of roots, the authority extracts each root and checks the corresponding value of ρ . If any check fails, it rejects the entire batch without releasing a hint. Retagging the challenge now requires a new check value at an oracle input containing the unknown challenge root. Copying the old value of ρ does not reveal the new value, since the tag is part of the oracle input.

This is a *private consistency check*, not a publicly verifiable proof of knowledge or an assumption that senders behave honestly. The authority can perform the check because it has the root-extraction trapdoor. Its successful response remains the same single group element W . The remaining security

argument must handle arbitrary submitted ciphertexts, rather than assuming that each sender knows its root. We do this directly in the batch-decryption security experiment.

Proving Security with Adaptive Releases. The root check also provides a way to simulate release queries without knowing the trapdoor. A simulator monitors random-oracle queries and can recognize a correct root by testing $x^e = C$. Once all roots needed by an accepted batch have appeared in the corresponding check queries, it can compute their product itself. Conversely, a check whose root has not been queried can succeed only through a correct guess of the hidden oracle value.

The probability that a private check in a release query succeeds before a corresponding check-oracle query reveals its root is negligible, even across adaptively chosen release queries. We can therefore simulate releases using the recorded roots, rejecting a batch if any required root is missing. Apart from negligible guessing events, distinguishing the challenge messages requires an oracle query revealing the challenge root, which yields a solution to the strong RSA problem (Section 3.4).

The tagged scheme accepts arbitrary polynomial-length tags, including repeated tags in the same batch. They carry no application-identity or recipient-key semantics. Only repeated exponents are rejected. Its security experiment excludes every admitted query at the challenge tag and records all admitted entries, even those whose private checks failed or were not reached. The challenge tag itself may be chosen adaptively after earlier releases.

From Tagged to Ordinary Batch Decryption. Exact-ciphertext exclusion permits a modified payload under the same tag. In the tagged scheme, this would release the challenge root and reveal the original message. The CHK signature transformation [CHK04], in Kiltz’s tag-based formulation [Kil06], handles this remaining case. A sender generates a fresh one-time signature key pair, uses $t = vk$ as the external tag, and signs the complete context of the resulting tagged ciphertext D . The ordinary ciphertext is (vk, D, sig) . No nonce is needed.

An admitted different ciphertext with the challenge verification key gives a strong one-time forgery, including a changed signature on the same message. Changing the key changes the tag and is covered by tagged security. The reduction can reserve its challenge signature key before setup and keep it hidden: an admitted prechallenge entry under that key is already a forgery before any signing query. This also accounts for coincident verification keys without assuming they are uniform or unique. Appendix B gives the compiler and its batch-specific proof. It validates the complete batch and forwards the whole tagged projection, preserving the one original hint and the order of its entries.

The resulting construction has batch-independent public and secret keys, ciphertexts, and hints. In particular, a successful hint contains one group element, and all ciphertext overhead depends only on the security parameter. Processing time still depends on the batch size. The product W authorizes recovery for the actual selected ciphertexts. This lets us avoid both epochs and a batch hash fixed at encryption time, and different releases may concern overlapping batches. The construction uses RSA arithmetic, with security based on the strong RSA assumption and the random-oracle and signature assumptions specified in Section 3. It uses neither pairings nor garbling.

Toward Threshold Release (Informal). Secret sharing suggests a way to distribute the authority, but further protocol design is needed. One could share the root-extraction trapdoor

among a committee and use an appropriate secure distributed computation to evaluate **PartDec** on an agreed public batch. That computation would have to perform the compiled public signature checks, extract the roots, evaluate their root-dependent hashes without exposing the hash inputs, perform the private consistency checks, and release only the prescribed response: rejection if a check fails, or the product W if all checks succeed. In particular, publishing individual roots or intermediate values before validation finishes can disclose decryption information for a batch that should be rejected.

Reconstructing the trapdoor at a designated combiner would let that party run the existing algorithm, but would also give it the ability to decrypt arbitrary ciphertexts on its own. Thus this shortcut distributes custody before reconstruction; it does not retain threshold protection afterward. Computing on shares could avoid that reconstruction under a suitable secure-computation model, but the committee’s communication may still grow with the batch size even though the final public hint is one group element. The present random-oracle proof does not provide such a distributed hash-evaluation protocol. Distributed setup, corruption thresholds, robustness against faulty parties, and a security proof for the distributed protocol remain outside the scope of this paper.

1.4 Technical Overview of the Construction with Epochs

We now turn to our identity-based construction with epochs. At a high level, an identity-based encryption scheme with batch decryption augments ordinary identity-based encryption with two additional algorithms **PartDec** and **Dec**. The setup algorithm **Setup** generates a public key and a secret key. Encryption proceeds exactly as in ordinary IBE: given a public key $\mathbf{pk}$, an epoch τ , an identity id , and a message m , the encryption algorithm **Enc** outputs a ciphertext CT . The partial decryption algorithm **PartDec** takes as input the secret key $\mathbf{sk}$, an epoch τ , and a batch of ciphertexts $\text{CT}_1, \dots, \text{CT}_\ell$ for that epoch, and computes a succinct batch key sbk . Finally, the decryption algorithm **Dec** takes as input sbk , an index $i^* \in [\ell]$, and the batch $\text{CT}_1, \dots, \text{CT}_\ell$, and uses the succinct batch key to decrypt the i^* -th ciphertext in the batch.

Correctness requires that for every honestly generated batch, the succinct batch key computed by **PartDec** allows one to recover the message contained in any ciphertext of the batch. Security is defined in the natural adaptive chosen-plaintext sense, in the presence of an oracle for **PartDec**: the adversary may adaptively query the partial decryption oracle and then request a challenge ciphertext for an epoch τ^* and identity id^* of its choice, subject to the restriction that it obtains at most one succinct batch key for the challenge epoch and that the corresponding queried batch, if it exists, excludes the challenge identity and challenge ciphertext.

We will now discuss the conceptual construction template underlying batch decryption with epochs.

Batch Decryption with Epochs via Witness Encryption. Before we provide the details underlying our construction, we first discuss the basic blueprint underlying several prior constructions of IBE with batch decryption [CGPP24, AFP25, GWWW26]. Such schemes can be understood as a type of witness encryption in a natural way. More specifically, they fit the signature-based witness encryption paradigm of [DHMW23]. Consider the following witness relation $\mathcal{R}$. A statement is a tuple $x = (\mathbf{vk}, \tau, \text{id})$ consisting of a signature verification key $\mathbf{vk}$, an epoch τ , and an identity id , whereas a witness is a tuple $w = (\mathbf{h}, \pi, \sigma)$, where $\mathbf{h}$ is a commitment to a set of identities, π is a *local opening* of $\mathbf{h}$, and σ is a signature. The relation $\mathcal{R}$ ensures that

- The local opening π opens the hash value h to the identity id , and
- σ is a valid signature of (τ, h) under the verification key vk .

Given a witness encryption scheme for the relation $\mathcal{R}$, we can obtain an IBE with batch decryption in the natural way:

- The setup algorithm generates a pair of verification and signing keys (vk, sk) for the underlying signature scheme.
- A ciphertext ct for epoch τ and identity id is a WE ciphertext for the statement $x = (vk, \tau, id)$.
- The **PartDec** algorithm proceeds as follows. Given $ct_1, \dots, ct_\ell$ with corresponding identities $id_1, \dots, id_\ell$, it hashes the identities into a succinct hash value h . The succinct batch key sbk is a signature σ of (τ, h) .
- The decryption algorithm proceeds as follows. Given epoch τ , a succinct batch key $sbk = \sigma$, and ciphertexts $ct_1, \dots, ct_\ell$, it recomputes the hash h of the identities $id_1, \dots, id_\ell$ and also computes local openings π_i for each identity id_i . To decrypt the ciphertext ct_i we can now use the witness $w = (h, \pi_i, \sigma)$.

Correctness of this scheme follows immediately from the correctness of the underlying primitives. To argue security, we would ideally like to rely on the unforgeability of the signature scheme, the binding property of the hash, and the security of the witness encryption scheme. However, the standard security notion for witness encryption only provides security for ciphertexts that were generated with respect to false statements [GGSW13]. Consequently, most constructions either rely on stronger building blocks and idealized models, as in the generic group model analysis of [CGPP24], or prove security directly from concrete pairing assumptions [GWWW26].

In the algebraic context of bilinear pairings, the witness relation can be *fully linearized* as all components have linear instantiations:

- The hash is realized via the KZG commitment scheme [KZG10] or LY commitments [LY10]. The opening relation is a linear pairing product equation.
- The signature is realized via the BLS scheme [BLS04] or the BB scheme [BB08]. The verification equation is also a linear pairing product equation.

Consequently, the relation $\mathcal{R}$ induces a linearly verifiable proof system [GHK⁺25], and such proof systems can be compiled into a special-purpose witness encryption mechanism [GHK⁺25]. Schemes obtained from this blueprint in the bilinear setting generally have compact ciphertexts consisting of only a constant number of group elements, but suffer from large common reference strings, as is common for pairing-based constructions that involve hashing or vector commitments.

Batch Decryption without linear Relations. The constructions derived from the blueprint discussed above heavily rely on the algebraic properties of linear pairing product equations. Specifically, linearity allows one to break the dependency of the ciphertext on the hash value of the identities by making the hash part of the witness. Hence, there is little hope of instantiating this approach from primitives that provide less algebraic structure.

To gain some understanding, let us first consider a counterfactual scenario in which the hash of the identities is already known at encryption time. Specifically, suppose that the encryption algorithm is given the hash of the identities that will later be decrypted. This is precisely the setting of registration-based encryption (RBE) [GHMR18, GHM⁺19]. The variant relevant for us is a simplified, keyless form of RBE, closely related to what recent work calls *laconic encryption* [DKL⁺23].

Standard RBE allows us to hash a set of identities, together with the corresponding registered public keys, into a succinct public value h and, for every identity contained in the set, derive a succinct decryption hint η_{id} . To encrypt a message to an identity id , the encrypter only needs the public parameters, the public hash h , and the identity id of the intended receiver. Importantly, the size of the resulting ciphertext is independent of the number of identities that were hashed into h . Given a ciphertext ct for identity id , the legitimate receiver can use the decryption hint η_{id} to decrypt.

For our application, we only need a keyless variant in which the hash depends solely on the identities. Concretely, one can obtain such a scheme from a standard RBE scheme by generating a single registration key pair during setup, placing the public key into the public parameters, and using this same public key for every identity. In this way, the public-key component no longer needs to be written explicitly in the hash input. In addition, we only require the simplified security notion of non-inclusion security: we only require privacy for identities that have not been hashed into h .

If the hash h were fixed before encryption, then RBE would already give the desired functionality: it binds a ciphertext ct to a hash h such that only legitimate users whose identities have been hashed into h can decrypt. Clearly, however, the order of operations is wrong for batch decryption. In batch decryption the ciphertexts are generated before the hash h is known.

Deferred Encryption. We solve this problem by not including the actual RBE ciphertext in the batch decryption ciphertext. Instead, the ciphertext contains a gadget that allows us to generate the appropriate RBE ciphertext once the batch is fixed. More specifically, the ciphertext contains a garbled circuit $C[pp, id, m, r](h)$ which, on input the hash h of the identities in the batch, outputs the RBE ciphertext $RBE.Enc(pp, h, id, m; r)$.

This is the place where we rely on the *deferred encryption* paradigm. The garbled circuit restores the correct order of operations: encryption can take place before the batch hash is known, and the actual RBE ciphertext is only materialized later, once the hash h has been fixed [DG17c, BLSV18]. This technique first appeared in the works of Cho *et al.* [CDG⁺17] for the construction of laconic oblivious transfer and Döttling and Garg [DG17c] for the construction of IBE; the name *deferred encryption* for this technique was coined by Brakerski *et al.* [BLSV18]. It was subsequently used and refined for adaptively secure IBE [DG17a], anonymous IBE [BLSV18], key-dependent message security [DGHM18, WW23], KDM-secure IBE [KT18], and registration-based encryption [GHMR18, GHM⁺19, GV20].

For our construction, the remaining problem is to provide the input h to the garbled circuit in a succinct and authorized way.

This is precisely where we use weak RNC-IBE. For every wire j of the garbled circuit and every bit $b \in \{0, 1\}$, we encrypt the corresponding garbled label under its own weak RNC-IBE instance. Once the batch of ciphertexts is fixed, the algorithm **PartDec** computes the hash h of the identities in the batch and outputs keys that can be viewed as the weak RNC-IBE keys corresponding to the pair (τ, h) . Hence the succinct batch key can be interpreted as a joint signature-based witness for the pair (τ, h) in the sense that it authorizes exactly the garbled-circuit labels corresponding to h .

and no others. In this way, our construction replaces the linear-SNARK machinery by a combination of deferred encryption, RBE, and weak RNC-IBE.

Proving Adaptive Security. Proving security of this scheme is non-trivial. The core difficulty is that at the time the challenge ciphertext is generated, we cannot anticipate which epoch will become the challenge epoch and, more critically, which set of identities the adversary will choose to hash for this epoch. Consequently, we do not know at encryption time which garbled-circuit labels will later need to be revealed.

To resolve this issue, we rely on a form of IBE that can deal with adaptive corruptions, namely weak receiver non-committing identity-based encryption (weak RNC-IBE). Intuitively, this allows us to prepare the weak RNC-IBE ciphertexts in the challenge ciphertext before the relevant hash value is known and only later explain them consistently once the adversary has fixed the batch for the challenge epoch. This is the primary reason why weak RNC-IBE enters the construction. This idea is inspired by recent work on non-committing IBE [GKK⁺25]; however, the notion of [GKK⁺25] equivocates the master secret key, whereas for our application it is more natural to equivocate only the user secret keys associated with the challenge epoch. We provide a conceptually simple construction of weak RNC-IBE in Appendix D.2.

A final aspect we have not yet considered is that the adversary sees the garbled circuit in the challenge ciphertext potentially *before* the relevant input is fixed, that is before querying a succinct decryption key for the challenge epoch. Hence, we will also rely on an adaptively secure garbling scheme [BHR12a, HJO⁺16].

1.5 Organization

The remainder of this paper is organized as follows. Section 2 establishes the notation and defines ordinary and tagged epochless batch decryption. Section 3 presents our compact tagged construction from strong RSA, its correctness and security arguments, and an ordinary batch-decryption corollary. Section 4 first defines the epoch-based model, then introduces the required registration-based encryption interface and weak receiver non-committing IBE, before presenting the generic batch construction, its security argument, and the CDH instantiation. Appendix A collects the background definitions, and Appendix B gives the CHK/Kiltz compiler and its proof for batch releases. The detailed proofs and RSA sampling procedures are deferred to the remaining appendices.

2 Preliminaries

In this section, we establish the notation and define ordinary and tagged epochless batch decryption. The epoch-based interface and its security restriction are defined separately in Section 4.

Notation. Let $\lambda \in \mathbb{N}$ be the security parameter, provided to all algorithms as 1^λ . For $n \geq 1$, we define $[n] := \{1, \dots, n\}$. We write $x \xleftarrow{\$} S$ to denote sampling x uniformly at random from a finite set S , and $y \leftarrow \mathcal{A}(x; r)$ to denote the execution of a randomized algorithm $\mathcal{A}$ on input x with random coins r . String concatenation is denoted by $x \parallel y$. For any binary string x , we use x_i to denote the i^{th} bit of x . We use $\text{negl}(\lambda)$ to denote an unspecified negligible function. Two distribution ensembles X and Y are computationally indistinguishable, denoted $X \approx_c Y$, if the advantage of any PPT distinguisher is bounded by $\text{negl}(\lambda)$.

The formal definitions of pseudorandom functions, garbling schemes, ordinary IBE, and registration-based encryption are given in Appendix A. We recall the required registration-based encryption interface where it is used in Section 4.1.

2.1 Epochless Batch Decryption

An epochless batch-decryption scheme $\text{BD} = (\text{Setup}, \text{Enc}, \text{PartDec}, \text{Dec})$ for message spaces $\mathcal{M}_\lambda$ consists of the following four polynomial-time algorithms.

$\text{Setup}(1^\lambda)$: *Run by the authority.* The setup algorithm outputs a public key pk and a secret key sk . Neither key changes between releases, and setup takes no maximum batch size or bound on the number of future ciphertexts.

$\text{Enc}(\text{pk}, m)$: *Run by a sender.* The encryption algorithm takes a message $m \in \mathcal{M}_\lambda$ and outputs a ciphertext CT . Encryption does not take an epoch or the eventual batch as input. Malformed inputs are rejected with $\perp$.

$\text{PartDec}(\text{sk}, \mathcal{B})$: *Run by the authority.* On input a batch $\mathcal{B} = (\text{CT}_1, \dots, \text{CT}_B)$, the partial-decryption algorithm outputs a succinct batch key T , the rejection symbol $\perp$, or a distinguished empty-batch response **empty**. It first checks public admissibility of the complete batch. Inadmissible batches are rejected; an admissible empty batch returns **empty**. Any subsequent private validation either accepts the whole batch or rejects it without releasing a batch key.

$\text{Dec}(T, i, \mathcal{B})$: *Run by any party with the batch key T .* The public decryption algorithm takes a batch key, an index $i \in [B]$, and the batch, and outputs a message or $\perp$. It returns $\perp$ for malformed inputs, an empty batch, or an out-of-range index. Any public parameters needed by this interface must be included in its inputs and counted in their sizes.

We write $\text{Adm}(\text{pk}, \mathcal{B})$ for the scheme's efficient public admissibility predicate. It checks well-formedness and consistency with pk and may impose scheme-specific public checks. The empty batch is admissible for a well-formed public key. Parsing and admissibility are conventions for the four algorithms, not additional scheme algorithms. Public admissibility does not assert knowledge of encryption randomness or imply that private validation will succeed.

Correctness and compactness. Fix any polynomial-time procedure that adaptively requests honest encryptions of messages of its choice and selects batches from this history. Each selected batch uses distinct encryption-invocation indices: the same invocation is not selected twice within one batch. Distinct invocations may nevertheless output identical ciphertext values; they remain distinct occurrences, and any resulting rejection is a correctness failure. Correctness requires that, except with negligible probability, every selected nonempty batch $\mathcal{B}$ receives a successful batch key $T \leftarrow \text{PartDec}(\text{sk}, \mathcal{B})$ and

$$\text{Dec}(T, i, \mathcal{B}) = m_i \quad \text{for every } i \in [B],$$

where m_i is the message used to generate CT_i . The probability includes setup, honest encryption, release and decryption randomness, and any random oracles in the scheme's model. Batches may overlap, and old batch keys may be reused with their original batches. This is an unconditional requirement: an honest batch's public rejection counts as a correctness failure. It imposes no availability requirement on batches containing malicious ciphertexts.

Succinctness requires a fixed polynomial in λ bounding the complete batch-key length, independently of B and of the number of earlier releases. We call the scheme *compact* if the public key, secret key, and ciphertext lengths also have such fixed bounds (with message length counted if it varies). The public batch is an input to decryption; processing time may grow with its size.

Ordinary security. For $b \in \{0, 1\}$, define the exact-ciphertext-exclusion experiment $\text{CT}_b^{\mathcal{A}}$ as follows. The adversary has access throughout to any public random oracles specified by the scheme's model.

- *Initialization Phase:* The challenger computes $(\text{pk}, \text{sk}) \leftarrow \text{Setup}(1^\lambda)$, sends pk to $\mathcal{A}$, and initializes an empty admission record. The admission record is maintained by the security experiment and is not part of the authority's state.
- *Pre-Challenge Query Phase:* For each adaptive batch query $\mathcal{B}$, the challenger performs the complete public admissibility check. It returns $\perp$ for an inadmissible batch and **empty** for an admissible empty batch. For an admitted nonempty batch, it records the complete batch before any private validation, then returns $\text{PartDec}(\text{sk}, \mathcal{B})$. The record includes entries whose private check fails or is never reached because an earlier check fails.
- *Challenge Phase:* At an adaptively chosen time, $\mathcal{A}$ submits a single challenge (m_0, m_1) with equal-length messages $m_0, m_1 \in \mathcal{M}_\lambda$. The challenger first generates $\text{CT}^* \leftarrow \text{Enc}(\text{pk}, m_b)$ with fresh randomness, then tests whether that exact ciphertext occurs in the earlier admission record. If so, the experiment outputs zero and terminates; otherwise, it returns CT^* . The challenge is never resampled to avoid this test.
- *Post-Challenge Query Phase:* Queries undergo the same public checks and recording as before the challenge. After recording an admitted nonempty batch, the experiment outputs zero and terminates if it contains the exact challenge ciphertext CT^* . Otherwise, the challenger returns $\text{PartDec}(\text{sk}, \mathcal{B})$. The exclusion test precedes private validation.
- *Response Phase:* The experiment outputs the adversary's final bit. A malformed challenge or an execution without a challenge has output zero.

Definition 2.1 *Define*

$$\Delta_{\text{CT}}(\mathcal{A}) = |\Pr[\text{CT}_0^{\mathcal{A}} = 1] - \Pr[\text{CT}_1^{\mathcal{A}} = 1]|.$$

A scheme satisfies exact-ciphertext-exclusion privacy if $\Delta_{\text{CT}}(\mathcal{A})$ is negligible in λ for every uniform PPT adversary $\mathcal{A}$.

Only the exact ciphertext is forbidden. A changed payload, signature, or other component is not excluded merely because it is related to the challenge.

2.2 Tagged Batch Decryption

A tagged batch-decryption scheme $\text{TBD} = (\text{Setup}, \text{Enc}, \text{PartDec}, \text{Dec})$ has the same interface, except that encryption takes an external tag, $D \leftarrow \text{Enc}(\text{pk}, t, m)$, and a public batch is a tuple $\mathcal{B} = ((t_1, D_1), \dots, (t_B, D_B))$. A tag is an arbitrary polynomial-length binary string. It has no required internal structure and does not specify a recipient key. Tags may repeat, including within a batch;

there is no general distinct-tag admission rule. The public admission convention, the responses on empty input and rejection, and the ordering of public and private checks are as defined above. If a ciphertext explicitly contains a tag, the public admission check additionally verifies that this embedded tag matches the externally supplied tag.

Correctness uses the same adaptive history and distinct invocation indices as above, with tags and messages chosen before each honest encryption. In particular, it covers repeated tags and coincident outputs of distinct invocations. Tag lengths are included in the length of the public input and in ciphertext size whenever stored there. Compactness requires fixed polynomial bounds in λ and the length of the individual tag and message, independent of batch size and lifetime; the hint bound depends only on λ .

Tag-exclusion security. The adaptive-tag experiment $\text{Tag}_b^{\mathcal{A}}$ has the phases of $\text{CT}_b^{\mathcal{A}}$, with tagged batches and the following changes. Every admitted entry and its tag is recorded before private validation, including failed or unreached entries. At the challenge, $\mathcal{A}$ chooses (t^*, m_0, m_1) after its prechallenge queries. If t^* occurs in the earlier admission record, the experiment terminates with output zero *before* challenge encryption. Otherwise it returns $D^* \leftarrow \text{Enc}(\text{pk}, t^*, m_b)$. After the challenge, an admitted batch containing any pair with tag t^* causes output zero before private validation. This excludes the whole challenge tag, not just (t^*, D^*) . Admission of the complete batch still precedes exclusion.

The selective-tag variant $\text{sTag}_b^{\mathcal{A}}$ additionally requires $\mathcal{A}$ to commit to t^* before setup. Messages and release queries remain adaptive. The tag is forbidden in all phases; because it is known in advance, an admitted prechallenge batch containing it terminates immediately with output zero. All other conventions agree with the adaptive-tag experiment.

Definition 2.2 For $X \in \{\text{Tag}, \text{sTag}\}$, let

$$\Delta_X(\mathcal{A}) = |\Pr[X_0^{\mathcal{A}} = 1] - \Pr[X_1^{\mathcal{A}} = 1]|.$$

Adaptive-tag (respectively, selective-tag) privacy requires negligible $\Delta_{\text{Tag}}(\mathcal{A})$ (respectively, $\Delta_{\text{sTag}}(\mathcal{A})$) for every uniform PPT adversary.

This is a batch-release version of tag-exclusion security for tag-based encryption [Kil06, Section 3.2]. Ordinary single-ciphertext tag-based security does not by itself protect a batch hint: the oracle here returns the actual whole-batch response. Appendix B adapts the CHK signature transformation to this interface.

3 A Compact Epochless Construction from Strong RSA

We describe a construction for compact epochless batch decryption in the RSA group $G = \mathbb{Z}_N^\times$ under the strong RSA assumption. Its ciphertexts as well as its batch-decryption hints have size independent of the batch size. The public and secret keys also have batch-independent size. Setup does not take a maximum batch size or a bound on the number of future encryptions. Senders encrypt independently, before the eventual batch is known, and batches may overlap.

The underlying algebra is the root-batching mechanism of Fiat [Fia97]. Each sender chooses a prime exponent e_i and a group element x_i , and publishes $C_i = x_i^{e_i}$. The authority can extract the roots and release their product $W = \prod_i x_i$. If the e_i are pairwise coprime, integer Chinese

remainder identities allow public recovery of each individual root from W and the ciphertexts. For adversarially chosen ciphertexts, we additionally need a random-oracle check binding the root to its complete public context. We first prove security of a tagged core, whose release experiment excludes the challenge tag. The standard CHK/Kiltz signature transformation then gives ordinary batch decryption with exact-ciphertext exclusion (Appendix B).

The authority retains $\varphi(N)$ to extract roots, while security against parties without this trapdoor relies on the strong RSA assumption stated below. We give the construction, correctness and size bounds, and a security argument in the random-oracle model. This does not establish a construction from CDH or DDH in publicly known prime-order groups, nor does it give threshold partial decryption.

3.1 RSA Groups and the Strong RSA Assumption

Throughout this section, $G = \mathbb{Z}_N^\times$, where $N = pq$ for distinct odd primes p, q , and 1_G denotes its identity. We use multiplicative group notation, with products, inverses, and powers evaluated modulo N . Exponents are integers, and negative exponents denote inversion followed by exponentiation. No generator or discrete logarithms of the sampled elements are required.

Let $\text{Gen}_{\text{rt}}(1^\lambda)$ denote RSA setup, which outputs a public descriptor pp containing N and the exponent-sampling parameters, together with the secret trapdoor $\text{td} = \varphi(N)$. We specify the samplers in Appendix C.1. The following notation records the RSA operations and their size and sampling bounds.

Public group operations. Membership, equality, multiplication, inversion, and integer exponentiation in G are efficient given N . We use the canonical integer representatives of units modulo N , of at most $\ell_G(\lambda)$ bits. The descriptor and trapdoor have lengths at most $\ell_{\text{pp}}(\lambda)$ and $\ell_{\text{td}}(\lambda)$, respectively.

Admissible exponents. The publicly decidable set $\mathcal{P}_{\text{pp}}$ consists of the primes $N/2 < e < N$, each of at most $\ell_e(\lambda)$ bits, including the designated fallback exponent. Since $p, q \leq N/3 < e$, we have $\gcd(e, \varphi(N)) = 1$, and the map $x \mapsto x^e$ is a permutation of G for every $e \in \mathcal{P}_{\text{pp}}$.

Trapdoor inversion. For every $e \in \mathcal{P}_{\text{pp}}$ and $C \in G$, the deterministic algorithm

$$x \leftarrow \text{Root}(\text{pp}, \text{td}, e, C)$$

returns the unique $x \in G$ satisfying $x^e = C$. With $M = \varphi(N)$ supplied by the trapdoor, it computes $x = C^{e^{-1} \bmod M}$ in G .

Public sampling. The algorithms $\text{Sample}_G(\text{pp})$ and $\text{Sample}_e(\text{pp})$ always return an element of G and an exponent in $\mathcal{P}_{\text{pp}}$, respectively. Conditional on each supported pp , the element distribution has statistical distance at most $\epsilon_G(\lambda)$ from uniform on G . The exponent sampler satisfies

$$\mathbb{E}_{\text{pp}} \left[\max_{e \in \mathcal{P}_{\text{pp}}} \Pr[\text{Sample}_e(\text{pp}) = e] \right] \leq \mu(\lambda),$$

where the expectation is over the complete RSA setup distribution, including its fallback. Both ϵ_G and μ are negligible. Every invocation uses fresh independent randomness. In particular, the exponent is sampled after the sender has fixed the entire ciphertext tag.

All these algorithms run in polynomial time, with fixed polynomial bounds on their randomness and output lengths. The functions $\ell_G, \ell_e, \ell_{\text{pp}}, \ell_{\text{td}}$ are fixed polynomials in λ . Group generation, sampling, root extraction, and the public group operations do not query the protocol's random oracles. Approximate sampling is explicit above; uniform sampling from a finite non-power-of-two set is not being assumed as a free bounded-time operation.

Strong RSA assumption. For $(\text{pp}, \text{td}) \leftarrow \text{Gen}_{\text{rt}}(1^\lambda)$ and uniform $U \in G$, define

$$\text{Adv}_{\text{root}}(\mathcal{A}) = \Pr[(v, e) \leftarrow \mathcal{A}(1^\lambda, \text{pp}, U) : v \in G, e > 1, v^e = U].$$

We use Adv_{root} to denote the strong RSA advantage. The strong RSA assumption states that this probability is negligible for every uniform PPT adversary without the trapdoor; the adversary chooses the exponent $e > 1$. The experiment uses the specified RSA modulus generation law and public sampling parameters, including any sampling fallbacks.

We work in the random-oracle model. The independently sampled oracles $H_0, H_1 : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$ are independent of RSA setup. All algorithms and adversaries use the same oracles; private queries do not expose timing, query counts, or cache information. The strong RSA reduction simulates these independent random oracles locally by lazy sampling, as described in the proof of Lemma C.3; the strong RSA experiment itself supplies no additional oracle.

3.2 The Tagged Batch-Decryption Construction

We instantiate the four-algorithm interface of Section 2.2. The public predicate Adm and the public recovery routine Recover below are construction-specific helpers used by these algorithms.

External tags are in $\{0, 1\}^*$ and messages are in $\{0, 1\}^\lambda$. Every polynomial-length tag is permitted, including repeated tags. We write all objects and hash inputs as tuples, with their ordinary unambiguous representations understood. No explicit serialization notation is needed. Length bounds below count every tuple component, including copies of the public key and of the tag.

Let $d_{\text{check}}, d_{\text{key}}, d_{\text{pad}}$ be fixed distinct domain-separation constants. Write

$$\begin{aligned} H_{\text{check}}(\text{pk}, t, e, C, x) &:= H_0(d_{\text{check}}, \text{pk}, t, e, C, x), \\ H_{\text{key}}(\text{pk}, X, x) &:= H_1(d_{\text{key}}, \text{pk}, X, x), \\ H_{\text{pad}}(\text{pk}, t, X, K) &:= H_1(d_{\text{pad}}, \text{pk}, t, X, K). \end{aligned}$$

Each output is a λ -bit string. None is interpreted as a scalar or an integer. All other hash uses have separate domains.

A ciphertext has the form

$$D = (\text{pk}, X, c), \quad X = (t, e, C, \rho).$$

The tag in X must agree with the external tag in a pair (t, D) . The core contains no application identity, signature key, signature, or independent nonce. Public parsing does not assert knowledge of an encryption witness. A successful response is a group element $W \in G$. The error response $\perp$ and empty-batch response empty are distinguished symbols outside G ; the identity 1_G is a valid successful hint.

$\text{Setup}(1^\lambda)$: *Run by the authority.*

- (1) Compute $(\text{pp}, \text{td}) \leftarrow \text{Gen}_{\text{rt}}(1^\lambda)$.
- (2) Set $\text{pk} = (\lambda, \text{pp})$ and $\text{sk} = (\text{td}, \text{pk})$.
- (3) Return (pk, sk) .

The authority publishes pk and keeps sk secret.

$\text{Enc}(\text{pk}, t, m)$: *Run by the sender.*

- (1) Check the public-key syntax, $t \in \{0, 1\}^*$, and $m \in \{0, 1\}^\lambda$. On failure, return $\perp$.
- (2) Independently sample $e \leftarrow \text{Sample}_e(\text{pp})$ and $x \leftarrow \text{Sample}_G(\text{pp})$, after t is fixed. Compute

$$\begin{aligned} C &= x^e, \\ \rho &= H_{\text{check}}(\text{pk}, t, e, C, x), \\ X &= (t, e, C, \rho), \quad K = H_{\text{key}}(\text{pk}, X, x). \end{aligned}$$

- (3) Compute $c = m \oplus H_{\text{pad}}(\text{pk}, t, X, K)$.
- (4) Return $D = (\text{pk}, X, c)$.

The sender submits the resulting ciphertext D to the authority for inclusion in a batch $\mathcal{B}$.

$\text{Adm}(\text{pk}, \mathcal{B})$: *Run by any party.*

- (1) Write $\mathcal{B} = ((t_1, D_1), \dots, (t_B, D_B))$. Reject malformed keys or ciphertext tuples, mismatching embedded public keys, unequal external and embedded tags, nonbinary tags, or checks and payloads that do not have length λ .
- (2) If $B = 0$, accept.
- (3) Check that $C_i \in G$ and $e_i \in \mathcal{P}_{\text{pp}}$ for every i , and that the e_i are pairwise distinct. Reject on failure.
- (4) Accept. There is no separate rejection for repeated tags.

$\text{PartDec}(\text{sk}, \mathcal{B})$: *Run by the authority.*

- (1) Read $\text{sk} = (\text{td}, \text{pk})$ and run $\text{Adm}(\text{pk}, \mathcal{B})$. Return $\perp$ on malformed input or rejection, and empty if $B = 0$.
- (2) For $i = 1, \dots, B$, in order:
 - (a) Compute $x_i \leftarrow \text{Root}(\text{pp}, \text{td}, e_i, C_i)$.
 - (b) If $H_{\text{check}}(\text{pk}, t_i, e_i, C_i, x_i) \neq \rho_i$, return $\perp$ immediately.
- (3) Compute $W = \prod_{i=1}^B x_i$ and return $T = W$.

Upon successful completion, publishes the resulting batch key T .

$\text{Recover}(\text{pk}, \mathcal{B}, T, i)$: *Run by any party with the batch key T .*

- (1) Check $1 \leq i \leq B$, run $\text{Adm}(\text{pk}, \mathcal{B})$, and require $T \in G$. On failure, return $\perp$. Set $W = T$.

(2) Compute the integers

$$E = \prod_{j=1}^B e_j, \quad E_i = E/e_i,$$

$$\beta_i = E_i(E_i^{-1} \bmod e_i).$$

Here $E_i^{-1} \bmod e_i$ is the least nonnegative modular inverse.

(3) Using only public group operations, compute

$$\hat{x}_i = W^{\beta_i} C_i^{(1-\beta_i)/e_i} \prod_{j \neq i} C_j^{-\beta_i/e_j}. \quad (1)$$

Every displayed exponent is an integer.¹

(4) Check $\hat{x}_i^{e_i} = C_i$ and $H_{\text{check}}(\mathbf{pk}, t_i, e_i, C_i, \hat{x}_i) = \rho_i$. On failure, return $\perp$.

(5) Set $K_i = H_{\text{key}}(\mathbf{pk}, X_i, \hat{x}_i)$ and return $m_i = c_i \oplus H_{\text{pad}}(\mathbf{pk}, t_i, X_i, K_i)$.

Dec($T, i, \mathcal{B}$): *Run by any party with the batch key T .*

(1) Read $\mathbf{pk}$ from the first ciphertext D_1 . Reject an empty or malformed batch, or inconsistent public-key copies, by returning $\perp$.

(2) Return $\text{Recover}(\mathbf{pk}, \mathcal{B}, T, i)$.

The embedded public key implements the same public decryption interface as Section 4, without an uncharged key lookup. Its size is counted below. The batch itself is already public input; the only new successful authority response is the single group element W .

3.3 Correctness and Compactness

Proposition 3.1 (Correctness) *For every fixed group, oracle realization, and choice of randomness, decryption is correct for every honest batch with distinct exponents, even when tags repeat. More generally, successful private validation guarantees correct recovery of every unmodified honest ciphertext in the batch. For any adaptive history with at most L honest encryptions, the probability, over setup and encryption randomness, that any selected batch with distinct invocation indices fails is at most*

$$\epsilon_{\text{corr}} \leq \min \left\{ 1, \binom{L}{2} \mu(\lambda) \right\}.$$

The bound permits overlapping batches and reuse of old hints.

proof[Proof sketch] Unique-root extraction recovers the sender's x_i . For distinct prime exponents, substituting $C_i = x_i^{e_i}$ and $W = \prod_j x_j$ into (1) gives

$$\hat{x}_i = \left(\prod_j x_j \right)^{\beta_i} x_i^{1-\beta_i} \prod_{j \neq i} x_j^{-\beta_i} = x_i.$$

¹Since the e_j are pairwise coprime, E_i is invertible modulo e_i , and the definition of β_i gives $\beta_i \equiv 1 \pmod{e_i}$. Hence $e_i \mid (1 - \beta_i)$. For $j \neq i$, we also have $e_j \mid E_i \mid \beta_i$, so $-\beta_i/e_j$ is an integer as well.

The same identity applies to the unique roots in any privately accepted batch. Conditional on the public descriptor, a union bound over the at most $\binom{L}{2}$ pairs of honest exponent samples bounds all collisions in the adaptive history; averaging over setup gives $\binom{L}{2}\mu(\lambda)$. Outside this event, all selected honest batches are correct simultaneously. Appendix C.2 gives the full argument, including coincident ciphertexts and reuse of earlier hints.

Complete object sizes. Put $p_0 = O(\ell_{\text{pp}} + \log(\lambda + 1))$, an upper bound on $|\text{pk}|$. With ordinary linear-overhead tuple representations, the complete sizes are as follows.

Object	Size in bits
Public key, including all public setup	p_0
Secret key, including its public-key copy	$O(\ell_{\text{td}} + p_0)$
Tagged ciphertext D	$O(p_0 + t + \ell_e + \ell_G + \lambda)$
Complete successful hint	ℓ_G
Error or empty-batch response	$O(1)$

All length functions in this table are evaluated at λ . The ciphertext count includes the embedded tag t , the group element C , the prime e , the check ρ , the message component c , and the embedded public key. The public pair (t, D) also includes its external tag copy, within the same asymptotic bound. There is no additional common reference string, public update, sender helper, or unaccounted decryption data.

Computation is not batch independent. Partial decryption performs public admission and at most B trapdoor root extractions and private hash checks. The integers E and β_i have $O(B\ell_e)$ bits. Public recovery consequently takes polynomial time in B , λ , and the full input length; it does not require taking roots without a hint. Compactness concerns communication and stored keys, not constant-time processing of a batch. Over Q queries the authority publishes at most $Q(\ell_G + O(1))$ bits.

3.4 Security Analysis

We prove adaptive-tag privacy in the sense of Definition 2.2. Throughout the proof, the only ideal oracles are the independent H_0, H_1 above; the group algorithms are independent of these oracles. An *ordinary query* is one made by the adversary or a public subroutine it invokes, or by simulated public validation. In particular, public encryption and recovery performed by the adversary contribute ordinary queries. The challenge encryption's three designated private evaluations of H_{check} , H_{key} , and H_{pad} , and the authority's private root checks, are distinguished from ordinary queries. They nevertheless remain part of the same random oracles and are charged in the running time and query bounds.

Theorem 3.2 *For every uniform PPT adversary $\mathcal{A}$ in the adaptive-tag experiment, there are uniform PPT root-extraction algorithms $\mathcal{R}_0, \mathcal{R}_1$ such that*

$$\begin{aligned} \Delta_{\text{Tag}}(\mathcal{A}) &\leq \sum_{b \in \{0,1\}} \text{Adv}_{\text{root}}(\mathcal{R}_b) + 2\epsilon_G(\lambda) \\ &\quad + 2(q_{\text{val}} + q_{\text{pad}})2^{-\lambda}. \end{aligned} \tag{2}$$

Here q_{val} bounds the total number of reached private root checks, and q_{pad} bounds ordinary queries to H_{pad} , including queries before the challenge. These are deterministic polynomial bounds covering the experiments and simulations below, with all auxiliary work included. The term ϵ_G is the group-sampling error in Section 3.1. Consequently, under the strong RSA assumption, the construction satisfies adaptive-tag privacy, and hence selective-tag privacy.

proof[Proof sketch] First replace the challenge root by a uniform element of G , at cost $\epsilon_G(\lambda)$. A simulator recognizes roots in ordinary check-oracle queries by testing $x^e = C$ and answers accepted release queries with their product. Rejecting every reached private check whose root has not appeared at the corresponding check coordinate changes the experiment by at most $q_{\text{val}}2^{-\lambda}$. Tag exclusion ensures that the published challenge check cannot supply a hidden value for a legal release query, including queries before the adaptive challenge.

In this simulation, replace the challenge-derived key by an independent uniform key. An ordinary query exposing its original key coordinate supplies a root of the embedded strong RSA challenge, so the change is bounded by $\text{Adv}_{\text{root}}(\mathcal{R}_b)$ for each challenge bit b . Replacing the message pad by an independent uniform pad costs at most $q_{\text{pad}}2^{-\lambda}$ and leaves a view independent of the challenge bit. Combining the bounds for both bits proves the theorem. Appendix C.3 gives the complete hybrids and reductions, preserving whole-batch admission, rejection, and overlapping releases throughout.

3.5 RSA Parameters and Compactness

Appendix C.1 specifies the bounded-time RSA setup and samplers, using $(\lambda + 2)$ -bit primes and exponents of at most $2\lambda + 4$ bits, with all fallback events included in their distributions.

We assume strong RSA for this complete RSA setup distribution. The independent random oracles used by the scheme are simulated inside the reduction, so the hardness assumption is the ordinary strong RSA assumption for this modulus generation law. The size bounds in Section 3.3 count the public-key copies required by the decryption interface and all public parameters. Here the tagged ciphertext has size $O(\lambda + |t|)$, the successful hint has size $O(\lambda)$, and both keys have size $O(\lambda)$. No batch-size bound is included in the keys or ciphertexts.

Corollary 3.3 (Ordinary epochless batch decryption) *Apply the CHK/Kiltz compiler of Appendix B to the tagged construction, using a strongly unforgeable one-time signature scheme with perfect correctness in the same oracle model. The resulting ordinary scheme has adaptive exact-ciphertext-exclusion privacy. For every uniform PPT adversary $\mathcal{A}$, there are uniform PPT algorithms $\mathcal{R}_0, \mathcal{R}_1, \mathcal{F}_0, \mathcal{F}_1$ such that*

$$\begin{aligned} \Delta_{\text{CT}}(\mathcal{A}) &\leq \sum_{b \in \{0,1\}} (\text{Adv}_{\text{root}}(\mathcal{R}_b) + \text{Adv}_{\text{SUF}}(\mathcal{F}_b)) + 2\epsilon_G(\lambda) \\ &\quad + 2(q_{\text{val}} + q_{\text{pad}})2^{-\lambda}. \end{aligned}$$

The polynomial query bounds cover the composed simulations, including prechallenge queries and signature work. Correctness for L honest invocations has error at most $\min\{1, \binom{L}{2}\mu(\lambda)\}$. Keys and successful hints have the same sizes as the core, and the ciphertext size is $O(\ell_{\text{pp}} + \ell_e + \ell_G + \lambda + v + s)$.

proof Theorem 3.2 supplies adaptive-tag privacy and its selective restriction. Theorem B.2 then gives the displayed bound. Proposition B.1 transfers the unconditional correctness bound from

Proposition 3.1. The compiler sets $t = vk$, so both tag copies have length $v(\lambda)$. No independent nonce or verification-key collision term is needed.

The domain-separated hash-then-Lamport instantiation in Appendix B.3 has $v = 2\lambda^2$ and $s = \lambda^2$. For the RSA lengths above this gives the complete sizes

$$|\text{CT}| = O(\lambda^2), \quad |T| = O(\lambda), \quad |\text{pk}|, |\text{sk}| = O(\lambda).$$

These bounds hold uniformly over every polynomial batch size and lifetime, without changing the scheme to suit those bounds.

A product-root hint authorizes recovery for its actual batch; it is not a key for future ciphertexts under a previously used tag. The construction neither assumes a pairing nor invokes garbling, but relies on the strong RSA assumption. It therefore does not resolve the feasibility or impossibility question for compact epochless batch decryption from assumptions such as CDH or DDH in publicly known prime-order groups.

4 Identity-Based Encryption with Batch Decryption

In this section, we define an identity-based encryption scheme with batch decryption. Informally, such a primitive allows one to derive, from a secret key and a batch of ciphertexts, a succinct batch key that enables decryption of any ciphertext in the batch while being shorter than the collection of the corresponding individual decryption keys. For brevity, we also refer to this primitive simply as batch decryption when the identity-based setting is clear. An identity-based encryption scheme with batch decryption for a set of identity spaces $\mathcal{I} = \{\{0, 1\}^n\}_{n \in \mathbb{N}}$, an epoch space $\mathcal{T}$, and a message space $\mathcal{M}$ containing a fixed dummy message 0 consists of the following algorithms.

$\text{Setup}(1^\lambda, 1^n)$: The setup algorithm takes as input the security parameter 1^λ and the identity length 1^n . It outputs a public key pk and a secret key sk .

$\text{Enc}(\text{pk}, \tau, \text{id}, m)$: The encryption algorithm takes as input a public key pk , an epoch $\tau \in \mathcal{T}$, an identity $\text{id} \in \{0, 1\}^n$, and a message $m \in \mathcal{M}$. It outputs a ciphertext CT .

$\text{PartDec}(\text{sk}, \tau, (\text{CT}_1, \dots, \text{CT}_\ell))$: The partial decryption algorithm takes as input a secret key sk , an epoch $\tau \in \mathcal{T}$, and a batch of ciphertexts $(\text{CT}_1, \dots, \text{CT}_\ell)$ such that $\text{id}(\text{CT}_1), \dots, \text{id}(\text{CT}_\ell)$ are pairwise distinct. It outputs a succinct batch key sbk .

$\text{Dec}(\text{sbk}, i^*, \text{CT}_1, \dots, \text{CT}_\ell)$: The decryption algorithm takes as input a succinct batch key sbk , an index $i^* \in [\ell]$, and a batch of ciphertexts $(\text{CT}_1, \dots, \text{CT}_\ell)$. It outputs either a message $m \in \mathcal{M}$ or $\perp$.

We additionally require ciphertexts to be self-identifying: there exists an efficient deterministic algorithm Id such that for every honestly generated ciphertext $\text{CT} \leftarrow \text{Enc}(\text{pk}, \tau, \text{id}, m)$, we have $\text{Id}(\text{CT}) = \text{id}$.

Correctness. For correctness, we require that there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda, n \in \mathbb{N}$, all epochs $\tau \in \mathcal{T}$, all pairwise distinct identities $\text{id}_1, \dots, \text{id}_\ell \in \{0, 1\}^n$, all messages $m_1, \dots, m_\ell \in \mathcal{M}$, and all indices $i^* \in [\ell]$,

$$\Pr \left[\text{Dec}(\text{sbk}, i^*, \text{CT}_1, \dots, \text{CT}_\ell) = m_{i^*} \mid \begin{array}{l} (\text{pk}, \text{sk}) \leftarrow \text{Setup}(1^\lambda, 1^n) \\ \text{CT}_i \leftarrow \text{Enc}(\text{pk}, \tau, \text{id}_i, m_i) \text{ for all } i \in [\ell] \\ \text{sbk} \leftarrow \text{PartDec}(\text{sk}, \tau, (\text{CT}_1, \dots, \text{CT}_\ell)) \end{array} \right] \geq 1 - \text{negl}(\lambda).$$

where the probability is over the randomness of Setup , Enc , and PartDec .

Security. We define standard indistinguishability under chosen-plaintext attacks in the presence of an oracle for partial decryption. Consider the following experiment with an adversary $\mathcal{A}$.

- Initialization Phase: $\mathcal{A}$ on input 1^λ outputs 1^n . The challenger computes $(\text{pk}, \text{sk}) \leftarrow \text{Setup}(1^\lambda, 1^n)$ and sends pk to $\mathcal{A}$.
- Pre-Challenge Query Phase: In this phase, $\mathcal{A}$ is allowed to make multiple queries $(\tau, (\text{CT}_1, \dots, \text{CT}_\ell))$, where $\text{Id}(\text{CT}_1), \dots, \text{Id}(\text{CT}_\ell)$ are pairwise distinct. For each such query, the challenger returns

$$\text{sbk} \leftarrow \text{PartDec}(\text{sk}, \tau, (\text{CT}_1, \dots, \text{CT}_\ell)).$$

- Challenge Phase: The adversary $\mathcal{A}$ sends an epoch τ^* , a challenge identity $\text{id}^* \in \{0, 1\}^n$, and two challenge messages $m_0^*, m_1^* \in \mathcal{M}$. If more than one PartDec query on epoch τ^* was made in the pre-challenge query phase, or if the unique such query, when it exists, contains a ciphertext CT such that $\text{Id}(\text{CT}) = \text{id}^*$, then the challenger aborts. Otherwise, it samples a random bit $b \leftarrow \{0, 1\}$, computes $\text{CT}^* \leftarrow \text{Enc}(\text{pk}, \tau^*, \text{id}^*, m_b^*)$, and sends CT^* to $\mathcal{A}$.
- Post-Challenge Query Phase: In this phase, $\mathcal{A}$ is allowed to make multiple queries $(\tau, (\text{CT}_1, \dots, \text{CT}_\ell))$, where $\text{Id}(\text{CT}_1), \dots, \text{Id}(\text{CT}_\ell)$ are pairwise distinct. For each such query, if $\tau = \tau^*$ and either a PartDec query on epoch τ^* was already answered in the pre-challenge query phase or a previous post-challenge query was already made on epoch τ^* , or $\text{CT}^* \in \{\text{CT}_1, \dots, \text{CT}_\ell\}$, or $\text{id}^* \in \{\text{Id}(\text{CT}_1), \dots, \text{Id}(\text{CT}_\ell)\}$, then the challenger aborts. Otherwise, it returns

$$\text{sbk} \leftarrow \text{PartDec}(\text{sk}, \tau, (\text{CT}_1, \dots, \text{CT}_\ell)).$$

- Response Phase: $\mathcal{A}$ outputs a bit b' .

Definition 4.1 *An identity-based encryption scheme with batch decryption is said to be secure if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that, for all $\lambda \in \mathbb{N}$,*

$$\Pr[b' = b] \leq \frac{1}{2} + \text{negl}(\lambda),$$

where the probability is over the randomness of $\mathcal{A}$, Setup , Enc , PartDec , and the challenger.

The simultaneous multi-challenge variant is recorded in [Appendix D.7](#).

4.1 Registration-Based Encryption

We use a static, keyless registration-based encryption scheme $\text{RBE} = (\text{Gen}, \text{Hash}, \text{Hint}, \text{Enc}, \text{Dec})$ with non-inclusion security. The hash depends only on a tuple of pairwise distinct identities, and there is no dynamic update interface. [Appendix A.4](#) gives the full definition and explains its relation to standard RBE.

The algorithm $\text{Gen}(1^\lambda)$ generates public parameters pp . For a tuple $I = (\text{id}_1, \dots, \text{id}_\ell)$ with $\text{id}_i \in \{0, 1\}^n$, the algorithm $\text{Hash}(\text{pp}, I)$ returns $h \in \{0, 1\}^k$, where $k = k(\lambda, n)$ is independent of the

batch size. Given $\text{id}_i \in I$, the public algorithm $\text{Hint}(\text{pp}, \text{id}_i, I)$ returns (h', η_i) . Encryption produces $\text{ct} \leftarrow \text{Enc}(\text{pp}, h, \text{id}, m)$, and decryption computes $\text{Dec}(\text{pp}, \eta_i, \text{ct})$. Correctness requires that $h' = h$ and that decryption recovers m for $\text{id} = \text{id}_i$, except with negligible probability.

Non-inclusion security protects an encryption under $h = \text{Hash}(\text{pp}, I)$ to a challenge identity $\text{id}^* \notin I$, even when the adversary chooses I , id^* , and the challenge messages after seeing pp (Definition A.8). In our construction, the ciphertext defers this encryption until the batch fixes h . The next ingredient allows the keys selecting that hash input to be revealed adaptively.

4.2 Receiver Non-Committing Identity-Based Encryption

We use ordinary IBE algorithms together with a simulator that generates public parameters and a challenge ciphertext for an adaptively chosen, previously unqueried identity without knowing its message. Once the message is supplied, the simulator produces a user secret key that explains the ciphertext consistently. The adversary may request keys for other identities before and after the challenge-key reveal. Security requires this simulated view to be indistinguishable from honest encryption and key generation. Neither the master secret key nor algorithm randomness is revealed. Appendix D.1 gives the full interface and security experiment.

Theorem 4.2 *Assuming that IBE is an adaptively secure IBE scheme and PRF is a secure pseudo-random function, the construction in Appendix D.2 is a secure weak RNC-IBE scheme.*

4.3 Batch Construction

Assume that $\mathcal{T} = \{0, 1\}^t$ for some $t = t(\lambda)$, let $\text{IBE} = (\text{Setup}, \text{Keygen}, \text{Enc}, \text{Dec})$ be a weak RNC-IBE scheme for identity space $\mathcal{T}$ and message space given by the garbling label space, and let RBE be a keyless registration based encryption scheme. Let $k = k(\lambda, n)$ denote the output length of the hash algorithm of RBE for identities in $\{0, 1\}^n$. The construction is as follows.

$\text{Setup}(1^\lambda, 1^n)$: *Run by the authority.*

- (1) For every $j \in [k]$ and $b \in \{0, 1\}$, compute $(\text{mpk}_{j,b}, \text{msk}_{j,b}) \leftarrow \text{IBE.Setup}(1^\lambda, 1^t)$.
- (2) Compute $\text{pp} \leftarrow \text{RBE.Gen}(1^\lambda)$
- (3) Output $\text{pk} = (\text{pp}, (\text{mpk}_{j,b})_{j \in [k], b \in \{0, 1\}})$ and $\text{sk} = (\text{pp}, (\text{msk}_{j,b})_{j \in [k], b \in \{0, 1\}})$.

The authority publishes pk and keeps sk secret.

$\text{Enc}(\text{pk}, \tau, \text{id}, m)$: *Run by the sender.*

- (1) Parse $\text{pk} = (\text{pp}, (\text{mpk}_{j,b})_{j \in [k], b \in \{0, 1\}})$.
- (2) Let $\text{C}[\text{pp}, \text{id}, m, r](h)$ be a boolean circuit which has pp , the message m and random coins $r \leftarrow_{\$} \{0, 1\}^\lambda$ hardwired. It takes as input a hash value $h \in \{0, 1\}^k$ and computes and outputs $\text{RBE.Enc}(\text{pp}, h, \text{id}, m; r)$.
- (3) Compute the garbling $(\text{GC}, (\text{lab}_{j,b})_{j \in [k], b \in \{0, 1\}}) \leftarrow \text{Garble}(1^\lambda, \text{C}[\text{pp}, \text{id}, m, r])$.
- (4) For $j \in [k]$ and $b \in \{0, 1\}$ compute $\text{ct}_{j,b} \leftarrow \text{IBE.Enc}(\text{mpk}_{j,b}, \tau, \text{lab}_{j,b})$.
- (5) Output $\text{CT} \leftarrow (\text{id}, (\text{ct}_{j,b})_{j \in [k], b \in \{0, 1\}}, \text{GC})$

The sender submits the resulting ciphertext CT to the authority.

$\text{PartDec}(\text{sk}, \tau, (\text{CT}_1, \dots, \text{CT}_m))$: Run by the authority.

- (1) Parse $\text{sk} = (\text{pp}, (\text{msk}_{j,b})_{j \in [k], b \in \{0,1\}})$ and $\text{CT}_i = (\text{id}_i, (\text{ct}_{i,j,b})_{j \in [k], b \in \{0,1\}}, \text{GC}_i)$ for each $i \in [m]$.
- (2) Compute $h = \text{RBE.Hash}(\text{pp}, (\text{id}_1, \dots, \text{id}_m))$.
- (3) For $j \in [k]$ compute $\sigma_j = \text{IBE.Keygen}(\text{msk}_{j,h_j}, \tau)$.
- (4) Output $\text{sbk} = (\text{pp}, \tau, \sigma_1, \dots, \sigma_k)$

Upon successful completion, publishes the resulting batch key sbk .

$\text{Dec}(\text{sbk}, i^*, \text{CT}_1, \dots, \text{CT}_m)$: Run by any party with the batch key sbk . On input a succinct batch key sbk , and an index i^* and ciphertexts $\text{CT}_1, \dots, \text{CT}_m$ proceed as follows.

- (1) Parse $\text{sbk} = (\text{pp}, \tau, \sigma_1, \dots, \sigma_k)$
- (2) Parse $\text{CT}_i = (\text{id}_i, (\text{ct}_{i,j,b})_{j \in [k], b \in \{0,1\}}, \text{GC}_i)$ for each $i \in [m]$.
- (3) Compute $(h, \eta) = \text{RBE.Hint}(\text{pp}, \text{id}_{i^*}, (\text{id}_1, \dots, \text{id}_m))$.
- (4) Compute $\hat{\text{lab}}_j = \text{IBE.Dec}(\sigma_j, \text{ct}_{i^*,j,h_j})$ for each $j \in [k]$.
- (5) Evaluate the garbled circuit GC_{i^*} on the labels $\hat{\text{lab}}_j$ to obtain ciphertext c .
- (6) Output $m = \text{RBE.Dec}(\text{pp}, \eta, c)$.

Correctness.

Lemma 4.3 *Assume that the weak RNC-IBE scheme IBE, garbling scheme GC, and registration-based encryption scheme RBE are correct. If, for every public parameter pp , identity tuple $I = (\text{id}_1, \dots, \text{id}_m)$, and index $i^* \in [m]$, the algorithm $\text{RBE.Hint}(\text{pp}, \text{id}_{i^*}, I)$ always returns $\text{RBE.Hash}(\text{pp}, I)$ as its first component, then the above batch decryption construction is correct.*

proof Hash/Hint consistency makes decryption use the same hash as batch-key generation. Correctness of weak RNC-IBE, garbling, and RBE then recovers the corresponding labels, RBE ciphertext, and message, respectively, with negligible total error by a union bound. Appendix D.4 gives the detailed argument.

Security. We now establish the security of our batch decryption construction according to Definition 4.1. The proof proceeds via a sequence of computationally indistinguishable hybrid experiments, demonstrating that any PPT adversary capable of breaking the scheme can be used to break the underlying security of the GC, RBE, or weak RNC-IBE primitives.

Theorem 4.4 *Assume that IBE is a secure weak RNC-IBE scheme in the sense of Definition D.1, GC is an adaptive secure garbling scheme, and RBE is a non-inclusion secure registration based encryption scheme. Then the above construction is secure in the sense of Definition 4.1.*

proof Let $\mathcal{A}$ be a PPT adversary in the security game of Definition 4.1. We prove the theorem by considering a sequence of hybrids $\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3$. The detailed reductions are given in Appendix D.5.

Hybrid $\mathcal{H}_0$: This is the original experiment of Definition 4.1.

Hybrid $\mathcal{H}_1$: Between $\mathcal{H}_0$ and $\mathcal{H}_1$, we define intermediate hybrids that switch the weak RNC-IBE ciphertexts $\text{ct}_{j,b}^*$ contained in the challenge ciphertext CT^* to simulated mode one instance at a time. Since each pair (j, b) uses its own independent weak RNC-IBE master public key, each step in this hybrid sequence changes exactly one challenge ciphertext in one weak RNC-IBE instance. Hybrid $\mathcal{H}_1$ is the experiment in which every challenge weak RNC-IBE component that need not remain decryptable under a pre-challenge succinct batch key has been switched to simulated mode. Concretely, if no succinct batch key for τ^* is ever obtained, or if it is obtained only after the challenge, then all weak RNC-IBE ciphertexts in CT^* are switched to simulated mode. If the unique succinct batch key for τ^* is obtained before the challenge, then only the sibling components $\text{ct}_{j,1-h_j^*}^*$ are switched to simulated mode, while the decryptable components $\text{ct}_{j,h_j^*}^*$ remain honest. The indistinguishability of $\mathcal{H}_0$ and $\mathcal{H}_1$ follows from the single-challenge security of the weak RNC-IBE scheme and a hybrid argument over the $2k$ challenge components. In the proofs of Lemma D.4 and Lemma D.5 we distinguish whether the unique succinct batch key for the challenge epoch τ^* is obtained before the challenge, after the challenge, or never. The details are given in Lemma D.4.

Hybrid $\mathcal{H}_2$: This hybrid proceeds exactly as $\mathcal{H}_1$, except that the garbled circuit GC^* contained in the challenge ciphertext CT^* is generated by the garbling simulator rather than by honestly garbling the circuit. Let h^* be the hash value corresponding to the unique batch on which the adversary queries the `PartDec` oracle at epoch τ^* , if such a query exists. If no such query exists, let

$$h^* = \text{RBE.Hash}(\text{pp}, (\hat{\text{id}}_1, \dots, \hat{\text{id}}_m))$$

for any fixed tuple of pairwise distinct identities that does not contain id^* . Since in $\mathcal{H}_1$ the weak RNC-IBE ciphertexts of CT^* are already simulated, labels from CT^* are only revealed by providing the corresponding weak RNC-IBE secret keys. By definition of the security game, the adversary can obtain at most one succinct batch key for the challenge epoch τ^* . This key may have been obtained either before or after the challenge, but in either case it determines only one hash input for CT^* . Hence the adversary can recover from CT^* labels corresponding to at most one garbling input. Therefore, we may invoke adaptive security of the garbling scheme. Concretely, we provide the value $\text{RBE.Enc}(\text{pp}, h^*, \text{id}^*, m_b^*)$ as the output value to the label simulator. The indistinguishability of $\mathcal{H}_1$ and $\mathcal{H}_2$ follows from adaptive security of the garbling scheme. The details are given in Lemma D.5.

Hybrid $\mathcal{H}_3$: This hybrid proceeds exactly as $\mathcal{H}_2$, except that the RBE ciphertext $\text{RBE.Enc}(\text{pp}, h^*, \text{id}^*, m_b^*)$ underlying the simulated garbled circuit is replaced by $\text{RBE.Enc}(\text{pp}, h^*, \text{id}^*, 0)$. The indistinguishability of $\mathcal{H}_2$ and $\mathcal{H}_3$ follows from non-inclusion security of the RBE scheme. Here we use that the unique admissible succinct batch key at the challenge epoch τ^* , if it exists, is associated with a batch that contains no ciphertext under identity id^* . Hence id^* does not belong to the identity set hashed into h^* , which is exactly the non-inclusion setting needed to apply security of the RBE scheme. The details are given in Lemma D.6.

In the final hybrid $\mathcal{H}_3$, the challenge ciphertext is independent of the challenge bit b , and therefore the adversary's advantage is 0.

Combining Lemmas D.4 to D.6 and using the triangle inequality, we conclude that for every PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathcal{H}_0(\lambda, \mathcal{A}) = 1] - \Pr[\mathcal{H}_3(\lambda, \mathcal{A}) = 1]| \leq \text{negl}(\lambda)$. Since $\mathcal{H}_3$ is independent of b , this proves the theorem.

4.4 Instantiations from Standard Assumptions

CDH implies one-way functions, which suffice for adaptively secure garbling [BHR12a, HJO⁺16] and pseudorandom functions. Standard IBE [DG17b] and the required non-inclusion secure RBE [GHMR18, GHM⁺19] are also available under CDH. Theorem 4.2 supplies the weak RNC-IBE layer, so Theorem 4.4 gives the following corollary. Appendix D.6 discusses alternative instantiation paths.

Corollary 4.5 *Assuming the hardness of the Computational Diffie-Hellman problem, there exists an adaptively secure identity-based encryption scheme with batch decryption.*

References

- [ADG⁺26] Amit Agarwal, Sourav Das, Babak Poorebrahim Gilkalaye, Peter Rindal, and Victor Shoup. BTX: Simple and efficient batch threshold encryption. Cryptology ePrint Archive, Paper 2026/754, 2026.
- [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.
- [BB08] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, April 2008.
- [BBD⁺20] Zvika Brakerski, Pedro Branco, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Constant ciphertext-rate non-committing encryption from standard assumptions. In *Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020, Proceedings, Part I 18*, pages 58–87. Springer, 2020.
- [BDD22] Pedro Branco, Nico Döttling, and Jesko Dujmović. Rate-1 incompressible encryption from standard assumptions. In *Theory of Cryptography Conference*, pages 33–69. Springer, 2022.
- [Bea97] Donald Beaver. Plug and play encryption. In *Annual International Cryptology Conference*, pages 75–89. Springer, 1997.
- [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 – EURO-CRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 533–556, Copenhagen, Denmark, May 11–15, 2014. Springer Berlin Heidelberg, Germany.
- [BHR12a] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658

of *Lecture Notes in Computer Science*, pages 134–153, Beijing, China, December 2–6, 2012. Springer Berlin Heidelberg, Germany.

- [BHR12b] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012: 19th Conference on Computer and Communications Security*, pages 784–796, Raleigh, NC, USA, October 16–18, 2012. ACM Press.
- [BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004.
- [BLSV18] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous ibe, leakage resilience and circular security from new assumptions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 535–564. Springer, 2018.
- [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.
- [BNRT26] Dan Boneh, Rohit Nema, Arnab Roy, and Ertem Nusret Tas. Efficient batch threshold encryption using partial fraction techniques. *Cryptology ePrint Archive*, Paper 2026/674, 2026.
- [CDG⁺17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 33–65, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland.
- [CDSMW09] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Improved non-committing encryption with applications to adaptively secure protocols. In *Advances in Cryptology–ASIACRYPT 2009: 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings 15*, pages 287–302. Springer, 2009.
- [CFGN96] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 639–648, 1996.
- [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 Wenyan Xu, editors, *USENIX Security 2024: 33rd USENIX Security Symposium*, Philadelphia, PA, USA, August 14–16, 2024. USENIX Association.
- [CHK04] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 207–222, Interlaken, Switzerland, May 2–6, 2004. Springer Berlin Heidelberg, Germany.

- [CHK05] Ran Canetti, Shai Halevi, and Jonathan Katz. Adaptively-secure, non-interactive public-key encryption. In *Theory of Cryptography Conference*, pages 150–168. Springer, 2005.
- [CPR17] Ran Canetti, Oxana Poburinnaya, and Mariana Raykova. Optimal-rate non-committing encryption. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 212–241. Springer, 2017.
- [DG17a] Nico Döttling and Sanjam Garg. From selective ibe to full ibe and selective hibe. In *Theory of Cryptography Conference*, pages 372–408. Springer, 2017.
- [DG17b] Nico Döttling and Sanjam Garg. Identity-based encryption from the Diffie-Hellman assumption. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 537–569, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland.
- [DG17c] Nico Döttling and Sanjam Garg. Identity-based encryption from the diffie-hellman assumption. In *Annual international cryptology conference*, pages 537–569. Springer, 2017.
- [DGHM18] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Daniel Masny. New constructions of identity-based and key-dependent message secure encryption schemes. In *IACR International Workshop on Public Key Cryptography*, pages 3–31. Springer, 2018.
- [DHMW23] Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wahnig. McFly: Verifiable encryption to the future made practical. In Foteini Baldimtsi and Christian Cachin, editors, *FC 2023: 27th International Conference on Financial Cryptography and Data Security, Part I*, volume 13950 of *Lecture Notes in Computer Science*, pages 252–269, Bol, Brač, Croatia, May 1–5, 2023. Springer, Cham, Switzerland.
- [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 – EURO-CRYPT 2023, Part III*, volume 14006 of *Lecture Notes in Computer Science*, pages 417–446, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.
- [DN00] Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In *Advances in Cryptology—CRYPTO 2000: 20th Annual International Cryptology Conference Santa Barbara, California, USA, August 20–24, 2000 Proceedings 20*, pages 432–450. Springer, 2000.
- [Fia97] Amos Fiat. Batch RSA. *Journal of Cryptology*, 10(2):75–88, March 1997.
- [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.
- [GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

- [GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 467–476, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.
- [GHK⁺25] Sanjam Garg, Mohammad Hajiabadi, Dimitris Kolonelos, Abhiram Kothapalli, and Guru-Vamsi Policharla. A framework for witness encryption from linearly verifiable SNARKs and 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 504–539, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland.
- [GHM⁺19] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmood, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In Dongdai Lin and Kazue Sako, editors, *PKC 2019: 22nd International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 11443 of *Lecture Notes in Computer Science*, pages 63–93, Beijing, China, April 14–17, 2019. Springer, Cham, Switzerland.
- [GHMR18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmood, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018: 16th Theory of Cryptography Conference, Part I*, volume 11239 of *Lecture Notes in Computer Science*, pages 689–718, Panaji, India, November 11–14, 2018. Springer, Cham, Switzerland.
- [GKK⁺25] Rishab Goyal, Fuyuki Kitagawa, Venkata Koppula, Ryo Nishimaki, Mahesh Sreekumar Rajasree, and Takashi Yamakawa. Non-committing identity based encryption: constructions and applications. In *IACR International Conference on Public-Key Cryptography*, pages 158–193. Springer, 2025.
- [GKRV25] Rishab Goyal, Venkata Koppula, Mahesh Sreekumar Rajasree, and Aman Verma. Incompressible Functional Encryption. In Raghu Meka, editor, *16th Innovations in Theoretical Computer Science Conference (ITCS 2025)*, volume 325 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 56:1–56:22, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- [GV20] Rishab Goyal and Satyanarayana Vusirikala. Verifiable registration-based encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 621–651, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.
- [GWW26] Junqing Gong, Brent Waters, Hoeteck Wee, and David J. Wu. Threshold batched identity-based encryption from pairings in the plain model. In *EUROCRYPT (5)*, *Lecture Notes in Computer Science*, pages 548–578. Springer, 2026.
- [GWZ22] Jiaxin Guan, Daniel Wichs, and Mark Zhandry. Incompressible Cryptography. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022*, *Lecture Notes in Computer Science*, pages 700–730, Cham, 2022. Springer International Publishing.

- [GWZ23] Jiaxin Guan, Daniel Wichs, and Mark Zhandry. Multi-instance randomness extraction and security against bounded-storage mass surveillance. In *Theory of Cryptography Conference*, pages 93–122. Springer, 2023.
- [HJO⁺16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 149–178, Santa Barbara, CA, USA, August 14–18, 2016. Springer Berlin Heidelberg, Germany.
- [HMNY21] Taiga Hiroka, Tomoyuki Morimae, Ryo Nishimaki, and Takashi Yamakawa. Quantum encryption with certified deletion, revisited: Public key, attribute-based, and classical communication. In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part I 27*, pages 606–636. Springer, 2021.
- [HOR15] Brett Hemenway, Rafail Ostrovsky, and Alon Rosen. Non-committing encryption from ϕ -hiding. In *Theory of Cryptography Conference*, pages 591–608. Springer, 2015.
- [HORR15] Brett Hemenway, Rafail Ostrovsky, Silas Richelson, and Alon Rosen. Adaptive security with quasi-optimal rate. In *Theory of Cryptography Conference*, pages 525–541. Springer, 2015.
- [JL00] Stanislaw Jarecki and Anna Lysyanskaya. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In *Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19*, pages 221–242. Springer, 2000.
- [JO20] Zahra Jafargholi and Sabine Oechsner. Adaptive security of practical garbling schemes. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *Progress in Cryptology - INDOCRYPT 2020: 21st International Conference in Cryptology in India*, volume 12578 of *Lecture Notes in Computer Science*, pages 741–762, Bangalore, India, December 13–16, 2020. Springer, Cham, Switzerland.
- [JW16] Zahra Jafargholi and Daniel Wichs. Adaptive security of Yao’s garbled circuits. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part I*, volume 9985 of *Lecture Notes in Computer Science*, pages 433–458, Beijing, China, October 31 – November 3, 2016. Springer Berlin Heidelberg, Germany.
- [Kil06] Eike Kiltz. Chosen-ciphertext security from tag-based encryption. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 581–600, New York, NY, USA, March 4–7, 2006. Springer Berlin Heidelberg, Germany.
- [KT18] Fuyuki Kitagawa and Keisuke Tanaka. Key dependent message security and receiver selective opening security for identity-based encryption. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018: 21st International Conference on Theory and Practice of*

- Public Key Cryptography, Part I*, volume 10769 of *Lecture Notes in Computer Science*, pages 32–61, Rio de Janeiro, Brazil, March 25–29, 2018. Springer, Cham, Switzerland.
- [KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194, Singapore, December 5–9, 2010. Springer Berlin Heidelberg, Germany.
- [LY10] Benoît Libert and Moti Yung. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 499–517, Zurich, Switzerland, February 9–11, 2010. Springer Berlin Heidelberg, Germany.
- [Pol26] Guru-Vamsi Policharla. A simple batched threshold encryption scheme. *IACR Cryptol. ePrint Arch.*, 2026:760, 2026.
- [WW23] Brent Waters and Daniel Wichs. Universal amplification of kdm security: From 1-key circular to multi-key kdm. In *Annual International Cryptology Conference*, pages 674–693. Springer, 2023.
- [Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.
- [YKT19] Yusuke Yoshida, Fuyuki Kitagawa, and Keisuke Tanaka. Non-committing encryption with quasi-optimal ciphertext-rate based on the ddh problem. In *Advances in Cryptology–ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part III 25*, pages 128–158. Springer, 2019.
- [YKXT20] Yusuke Yoshida, Fuyuki Kitagawa, Keita Xagawa, and Keisuke Tanaka. Non-committing encryption with constant ciphertext expansion from standard assumptions. In *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II 26*, pages 36–65. Springer, 2020.

Declaration on the Use of AI Tools

OpenAI tools assisted with language editing and with the technical investigation, drafting, and automated consistency checking of the definitions and proofs. The connection between a preliminary version of the epochless construction in RSA groups and Fiat’s batch RSA [Fia97] was made by GPT 6 Astra. The authors retain responsibility for the final manuscript’s content, claims, proofs, and conclusions.

A Additional Preliminaries

A.1 Pseudorandom Functions

A family of pseudorandom functions $\text{PRF} = (\text{Setup}, \text{Eval})$ with key space $\{\{0, 1\}^\lambda\}_\lambda$, input space $\{\{0, 1\}^{m(\lambda)}\}_\lambda$ and output space $\{\{0, 1\}^{n(\lambda)}\}_\lambda$ consists of the following algorithms.

- $\text{Setup}(1^\lambda)$: The key generation algorithm is a randomized algorithm that takes as input the security parameter 1^λ and outputs a key $k \in \{0, 1\}^\lambda$.
- $\text{Eval}(k, x)$: The evaluation algorithm is a deterministic algorithm that takes as input a key $k \in \{0, 1\}^\lambda$ and $x \in \{0, 1\}^{m(\lambda)}$ and outputs $y \in \{0, 1\}^{n(\lambda)}$.

Definition A.1 *A PRF scheme PRF is secure if for all PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathcal{A}^{\text{Eval}(k, \cdot)}(1^\lambda) = 1 : k \leftarrow \text{Setup}(1^\lambda)] - \Pr[\mathcal{A}^{R(\cdot)}(1^\lambda) = 1 : R \leftarrow \mathcal{U}_\lambda]| \leq \text{negl}(\lambda)$$

where $\mathcal{U}_\lambda$ is the set of all functions from $\{0, 1\}^{m(\lambda)}$ to $\{0, 1\}^{n(\lambda)}$.

Theorem A.2 ([GGM86]) *Assuming the existence of one-way functions, there exists secure PRF for any polynomials $m(\cdot), n(\cdot)$.*

A.2 Garbling Schemes

A garbling scheme $\text{GC} = (\text{GarbleC}, \text{GarbleI}, \text{Eval}, \text{SimC}, \text{SimI})$ for a class of circuits $\{\mathcal{C}_\lambda\}_\lambda$ consists of the following algorithms.

- $\text{GarbleC}(1^\lambda, C)$: The circuit garbling algorithm is a randomized algorithm that takes as input the security parameter 1^λ and a circuit $C \in \mathcal{C}_\lambda$ such that $C : \{0, 1\}^n \rightarrow \{0, 1\}^m$ and outputs a garbled circuit $\hat{C}$ and a key k .
- $\text{GarbleI}(k, x)$: The input garbling algorithm is a randomized algorithm that takes as input a key k and a input $x \in \{0, 1\}^n$ and outputs $\{\text{lab}_i\}_{i \in [n]}$.
- $\text{Eval}(\hat{C}, \{\text{lab}_i\}_{i \in [n]})$: The evaluation algorithm takes as input a garbled circuit $\hat{C}$ and a set of n labels $\{\text{lab}_i\}_{i \in [n]}$ and outputs $y \in \{0, 1\}^m$.
- $\text{SimC}(1^\lambda, 1^n, 1^m, 1^s)$: The circuit simulation algorithm takes as input a security parameter 1^λ , input length parameter 1^n , output length parameter 1^m , size parameter 1^s and outputs a garbled circuit $\hat{C}$ and a state st .
- $\text{SimI}(\text{st}, y)$: The input simulation algorithm takes as input a state st and an output value $y \in \{0, 1\}^m$ and outputs $\{\text{lab}_i\}_{i \in [n]}$.

Remark A.3 *For readability, we also use the algorithm Garble , which combines GarbleC and GarbleI as follows. On input the security parameter 1^λ and a circuit $C \in \mathcal{C}_\lambda$ with $C : \{0, 1\}^n \rightarrow \{0, 1\}^m$, the randomized algorithm $\text{Garble}(1^\lambda, C)$ outputs the garbled circuit $\hat{C}$ together with the set of labels $\{\text{lab}_{i,b}\}_{i \in [n], b \in \{0, 1\}}$. Similarly, we use Sim to denote the algorithm that combines SimC and SimI .*

Correctness. For correctness of a garbling scheme GC for a class of circuits $\{\mathcal{C}_\lambda\}_\lambda$, we require that for all $\lambda \in \mathbb{N}$, $C \in \mathcal{C}_\lambda$, $x \in \{0, 1\}^n$, $(\hat{C}, \mathbf{k}) \leftarrow \text{GarbleC}(1^\lambda, C)$, $\{\text{lab}_{i,x_i}\}_{i \in [n]} \leftarrow \text{Garblel}(\mathbf{k}, x)$,

$$\text{Eval}(\hat{C}, \{\text{lab}_{i,x_i}\}_{i \in [n]}) = C(x)$$

Selective Security. For selective security, we define simulation based security [Yao86, BHR12b].

Definition A.4 A garbling scheme $\text{GC} = (\text{Garble}, \text{Eval}, \text{Sim})$ for a class of circuits $\{\mathcal{C}_\lambda\}_\lambda$ is said to be selective secure if for all PPT stateful adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds.

$$\left| \Pr \left[\mathcal{A}(\hat{C}, \{\text{lab}_{i,x_i}\}_{i \in [n]}) = 1 : \begin{array}{l} (C, x) \leftarrow \mathcal{A}(1^\lambda), \\ (\hat{C}, \{\text{lab}_{i,b}\}_{i \in [n], b \in \{0,1\}}) \leftarrow \text{Garble}(1^\lambda, C) \end{array} \right] \right. \\ \left. - \Pr \left[\mathcal{A}(\hat{C}, \{\text{lab}_i\}_{i \in [n]}) = 1 : \begin{array}{l} (C, x) \leftarrow \mathcal{A}(1^\lambda), \\ (\hat{C}, \{\text{lab}_i\}_{i \in [n]}) \leftarrow \text{Sim}(1^\lambda, 1^n, 1^m, 1^{|C|}, C(x)) \end{array} \right] \right| \leq \text{negl}(\lambda).$$

where the probability is over the randomness of $\mathcal{A}$, Sim and GC algorithms.

Adaptive Security. For adaptive security, we define simulation based security [BHR12a, HJO⁺16, JW16, JO20].

Definition A.5 A garbling scheme $\text{GC} = (\text{GarbleC}, \text{Garblel}, \text{Eval}, \text{SimC}, \text{SimI})$ for a class of circuits $\{\mathcal{C}_\lambda\}_\lambda$ is said to be adaptive secure if for all PPT stateful adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds.

$$\left| \Pr \left[\mathcal{A}(\hat{C}, \{\text{lab}_{i,x_i}\}_{i \in [n]}) = 1 : \begin{array}{l} C \leftarrow \mathcal{A}(1^\lambda), \\ (\hat{C}, \mathbf{k}) \leftarrow \text{GarbleC}(1^\lambda, C) \\ x \leftarrow \mathcal{A}(\hat{C}) \\ \{\text{lab}_{i,x_i}\}_{i \in [n]} \leftarrow \text{Garblel}(\mathbf{k}, x) \end{array} \right] \right. \\ \left. - \Pr \left[\mathcal{A}(\hat{C}, \{\text{lab}_i\}_{i \in [n]}) = 1 : \begin{array}{l} C \leftarrow \mathcal{A}(1^\lambda), \\ (\hat{C}, \text{st}) \leftarrow \text{SimC}(1^\lambda, 1^n, 1^m, 1^{|C|}) \\ x \leftarrow \mathcal{A}(\hat{C}) \\ \{\text{lab}_i\}_{i \in [n]} \leftarrow \text{SimI}(\text{st}, C(x)) \end{array} \right] \right| \leq \text{negl}(\lambda).$$

where the probability is over the randomness of $\mathcal{A}$ and GC algorithms.

Theorem A.6 Assuming the existence of secure one-way functions, there exists adaptive secure garbling schemes for polysized Boolean circuits.

A.3 Identity-Based Encryption

An identity-based encryption (IBE) scheme $\text{IBE} = (\text{Setup}, \text{Keygen}, \text{Enc}, \text{Dec})$ for set of identity spaces $\mathcal{I} = \{\{0, 1\}^n\}_{n \in \mathbb{N}}$ and message spaces $\mathcal{M}$ consists of the following algorithms:

$\text{Setup}(1^\lambda, 1^n)$: The setup algorithm takes as input the security parameter λ and identity length n . It outputs the public parameters mpk and the master secret key msk .

$\text{Keygen}(\text{msk}, \text{id})$: The key generation algorithm takes as input the master secret key msk and an identity $\text{id} \in \{0, 1\}^n$. It outputs a secret key sk_{id} .

$\text{Enc}(\text{mpk}, \text{id}, m)$: The encryption algorithm takes as input the public parameters mpk , a message $m \in \mathcal{M}$, and an identity $\text{id} \in \{0, 1\}^n$. It outputs a ciphertext ct .

$\text{Dec}(\text{sk}_{\text{id}}, \text{ct})$: The decryption algorithm takes as input a secret key sk_{id} and a ciphertext ct . It outputs either a message $m \in \mathcal{M}$ or a special symbol $\perp$.

Correctness. We say an IBE scheme $\text{IBE} = (\text{Setup}, \text{Keygen}, \text{Enc}, \text{Dec})$ satisfies correctness if for all $\lambda, n \in \mathbb{N}$, $(\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^\lambda, 1^n)$, $\text{id} \in \{0, 1\}^n$, $m \in \mathcal{M}$, $\text{sk}_{\text{id}} \leftarrow \text{Keygen}(\text{msk}, \text{id})$,

$$\Pr_r[\text{Dec}(\text{sk}_{\text{id}}, \text{ct}) = m \mid \text{ct} = \text{Enc}(\text{mpk}, \text{id}, m; r)] = 1 - \text{negl}(\lambda)$$

where r is sampled uniformly at random.

Definition A.7 We say an IBE scheme $\text{IBE} = (\text{Setup}, \text{Keygen}, \text{Enc}, \text{Dec})$ is adaptively secure if for any stateful PPT adversary $\mathcal{A}$ and polynomial $\text{poly}(\cdot)$ there exists a negligible function $\text{negl}(\cdot)$, such that for all $\lambda \in \mathbb{N}$ and $n = \text{poly}(\lambda)$, the following holds

$$\Pr \left[\mathcal{A}^{\text{Keygen}(\text{msk}, \cdot)}(\text{ct}) = b : \begin{array}{l} 1^n \leftarrow \mathcal{A}(1^\lambda) \\ (\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^\lambda, 1^n); \quad b \leftarrow \{0, 1\} \\ (m_0, m_1, \text{id}^*) \leftarrow \mathcal{A}^{\text{Keygen}(\text{msk}, \cdot)}(\text{mpk}) \\ \text{ct} \leftarrow \text{Enc}(\text{mpk}, \text{id}^*, m_b) \end{array} \right] \leq \frac{1}{2} + \text{negl}(\lambda),$$

where all identities id queried by $\mathcal{A}$ satisfy $\text{id} \neq \text{id}^*$ and the probability is over the randomness of $\mathcal{A}$, Setup , Keygen and Enc algorithms.

A.4 Registration Based Encryption

In this work, we formalize only the restricted variant of registration-based encryption needed for our construction: the batch of identities is hashed all at once, there is no dynamic update interface, we use a keyless interface in which the hash depends only on identities, and we only require non-inclusion security. The keyless interface can be obtained from a standard RBE scheme by generating a single registration key pair during setup and using the same public key for every identity, so that the public-key component need not be written explicitly in the hash input. This interface is closely related to what recent work calls laconic encryption. For this restricted interface, one may view the literature as providing two natural instantiation paths: the original RBE line from CDH-type assumptions [GHMR18, GHM⁺19], and the more recent laconic-encryption line with efficient lattice-based realizations [DKL⁺23].

A registration based encryption (RBE) scheme $\text{RBE} = (\text{Gen}, \text{Hash}, \text{Hint}, \text{Enc}, \text{Dec})$ for a set of identity spaces $\mathcal{I} = \{\{0, 1\}^n\}_{n \in \mathbb{N}}$ and message space $\mathcal{M}$ consists of the following algorithms.

$\text{Gen}(1^\lambda)$: The setup algorithm takes as input the security parameter 1^λ and outputs public parameters pp .

$\text{Hash}(\text{pp}, (\text{id}_1, \dots, \text{id}_\ell))$: The hashing algorithm takes as input the public parameters pp and a tuple of pairwise distinct identities $(\text{id}_1, \dots, \text{id}_\ell)$, where each $\text{id}_i \in \{0, 1\}^n$, and outputs a hash value $\text{h} \in \{0, 1\}^k$, where $k = k(\lambda, n)$ is the fixed output length of the hash for identity space $\{0, 1\}^n$.

$\text{Hint}(\text{pp}, \text{id}_{i^*}, (\text{id}_1, \dots, \text{id}_\ell))$: The hint generation algorithm takes as input the public parameters pp , an identity id_{i^*} contained in the tuple $(\text{id}_1, \dots, \text{id}_\ell)$, and outputs a pair (h, η) .

$\text{Enc}(\text{pp}, \text{h}, \text{id}, m)$: The encryption algorithm takes as input the public parameters pp , a hash value h , an identity $\text{id} \in \{0, 1\}^n$, and a message $m \in \mathcal{M}$. It outputs a ciphertext ct .

$\text{Dec}(\text{pp}, \eta, \text{ct})$: The decryption algorithm takes as input the public parameters pp , a hint η , and a ciphertext ct . It outputs either a message $m \in \mathcal{M}$ or $\perp$.

Correctness. We say an RBE scheme $\text{RBE} = (\text{Gen}, \text{Hash}, \text{Hint}, \text{Enc}, \text{Dec})$ satisfies correctness if there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda, n \in \mathbb{N}$, all tuples of pairwise distinct identities $(\text{id}_1, \dots, \text{id}_\ell)$ with $\text{id}_i \in \{0, 1\}^n$, all indices $i^* \in [\ell]$, and all messages $m \in \mathcal{M}$, if

$$\begin{aligned} \text{pp} &\leftarrow \text{Gen}(1^\lambda), & \text{h} &\leftarrow \text{Hash}(\text{pp}, (\text{id}_1, \dots, \text{id}_\ell)), \\ (\text{h}', \eta) &\leftarrow \text{Hint}(\text{pp}, \text{id}_{i^*}, (\text{id}_1, \dots, \text{id}_\ell)), & \text{ct} &\leftarrow \text{Enc}(\text{pp}, \text{h}, \text{id}_{i^*}, m), \end{aligned}$$

then

$$\Pr[\text{h}' = \text{h} \text{ and } \text{Dec}(\text{pp}, \eta, \text{ct}) = m] \geq 1 - \text{negl}(\lambda),$$

where the probability is over the randomness of Gen , Hint , and Enc .

Definition A.8 We say an RBE scheme $\text{RBE} = (\text{Gen}, \text{Hash}, \text{Hint}, \text{Enc}, \text{Dec})$ is non-inclusion secure if for any stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and $n = \text{poly}(\lambda)$, the following holds in the non-inclusion setting, where the challenge identity id^* does not belong to the set of identities hashed into h :

$$\Pr \left[\mathcal{A}(\text{ct}^*) = b \mid \begin{array}{l} \text{pp} \leftarrow \text{Gen}(1^\lambda); \quad b \leftarrow \{0, 1\} \\ (m_0, m_1, \text{id}_1, \dots, \text{id}_\ell, \text{id}^*) \leftarrow \mathcal{A}(\text{pp}) \\ \text{such that } \text{id}_1, \dots, \text{id}_\ell \text{ are pairwise distinct and} \\ \text{id}^* \notin \{\text{id}_1, \dots, \text{id}_\ell\} \\ \text{h} \leftarrow \text{Hash}(\text{pp}, (\text{id}_1, \dots, \text{id}_\ell)) \\ \text{ct}^* \leftarrow \text{Enc}(\text{pp}, \text{h}, \text{id}^*, m_b) \end{array} \right] \leq \frac{1}{2} + \text{negl}(\lambda),$$

where all identities belong to $\{0, 1\}^n$, and the probability is over the randomness of $\mathcal{A}$, Gen , and Enc .

B From Tagged to Ordinary Batch Decryption

The CHK transformation uses a fresh signature verification key as the encryption tag and signs the resulting ciphertext [CHK04]. Kiltz shows that selective-tag security suffices for this transformation [Kil06, Sections 3.2 and 4.1]. We give its batch-release version below. The additional obligation is to preserve the actual batch hint and the complete admission history; single-ciphertext tag-based security alone does not imply that obligation.

Let TBD be a tagged batch-decryption scheme as in Section 2.2, and let $\text{OTS} = (\text{Keygen}, \text{Sign}, \text{Ver})$ have perfect correctness and strong one-time unforgeability. Its verification keys are valid tags. The verification-key and signature lengths $v(\lambda)$ and $s(\lambda)$ are fixed polynomials, independent of the message and batch size; the signing domain contains all polynomial-length tuples below. In the strong unforgeability experiment, the forger receives a fresh verification key and may make at most one signing query. It succeeds by outputting a valid message–signature pair different from every pair returned by that query. In particular, a valid pair before any signing query is a forgery. Write $\text{Adv}_{\text{SUF}}(\mathcal{F})$ for its success probability. Security of both primitives is required in the same allowed oracle environment, with all component and simulation work included. The allowed ideal oracles

are fixed, independently sampled random functions shared by all parties, as in Section 3.1; private queries reveal no timing, query-count or cache information. The theorem also applies in the standard model. It makes no claim for mutable or arbitrarily correlated auxiliary oracle environments.

For public decryption, we use the explicit-public-parameter form $D = (\mathbf{pk}, D_0)$: admission checks the embedded key against $\mathbf{pk}$. Section 3.2 already has this form. For a general tagged scheme one can attach this copy and strip it before calling the original algorithms, at a charged additional cost $O(|\mathbf{pk}|)$ per ciphertext. This public, injective change preserves both games and correctness. In what follows D includes any such necessary copy. All tuples have their ordinary unambiguous representations. Fix a signature domain constant d_{sig} .

B.1 The Compiler

For a batch $\mathcal{B} = (\text{CT}_1, \dots, \text{CT}_B)$ with $\text{CT}_i = (\mathbf{vk}_i, D_i, \text{sig}_i)$, write

$$\pi(\mathcal{B}) = ((\mathbf{vk}_1, D_1), \dots, (\mathbf{vk}_B, D_B)).$$

The public admission predicate $\text{Adm}_c(\mathbf{pk}, \mathcal{B})$ checks the complete tuple syntax, verification-key and signature lengths, consistent public-key copies, and every signature

$$\text{OTS.Ver}(\mathbf{vk}_i, (d_{\text{sig}}, \mathbf{pk}, \mathbf{vk}_i, D_i), \text{sig}_i) = 1.$$

It also checks $\text{TBD.Adm}(\mathbf{pk}, \pi(\mathcal{B}))$. It accepts exactly when all these public checks pass. The empty batch is admitted for a well-formed key. No check requires distinct verification keys; all underlying batch checks remain in force. This predicate is a convention for the following four algorithms.

$\text{Setup}(1^\lambda)$: Return $\text{TBD.Setup}(1^\lambda)$.

$\text{Enc}(\mathbf{pk}, m)$:

- (1) Reject malformed keys or messages with $\perp$. Generate

$$(\mathbf{vk}, \text{ssk}) \leftarrow \text{OTS.Keygen}(1^\lambda).$$

- (2) Compute $D \leftarrow \text{TBD.Enc}(\mathbf{pk}, \mathbf{vk}, m)$. If this returns $\perp$, return $\perp$.
- (3) Compute

$$\text{sig} \leftarrow \text{OTS.Sign}(\text{ssk}, (d_{\text{sig}}, \mathbf{pk}, \mathbf{vk}, D))$$

and return $\text{CT} = (\mathbf{vk}, D, \text{sig})$.

$\text{PartDec}(\text{sk}, \mathcal{B})$:

- (1) Obtain the associated $\mathbf{pk}$ from the key representation and complete $\text{Adm}_c(\mathbf{pk}, \mathcal{B})$. Return $\perp$ on rejection, and empty for an admitted empty batch.
- (2) Return $\text{TBD.PartDec}(\text{sk}, \pi(\mathcal{B}))$ unchanged.

$\text{Dec}(T, i, \mathcal{B})$:

- (1) Reject an empty or malformed batch or an out-of-range index with $\perp$. Read $\mathbf{pk}$ from D_1 and complete $\text{Adm}_c(\mathbf{pk}, \mathcal{B})$; return $\perp$ on rejection.

(2) Return $\text{TBD.Dec}(T, i, \pi(\mathcal{B}))$.

The associated public key can be retained as part of the underlying secret-key representation, as it is in our core; if this normalization is needed, its $O(|\text{pk}|)$ storage is charged once. Relative to this explicit representation setup changes neither key. All outer signatures and underlying public checks precede any private validation. The whole projection is passed in order, with no filtering, scalar decryption calls, or separate coordinate hints. In particular, an underlying private failure rejects the whole batch. The public decoder obtains its projection from its actual public input and keeps the same index.

Proposition B.1 (Correctness and sizes of the compiler) *The compiler preserves adaptive honest-history correctness, including repeated verification keys and coincident outputs of distinct encryption invocations. Relative to the explicit-public-parameter form above, keys and hint lengths are unchanged, and ciphertext overhead is $O(v + s)$ beyond the underlying tagged ciphertext, including the outer tag copy. Public validation adds B signature verifications.*

proof An adaptive history of compiled encryptions induces an adaptive history of honest tagged encryptions: generate each signature key, choose its verification key as the tag, invoke the underlying honest encryption, and sign its output. This is a permitted history in the underlying correctness definition, including all public oracle computations and repeated tags. Each selected set of distinct compiled invocation indices projects to the same distinct underlying invocation indices. Perfect signature correctness ensures that every added public check passes on an honestly generated ciphertext. Therefore every successful underlying selected batch yields exactly the same hint and recovered messages in the compiled scheme. Overlap and reuse preserve the original projections and indices.

This argument does not condition on distinct output values. Even if two underlying outputs or two complete compiled outputs coincide, the selected occurrences remain those of different honest invocations. If their projected batch is rejected, that is an underlying honest correctness failure and is charged to its error bound. No signature-key collision probability or extra distinct-key assumption is used. The size claims follow by counting the displayed tuples and unchanged response.

B.2 Security with Complete Batch Releases

Theorem B.2 (CHK compiler for batch decryption) *For every uniform PPT adversary $\mathcal{A}$ against the compiled ordinary scheme there are a selective-tag adversary $\mathcal{B}$ against TBD and strong one-time signature forgers $\mathcal{F}_0, \mathcal{F}_1$, each uniform PPT, such that*

$$\Delta_{\text{CT}}(\mathcal{A}) \leq \Delta_{\text{sTag}}(\mathcal{B}) + \sum_{b \in \{0,1\}} \text{Adv}_{\text{SUF}}(\mathcal{F}_b).$$

Thus selective-tag privacy suffices for ordinary exact-ciphertext-exclusion privacy. No entropy or uniqueness condition on signature verification keys is required.

proof Fix $b \in \{0, 1\}$ and let $p_b^j = \Pr[J_b^j = 1]$ for the following hybrids. Probabilities include all terminal-zero branches.

Hybrid J_b^0 : This is the compiled exact-ciphertext-exclusion experiment of Definition 2.1. We equivalently generate the challenge signature key pair before setup and keep it private until the challenge, writing its verification key as vk^* . The rest of challenge encryption occurs at the adaptive challenge time. For primitives using fixed shared random oracles, condition on those functions: the independent signature-key tape can be moved earlier without changing any output law. Private oracle evaluations reveal no cache or timing information. The same reasoning applies without oracles.

Hybrid J_b^1 : This hybrid adds termination with output zero at the first publicly admitted prechallenge batch containing an entry with verification key vk^* . After the challenge it also terminates on an admitted batch with that verification key. The original exact-ciphertext test is performed first on a postchallenge admitted batch; if it fires, both hybrids already terminate. Every admitted batch is recorded in full before these tests and before any private validation.

The following two lemmas show

$$|p_b^0 - p_b^1| \leq \text{Adv}_{\text{SUF}}(\mathcal{F}_b), \quad p_b^1 = \Pr[\text{sTag}_b^B = 1].$$

Applying the triangle inequality to the two endpoints proves the theorem. The reductions preserve all malformed-input, empty-batch, no-challenge and whole-batch rejection conventions. Their polynomial envelopes include the signature algorithms, public checks, source queries, oracle calls and all local simulation work.

Lemma B.3 *For each $b \in \{0, 1\}$, there is a forger $\mathcal{F}_b$ making at most one signing query such that $|p_b^0 - p_b^1| \leq \text{Adv}_{\text{SUF}}(\mathcal{F}_b)$.*

proof Given its signature challenger's key vk^* , the forger runs the actual underlying setup, retains the secret key, and simulates J_b^0 with genuine whole-batch releases and shared oracles. It hides vk^* from $\mathcal{A}$ until the challenge.

At the first admitted prechallenge batch containing vk^* , it outputs the valid pair (M, sig) of any such entry, where $M = (d_{\text{sig}}, \text{pk}, vk^*, D)$. It has made zero signing queries. This applies even if the batch would later fail private validation, or if the entry would not be reached. It also covers a coincident verification key generated by an adversarial public encryption invocation: such an entry is still a valid zero-query forgery under the supplied key.

If no such prechallenge event occurs, the forger generates the underlying challenge $D^* \leftarrow \text{TBD.Enc}(\text{pk}, vk^*, m_b)$ at the requested time. It asks its single signing query on $M^* = (d_{\text{sig}}, \text{pk}, vk^*, D^*)$, obtains sig^* , and forms $\text{CT}^* = (vk^*, D^*, \text{sig}^*)$. The retrospective exact-ciphertext test cannot succeed on this continuing branch: any prior exact occurrence would already have had key vk^* and triggered the prechallenge event.

For a later admitted batch, an exact occurrence of CT^* terminates both hybrids and needs no forgery charge. Otherwise, if the batch contains $\text{CT} = (vk^*, D, \text{sig})$, the forger outputs its pair. Complete tuple binding and $\text{CT} \neq \text{CT}^*$ give $(M, \text{sig}) \neq (M^*, \text{sig}^*)$, including a different signature on the same signed message. Admission ensures verification. Coupling the two hybrids to their first additional termination shows that every possible difference is covered by one of these forgeries. The forger may stop on additional branches where both final outputs would have been zero, which only increases its success probability.

Lemma B.4 *There is one uniform PPT selective-tag adversary $\mathcal{B}$ such that $p_b^1 = \Pr[\text{sTag}_b^{\mathcal{B}} = 1]$ for both bits b .*

proof Before underlying setup, $\mathcal{B}$ generates $(\text{vk}^*, \text{ssk}^*)$ and commits to the tag vk^* . It gives the returned public key to $\mathcal{A}$ but withholds the reserved signature key. For each release query it completes all compiled public checks locally. It returns $\perp$ on rejection and the specified empty response for an admitted empty batch. Otherwise it records every entry before processing the guards. If any entry has key vk^* , it terminates with output zero, without requesting a source release. On a prechallenge branch it makes no challenge request, so the source's no-challenge convention gives zero as well. On every other branch it forwards the entire ordered projection $\pi(\mathcal{B})$ as one source release query and returns its hint or rejection unchanged.

At a well-formed challenge it forwards $\mathcal{A}$'s two messages and signs the returned D^* using ssk^* . It then returns $(\text{vk}^*, D^*, \text{sig}^*)$. Every earlier forwarded entry has a tag different from vk^* , so both the source tag test and the target retrospective exact-ciphertext test pass. Malformed or absent challenges terminate with the prescribed zero output.

The local admission record is exactly the target's full record. On continuing branches the source record is its entrywise projection, including every entry of a batch whose private checks failed or were not reached. A terminal guarded batch need not be sent to the source: its output is already fixed to zero in both experiments. After the challenge, exact and nonexact ciphertexts under vk^* both give zero in J_b^1 , so the local tag guard has exactly the required endpoint law. All allowed batches are forwarded whole; overlapping releases and old hints therefore retain their complete joint distribution. Public checks may be repeated by the source, with their work charged and their fixed-oracle answers unchanged. Finally $\mathcal{B}$ outputs $\mathcal{A}$'s bit. This simulation uses neither the hidden challenge bit nor the underlying secret key.

B.3 A Concrete One-Time Signature Instantiation

For example, a domain-separated hash-then-Lamport one-time signature has $v = 2\lambda^2$ and $s = \lambda^2$: sample secret blocks $z_{j,b} \leftarrow_{\$} \{0, 1\}^\lambda$, publish their indexed hash values, hash the signed tuple to a λ -bit digest, and reveal one block per digest bit. Commitment and message hashing use separate domains of H_0 and H_1 . Strong one-time security is required with all component oracle queries included; an alternative valid signature on the same digest is also a forgery.

C Proofs and Sampling for the Epochless Construction

This appendix gives the concrete RSA sampling procedures and the full correctness and security proofs for the construction in Section 3.

C.1 RSA Setup and Sampling

Recall that $G = \mathbb{Z}_N^\times$, where $N = pq$ for distinct odd primes, and the trapdoor is $\varphi(N) = (p-1)(q-1)$. The public descriptor contains N and the exponent-sampling parameters. For the admitted primes $N/2 < e < N$, $\gcd(e, \varphi(N)) = 1$ and

$$\text{Root}(\text{pp}, \varphi(N), e, C) = C^{e^{-1} \bmod \varphi(N)} \bmod N.$$

Sampling a unit requires no factorization: test a candidate integer x by $1 \leq x < N$ and $\gcd(x, N) = 1$.

The RSA setup uses $(\lambda + 2)$ -bit primes p, q and polynomially bounded first-success sampling. It rejects $q = p$ and samples a public prime $e_0 \in (N/2, N)$. To sample a prime in this interval, test independent uniform $\lceil \log_2 N \rceil$ -bit strings, accepting the first whose integer value is prime and lies in $(N/2, N)$, with a sufficiently large polynomial bound on the number of trials. Thus exponents have at most $2\lambda + 4$ bits. If setup sampling fails, use the complete fallback $(N, \varphi(N), e_0) = (15, 8, 11)$. Encryption uses the same prime sampler with fallback e_0 , and a unit sample falls back to 1. The fallback probabilities and unit sampling distance are negligible; they are included rather than conditioned away. The setup-averaged exponent bound includes the negligible probability of the complete setup fallback. These are asymptotic length choices, not a claim about practical RSA security at a modulus of $2\lambda + 4$ bits.

C.2 Proof of Proposition 3.1

proof The unique-root property ensures that private extraction returns the sender's x_i . Thus every honest root check succeeds. Distinct prime exponents imply $\beta_i \equiv 1 \pmod{e_i}$ and $\beta_i \equiv 0 \pmod{e_j}$ for $j \neq i$. Substituting $C_i = x_i^{e_i}$ and $W = \prod_j x_j$ into (1), commutativity gives

$$\hat{x}_i = \left(\prod_j x_j \right)^{\beta_i} x_i^{1-\beta_i} \prod_{j \neq i} x_j^{-\beta_i} = x_i.$$

Hence the same root, derived key, and message pad are recovered. This identity also applies to the unique roots of adversarially formed group elements in any privately accepted batch. We assign no intended plaintext to an arbitrary malicious ciphertext, and a malicious neighbor may cause the whole batch to be rejected.

For a fixed public descriptor, put $m(\mathbf{pp}) = \max_{e \in \mathcal{P}_{\mathbf{pp}}} \Pr[\text{Sample}_e(\mathbf{pp}) = e]$. Before each honest exponent is sampled, condition on the full prior history and the public descriptor. The new independent sample equals any specified earlier exponent with probability at most $m(\mathbf{pp})$. A union bound over the at most $\binom{L}{2}$ pairs, followed by averaging over setup using $\mathbb{E}_{\mathbf{pp}}[m(\mathbf{pp})] \leq \mu(\lambda)$, therefore bounds every honest exponent collision in the history. Outside this single event, all subsequently selected honest batches with distinct invocation indices are correct simultaneously, regardless of repeated tags. Coincident ciphertext values from different invocations imply an exponent collision and are included in the bad event; their rejection is not conditioned away. Reusing the same invocation in later batches is permitted, while selecting it twice within one batch is outside the correctness convention. Root-sampling bias causes no correctness error because roots always lie in G . No conditioning on successful admission is used.

C.3 Proof of Theorem 3.2

proof Fix a uniform PPT adversary $\mathcal{A}$ and a challenge bit $b \in \{0, 1\}$. We first describe hybrids $G_b^0, \dots, G_b^4$ and then prove the transition bounds in Lemmas C.1–C.4. Write $p_b^j = \Pr[G_b^j = 1]$. Every probability is unconditional, including all failed queries and terminal-output-zero branches.

Hybrid G_b^0 : This is the adaptive-tag experiment of Definition 2.2, instantiated with the algorithms of Section 3.2. Its phases are as follows.

- *Initialization Phase:* The challenger computes $(\text{pk}, \text{sk}) \leftarrow \text{Setup}(1^\lambda)$, gives pk to $\mathcal{A}$, and initializes an empty admission record. All parties use the same independent random oracles H_0, H_1 throughout.
- *Pre-Challenge Query Phase:* For each adaptive batch query $\mathcal{B}$, the challenger completes $\text{Adm}(\text{pk}, \mathcal{B})$. It returns $\perp$ if admission fails, and **empty** for an admitted empty batch. Otherwise, it records the entire batch and all its tags before returning $\text{PartDec}(\text{sk}, \mathcal{B})$. The record retains every entry, including entries whose private checks fail or are not reached.
- *Challenge Phase:* At an adaptively chosen point, $\mathcal{A}$ submits (t^*, m_0, m_1) with $t^* \in \{0, 1\}^*$ and $m_0, m_1 \in \{0, 1\}^\lambda$. If t^* occurs in the earlier admission record, the challenger outputs zero and terminates. Otherwise it generates $D^* \leftarrow \text{Enc}(\text{pk}, t^*, m_b)$ with fresh encryption randomness and sends D^* to $\mathcal{A}$.
- *Post-Challenge Query Phase:* The challenger completes the same public admission and empty-batch checks. It records every admitted nonempty batch and all its tags. If that batch contains any pair with tag t^* , it outputs zero and terminates before private validation; otherwise, it returns $\text{PartDec}(\text{sk}, \mathcal{B})$. Other-tag queries may copy or modify the challenge's group components.
- *Response Phase:* The experiment outputs $\mathcal{A}$'s final bit. Malformed challenges and executions without a challenge have output zero.

Hybrid G_b^1 : This hybrid proceeds as G_b^0 , except that the challenge root x^* is sampled uniformly from G , instead of using Sample_G . Every other sampling law, including setup and the fresh exponent sample, remains unchanged.

Hybrid G_b^2 : This hybrid proceeds as G_b^1 , except that a reached private root check is rejected whenever the correct root has not appeared in an ordinary check-domain query for that same public tuple (pk, t, e, C) . Such a rejection rejects the whole batch, even if the submitted ρ equals the hidden check value. If the root has appeared at that check coordinate, validation proceeds normally. All evaluations refer to the same fixed random-oracle coordinates.

Hybrid G_b^3 : This hybrid proceeds as G_b^2 , except that the key used to encrypt the challenge message is an independent uniform $K^* \in \{0, 1\}^\lambda$. The random oracle H_{key} is unchanged; its designated challenge value is no longer used to form the message component.

Hybrid G_b^4 : This hybrid proceeds as G_b^3 , except that the challenge message component is $c^* = m_b \oplus P^*$ for an independent uniform $P^* \in \{0, 1\}^\lambda$. The key K^* is now unused, and the designated private pad evaluation may be omitted. Ordinary oracle queries retain their usual answers.

All hybrids retain complete public admission before private processing, the full admission record, whole-batch rejection, and the empty, malformed, and absent-challenge conventions of G_b^0 . Their modifications are changes to the experiment, not assumptions about which ciphertexts the adversary can construct.

By Lemmas C.1–C.4, for each b ,

$$|p_b^0 - p_b^4| \leq \epsilon_G(\lambda) + \text{Adv}_{\text{root}}(\mathcal{R}_b) + (q_{\text{val}} + q_{\text{pad}})2^{-\lambda}.$$

The final hybrids G_0^4 and G_1^4 have identical distributions. Couple them using the same uniform message component c^* , random functions, and all other tapes. Their prechallenge views and hence their chosen tag and messages agree. Every subsequent operation depends on the public component c^* rather than on m_b , so admission, release responses, and all terminal decisions agree as well. Thus $p_0^4 = p_1^4$, and the triangle inequality gives (2).

All simulations retain the full publicly admitted batch, including coordinates not reached after a private failure. Overlapping batches and old hints need no special treatment: the same oracle tables and the same recorded roots are used throughout. Duplicate exponents are rejected by the actual public test; privacy does not condition on a collision-free exponent event. Repeated nonchallenge tags are allowed; the table is indexed by the full tuple (pk, t, e, C) , not by tag alone. Table maintenance, root recognition, and integer group operations take polynomial time, and all reached private and ordinary queries contribute to the resource bounds. Under the stated assumptions every term is negligible.

For selective-tag privacy, an adaptive-tag adversary runs the selective adversary's commitment stage before setup and retains its tag. It can enforce the selective experiment's early forbidden-tag termination locally, making no challenge request on such a branch. On continuing branches it supplies the retained tag at the challenge. The endpoint output laws, including every terminal-zero branch, agree exactly.

Lemma C.1 *For each $b \in \{0, 1\}$,*

$$|p_b^0 - p_b^1| \leq \epsilon_G(\lambda).$$

proof Conditional on every supported public descriptor and challenge prefix, the distributions of $\text{Sample}_G(\text{pp})$ and a uniform element of G have distance at most $\epsilon_G(\lambda)$. Applying the same subsequent randomized computation, including its terminal decisions, cannot increase this distance. Hence $|p_b^0 - p_b^1| \leq \epsilon_G(\lambda)$. No other sampling law is changed. In particular, setup fallbacks remain in the strong RSA assumption, and e^* retains its actual fresh distribution. Since powering by e^* permutes G , the component $C^* = (x^*)^{e^*}$ is now uniform in G . This is an analytical change of distribution; it does not require an efficient exact uniform sampler.

Lemma C.2 *For each $b \in \{0, 1\}$,*

$$|p_b^1 - p_b^2| \leq q_{\text{val}} 2^{-\lambda}.$$

Moreover, releases in G_b^2 can be computed from ordinary check queries without the root-extraction trapdoor.

proof The difficulty is to answer release queries without the trapdoor, including queries containing copied or modified challenge components. For every valid public tuple $y = (pk, t, e, C)$, let x_y be its unique root and write

$$V_y = H_{\text{check}}(pk, t, e, C, x_y).$$

These are distinct random-oracle coordinates for distinct y , even if some roots coincide. We may equivalently index these coordinates by y : an ordinary check query supplies a candidate z , and the public test $z \in G$ and $z^e = C$ recognizes whether it accesses V_y . All other queries, including malformed or wrong-root queries, retain their ordinary cached random-oracle answers. A private validation reads the same V_y ; relabeling a coordinate does not reveal the root or add an oracle interface.

The forced rejection rule in G_b^2 can be implemented by maintaining an ordinary check-query table. For each reached tuple y , search this table for a candidate $z \in G$ with $z^e = C$. Root uniqueness identifies the correct root, and the cached check answer determines whether validation succeeds. If no such query exists, reject the entire batch. If every coordinate passes, return the product of the recorded roots. This uses no trapdoor and retains all public admission and recording steps.

To bound this change, fix y and consider the distinct guesses submitted at reached private checks for V_y in the modified experiment, stopping before the first ordinary query that supplies x_y at this check coordinate. Denote this set by S_y . Coordinates not reached after an earlier failure remain in the admission record but contribute no guesses. Fix the setup, tapes, and all oracle coordinates other than V_y . Before that exposure, the modified private replies do not depend on V_y , and therefore neither does the stopped set S_y . More formally, in the product probability space obtained by deleting coordinate V_y ,

$$\Pr[V_y \in S_y \mid \text{all other coordinates and tapes}] = |S_y| 2^{-\lambda}.$$

The challenge's published check requires one qualification. If y is the challenge tuple, an earlier admitted occurrence of its tag makes the experiment terminate before the challenge is returned; otherwise there is no earlier private guess at that tag and no later legal one. Thus the displayed ρ^* never supplies an uncharged value for a continuing guess sequence. The tag-separation rules above are essential for this argument.

The first observable difference between G_b^1 and G_b^2 must be a matching guess in one of these stopped sets. Summing over coordinates and using $\sum_y |S_y| \leq q_{\text{val}}$ gives

$$|p_b^1 - p_b^2| \leq q_{\text{val}} 2^{-\lambda}.$$

This is an unconditional bound on a first difference. After several real failures a new guess need not have conditional success probability $2^{-\lambda}$; repeated failures must not be treated as independent trials. Repeated private reads and later ordinary exposure always reuse the same oracle value.

Lemma C.3 *For each $b \in \{0, 1\}$, there is a uniform PPT root-extraction algorithm $\mathcal{R}_b$ such that*

$$|p_b^2 - p_b^3| \leq \text{Adv}_{\text{root}}(\mathcal{R}_b).$$

proof The reduction receives (pp, U) with U uniform in G and does not know the trapdoor. It privately reserves the fresh exponent e^* before running $\mathcal{A}$, using the actual exponent sampler, and sets $C^* = U$ at the adaptive challenge time. Reserving these independent coins does not reserve or guess the challenge tag. The reduction runs $\mathcal{A}$ and the public algorithms with one common pair of locally sampled random oracles. Group generation and operations are oracle independent, so this has the required joint distribution; no externally fixed oracle is being programmed.

Private validations are simulated as in G_b^2 : unknown roots cause rejection, and accepted batches yield products of recorded roots. The challenge check is obtained from the coordinate $V_{(\text{pk}, t^*, e^*, U)}$. The reduction samples K^* and forms the masked message, and thus simulates G_b^3 without knowing a root of U . The now-unused private evaluation of the original $H_{\text{key}}(\text{pk}, X^*, x^*)$ need not be materialized: its value remains a lazily sampled oracle coordinate, and its private cache entry has no observable effect. This does not omit its cost from the source query and running-time bounds. It monitors every ordinary check- or key-domain query, both before and after the challenge. Whenever a query supplies $z \in G$ with $z^{e^*} = U$, it outputs (z, e^*) and stops.

To justify the comparison, couple the two hybrids by making the latent value $H_{\text{key}}(\text{pk}, X^*, x^*)$ in G_b^2 equal the independent key used in G_b^3 . Until an ordinary query accesses this key coordinate, their observable transcripts have identical laws. An access before the challenge is also covered: it already supplies the root at an ordinary query monitored by $\mathcal{R}_b$. On a history with no such earlier query, the coordinate is unexposed when the challenge is formed. Its value is fresh, even though X^* contains an adaptively chosen tag and a check value from the independent check domain. Every first access supplies x^* and hence solves the strong RSA instance. Monitoring additional candidate roots can only increase the reduction's success probability. This is a coupling up to a first query, not a claim of uniformity after conditioning on no root ever being found.

Lemma C.4 *For each $b \in \{0, 1\}$,*

$$|p_b^3 - p_b^4| \leq q_{\text{pad}} 2^{-\lambda}.$$

proof The key K^* from G_b^3 is unused in G_b^4 . In this last experiment the entire visible execution, including its random-oracle answers, is independent of K^* . The list of keys proposed in ordinary queries at addresses (pk, t^*, X^*, K) in the pad domain therefore contains K^* with probability at most $q_{\text{pad}} 2^{-\lambda}$. The list includes queries made before the challenge as well as those chosen adaptively afterwards. Coupling the pad coordinate to P^* until its first such access gives

$$|p_b^3 - p_b^4| \leq q_{\text{pad}} 2^{-\lambda}.$$

The challenge encryption's own private pad evaluation is not a guess and is not included in this list.

D Additional Definitions and Proofs for the Epoch-Based Construction

D.1 Receiver Non-Committing Identity-Based Encryption

We define a weaker variant of receiver non-committing identity-based encryption (RNC-IBE) where the adversary receives only the user key for the challenge identity rather than the master secret key. A weak RNC-IBE scheme for a set of identity spaces $\mathcal{I} = \{\{0, 1\}^n\}_{n \in \mathbb{N}}$ and message spaces $\mathcal{M}$ consists of the following algorithms.

Setup $(1^\lambda, 1^n)$: The setup algorithm takes as input the security parameter 1^λ and the length of the identities 1^n . It outputs the master public key mpk and master secret key msk .

Keygen (msk, id) : The key generation algorithm takes as input a master secret key msk and an identity id and outputs a secret key sk_{id} .

Enc $(\text{mpk}, \text{id}, m)$: The encryption algorithm takes as input a master public key mpk , an identity id and a message m and outputs a ciphertext ct .

Dec (sk, ct) : The decryption algorithm takes as input a secret key sk and a ciphertext ct and outputs either a message m or $\perp$.

Sim $_1(1^\lambda, 1^n)$: The first simulator takes as input the security parameter 1^λ and the length of the identities 1^n and outputs a master public key mpk and a state st .

$\text{Sim}_2(\text{st}, \text{id})$: The second simulator takes as input a state st and an identity id and outputs a secret key sk_{id} and an updated st .

$\text{Sim}_3(\text{st}, \text{id}^*)$: The third simulator takes as input a state st and an identity id^* and outputs a ciphertext ct^* and an updated state st .

$\text{Sim}_4(\text{st}, m^*)$: The fourth simulator takes as input a state st and a message m^* and outputs the secret key sk_{id^*} .

Correctness. For correctness, we require that there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}, n \in \mathbb{N}$, any identity $\text{id} \in \{0, 1\}^n$ and message m ,

$$\Pr \left[\text{Dec}(\text{sk}_{\text{id}}, \text{ct}) = m \mid \begin{array}{l} (\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^\lambda, 1^n) \\ \text{ct} \leftarrow \text{Enc}(\text{mpk}, \text{id}, m) \\ \text{sk}_{\text{id}} \leftarrow \text{Keygen}(\text{msk}, \text{id}) \end{array} \right] \geq 1 - \text{negl}(\lambda)$$

where the probability is over the randomness of Setup , Keygen and Enc algorithms.

Security. Consider the following two experiments with an adversary $\mathcal{A}$.

- **Real World:**

- Initialization Phase: $\mathcal{A}$ on input 1^λ , outputs 1^n . The challenger computes $(\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^\lambda, 1^n)$ and sends mpk to $\mathcal{A}$.
- Pre-Challenge Query Phase: In this phase, $\mathcal{A}$ is allowed to make multiple queries id . For each id , the challenger sends $\text{sk}_{\text{id}} \leftarrow \text{Keygen}(\text{msk}, \text{id})$ to $\mathcal{A}$.
- Challenge Phase: The adversary $\mathcal{A}$ sends (m^*, id^*) to the challenger. If id^* was queried in the pre-challenge query phase, the challenger aborts. Otherwise, it computes $\text{ct}^* \leftarrow \text{Enc}(\text{mpk}, \text{id}^*, m^*)$ and sends it to $\mathcal{A}$.
- Post-Challenge Query Phase: In this phase, $\mathcal{A}$ is allowed to make multiple queries id . For each id , if $\text{id} = \text{id}^*$, the challenger aborts. Otherwise, it sends $\text{sk}_{\text{id}} \leftarrow \text{Keygen}(\text{msk}, \text{id})$ to $\mathcal{A}$.
- Challenge-Key Reveal Phase: The challenger computes $\text{sk}_{\text{id}^*} \leftarrow \text{Keygen}(\text{msk}, \text{id}^*)$ and sends it to $\mathcal{A}$.
- Final Query Phase: In this phase, $\mathcal{A}$ is again allowed to make multiple queries id . For each id , if $\text{id} = \text{id}^*$, the challenger aborts. Otherwise, it sends $\text{sk}_{\text{id}} \leftarrow \text{Keygen}(\text{msk}, \text{id})$ to $\mathcal{A}$.
- Response Phase: $\mathcal{A}$ outputs b .

- **Simulated World:**

- Initialization Phase: $\mathcal{A}$ on input 1^λ , outputs 1^n . The challenger computes $(\text{mpk}, \text{st}) \leftarrow \text{Sim}_1(1^\lambda, 1^n)$ and sends mpk to $\mathcal{A}$.
- Pre-Challenge Query Phase: In this phase, $\mathcal{A}$ is allowed to make multiple queries id . For each id , the challenger generates $(\text{sk}_{\text{id}}, \text{st}) \leftarrow \text{Sim}_2(\text{st}, \text{id})$ and sends sk_{id} to $\mathcal{A}$.

- Challenge Phase: The adversary $\mathcal{A}$ sends (m^*, id^*) to the challenger. If id^* was queried in the pre-challenge query phase, the challenger aborts. Otherwise, it computes $(\text{ct}^*, \text{st}) \leftarrow \text{Sim}_3(\text{st}, \text{id}^*)$ and sends ct^* to $\mathcal{A}$.
- Post-Challenge Query Phase: In this phase, $\mathcal{A}$ is allowed to make multiple queries id . For each id , if $\text{id} = \text{id}^*$, the challenger aborts. Otherwise, it generates $(\text{sk}_{\text{id}}, \text{st}) \leftarrow \text{Sim}_2(\text{st}, \text{id})$ and sends sk_{id} to $\mathcal{A}$.
- Challenge-Key Reveal Phase: The challenger computes $\text{sk}_{\text{id}^*} \leftarrow \text{Sim}_4(\text{st}, m^*)$ and sends it to $\mathcal{A}$.
- Final Query Phase: In this phase, $\mathcal{A}$ is again allowed to make multiple queries id . For each id , if $\text{id} = \text{id}^*$, the challenger aborts. Otherwise, it generates $(\text{sk}_{\text{id}}, \text{st}) \leftarrow \text{Sim}_2(\text{st}, \text{id})$ and sends sk_{id} to $\mathcal{A}$.
- Response Phase: $\mathcal{A}$ outputs b .

It is important to note that in the challenge phase, the adversary obtains the challenge ciphertext and later obtains the secret key for the challenge identity, and not the randomness used by the Setup, Keygen or Enc algorithms in the experiment. After obtaining the challenge secret key, the adversary may continue to query secret keys for identities $\text{id} \neq \text{id}^*$. Let p_{real} and p_{sim} be the probabilities with which $\mathcal{A}$ outputs 1 in the real world and simulated world, respectively.

Definition D.1 *A weak RNC-IBE scheme is said to be secure if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that, for all $\lambda \in \mathbb{N}$,*

$$|p_{\text{real}} - p_{\text{sim}}| \leq \text{negl}(\lambda).$$

D.2 Constructing weak RNC-IBE from IBE

The construction uses two independently keyed IBE instances for each message bit. Honest encryption places the message bit in both branches. The simulator instead encrypts 0 in one branch and 1 in the other, and later reveals the key for the branch matching the desired message. Pseudorandom functions make the branch choices and key-generation randomness consistent across queries for the same identity. The detailed security argument is given in Appendix D.3.

We present a generic construction of weak RNC-IBE for the identity spaces $\mathcal{I} = \{\{0, 1\}^n\}_{n \in \mathbb{N}}$ and message spaces $\mathcal{M}_\ell = \{0, 1\}^\ell$, using the following primitives.

- $\text{IBE} = (\text{IBE.Setup}, \text{IBE.Keygen}, \text{IBE.Enc}, \text{IBE.Dec})$: an identity-based encryption scheme for the identity spaces $\{\{0, 1\}^n\}_{n \in \mathbb{N}}$ and one-bit message space $\{0, 1\}$.
- $\text{PRF} = (\text{PRF.Setup}, \text{PRF.Eval})$: a family of pseudorandom functions. We use domain separation so that $\text{PRF.Eval}(\text{prf.k}, 0 \parallel \text{id})$ outputs an ℓ -bit string, while $\text{PRF.Eval}(\text{prf.k}, 1 \parallel \text{id} \parallel i)$ outputs randomness of the appropriate length for IBE.Keygen .

The construction is described next.

Setup $(1^\lambda, 1^n)$: For every $i \in [\ell]$ and $b \in \{0, 1\}$, the setup algorithm runs $(\text{ibe.mpk}_{i,b}, \text{ibe.msk}_{i,b}) \leftarrow \text{IBE.Setup}(1^\lambda, 1^n)$. It also samples a PRF key $\text{prf.k} \leftarrow \text{PRF.Setup}(1^\lambda)$. It outputs $\text{mpk} = \{\text{ibe.mpk}_{i,b}\}_{i \in [\ell], b \in \{0, 1\}}$ and $\text{msk} = (\text{prf.k}, \{\text{ibe.msk}_{i,b}\}_{i \in [\ell], b \in \{0, 1\}})$.

$\text{Keygen}(\text{msk}, \text{id})$: The key-generation algorithm computes $z_{\text{id}} = \text{PRF.Eval}(\text{prf.k}, 0 \parallel \text{id}) \in \{0, 1\}^\ell$. For every $i \in [\ell]$, it samples the key-generation randomness as $r_{\text{Keygen},i} = \text{PRF.Eval}(\text{prf.k}, 1 \parallel \text{id} \parallel i)$, and computes $\text{ibe.sk}_{\text{id},i} \leftarrow \text{IBE.Keygen}(\text{ibe.msk}_{i,z_{\text{id},i}}, \text{id}; r_{\text{Keygen},i})$. It outputs $\text{sk}_{\text{id}} = (z_{\text{id}}, \{\text{ibe.sk}_{\text{id},i}\}_{i \in [\ell]})$.

$\text{Enc}(\text{mpk}, \text{id}, m)$: For every $i \in [\ell]$ and $b \in \{0, 1\}$, the encryption algorithm computes $\text{ibe.ct}_{i,b} \leftarrow \text{IBE.Enc}(\text{ibe.mpk}_{i,b}, \text{id}, m_i)$. It outputs $\text{ct} = \{\text{ibe.ct}_{i,b}\}_{i \in [\ell], b \in \{0,1\}}$.

$\text{Dec}(\text{sk}_{\text{id}}, \text{ct})$: Parse $\text{sk}_{\text{id}} = (z_{\text{id}}, \{\text{ibe.sk}_{\text{id},i}\}_{i \in [\ell]})$ and $\text{ct} = \{\text{ibe.ct}_{i,b}\}_{i \in [\ell], b \in \{0,1\}}$. For every $i \in [\ell]$, the decryption algorithm computes $m_i \leftarrow \text{IBE.Dec}(\text{ibe.sk}_{\text{id},i}, \text{ibe.ct}_{i,z_{\text{id},i}})$. It outputs $m = m_1 \dots m_\ell$.

$\text{Sim}_1(1^\lambda, 1^n)$: For every $i \in [\ell]$ and $b \in \{0, 1\}$, the simulator runs $(\text{ibe.mpk}_{i,b}, \text{ibe.msk}_{i,b}) \leftarrow \text{IBE.Setup}(1^\lambda, 1^n)$. It initializes two empty tables T_z and T_{Keygen} , sets $\text{mpk} = \{\text{ibe.mpk}_{i,b}\}_{i,b}$, and outputs mpk together with the state $\text{st} = (\{\text{ibe.mpk}_{i,b}, \text{ibe.msk}_{i,b}\}_{i,b}, T_z, T_{\text{Keygen}})$.

$\text{Sim}_2(\text{st}, \text{id})$: Given $\text{st} = (\{\text{ibe.mpk}_{i,b}, \text{ibe.msk}_{i,b}\}_{i,b}, T_z, T_{\text{Keygen}})$, the simulator lazy-samples the entries in the tables. If $T_z[\text{id}]$ is undefined, it samples a fresh uniform string $z_{\text{id}} \leftarrow \{0, 1\}^\ell$ and stores it as $T_z[\text{id}]$; otherwise it sets $z_{\text{id}} = T_z[\text{id}]$.

For every $i \in [\ell]$, if $T_{\text{Keygen}}[\text{id}, i]$ is undefined, it samples fresh key-generation randomness $r_{\text{Keygen},i}$ and stores it as $T_{\text{Keygen}}[\text{id}, i]$; otherwise it sets $r_{\text{Keygen},i} = T_{\text{Keygen}}[\text{id}, i]$. It then computes

$$\text{ibe.sk}_{\text{id},i} \leftarrow \text{IBE.Keygen}(\text{ibe.msk}_{i,z_{\text{id},i}}, \text{id}; r_{\text{Keygen},i}).$$

Finally, it outputs $\text{sk}_{\text{id}} = (z_{\text{id}}, \{\text{ibe.sk}_{\text{id},i}\}_{i \in [\ell]})$ and the updated state.

$\text{Sim}_3(\text{st}, \text{id}^*)$: Given $\text{st} = (\{\text{ibe.mpk}_{i,b}, \text{ibe.msk}_{i,b}\}_{i,b}, T_z, T_{\text{Keygen}})$, the simulator samples a fresh uniform string $x^* \leftarrow \{0, 1\}^\ell$. For every $i \in [\ell]$, it computes $\text{ibe.ct}_{i,x_i^*}^* \leftarrow \text{IBE.Enc}(\text{ibe.mpk}_{i,x_i^*}, \text{id}^*, 0)$ and $\text{ibe.ct}_{i,1-x_i^*}^* \leftarrow \text{IBE.Enc}(\text{ibe.mpk}_{i,1-x_i^*}, \text{id}^*, 1)$. It sets $\text{ct}^* = \{\text{ibe.ct}_{i,b}^*\}_{i \in [\ell], b \in \{0,1\}}$, updates the state to

$$\text{st} = (\{\text{ibe.mpk}_{i,b}, \text{ibe.msk}_{i,b}\}_{i,b}, T_z, T_{\text{Keygen}}, \text{id}^*, x^*),$$

and outputs (ct^*, st) .

$\text{Sim}_4(\text{st}, m^*)$: Given $\text{st} = (\{\text{ibe.mpk}_{i,b}, \text{ibe.msk}_{i,b}\}_{i,b}, T_z, T_{\text{Keygen}}, \text{id}^*, x^*)$ and $m^* = m_1^* \dots m_\ell^*$, the simulator sets $z^* = x^* \oplus m^*$. For every $i \in [\ell]$, it samples fresh key-generation randomness $r_{\text{Keygen},i}$ and computes

$$\text{ibe.sk}_{\text{id}^*,i} \leftarrow \text{IBE.Keygen}(\text{ibe.msk}_{i,z_i^*}, \text{id}^*; r_{\text{Keygen},i}).$$

It outputs $\text{sk}_{\text{id}^*} = (z^*, \{\text{ibe.sk}_{\text{id}^*,i}\}_{i \in [\ell]})$.

Correctness. Let $\text{sk}_{\text{id}} = (z_{\text{id}}, \{\text{ibe.sk}_{\text{id},i}\}_{i \in [\ell]})$ be an honestly generated secret key for identity id , and let $\text{ct} = \{\text{ibe.ct}_{i,b}\}_{i \in [\ell], b \in \{0,1\}}$ be an honestly generated ciphertext encrypting $m = m_1 \dots m_\ell$ under identity id . For every $i \in [\ell]$, the ciphertext component $\text{ibe.ct}_{i,z_{\text{id},i}}$ is an IBE encryption of m_i under master public key $\text{ibe.mpk}_{i,z_{\text{id},i}}$ and identity id . The secret-key component $\text{ibe.sk}_{\text{id},i}$ is the corresponding IBE secret key for the same master secret key and identity. Therefore, by correctness of the underlying IBE scheme, $\text{IBE.Dec}(\text{ibe.sk}_{\text{id},i}, \text{ibe.ct}_{i,z_{\text{id},i}}) = m_i$ for every $i \in [\ell]$. Hence the decryption algorithm outputs $m = m_1 \dots m_\ell$.

Proof sketch. First replace the PRF outputs by consistently lazy-sampled random values. For each message bit, adaptive IBE security then allows us to change the ciphertext in the branch not selected by the challenge secret key from an encryption of m_i^* to an encryption of $1 - m_i^*$. The selected branch remains unchanged, so the reduction can reveal the challenge secret key and continue answering extraction queries for other identities. Writing $x^* = z_{id^*} \oplus m^*$, the resulting challenge ciphertext has exactly the distribution produced by Sim_3 , and the challenge key has the distribution produced by Sim_4 . The full hybrid argument is given in Appendix D.3.

D.3 Security of Weak RNC-IBE

We prove Theorem 4.2.

The proof proceeds through a sequence of hybrids. We first describe the hybrids and then show that adjacent hybrids are computationally indistinguishable.

1. **Hybrid $\mathcal{H}_1$:** This corresponds to the original weak RNC-IBE security game.

- Initialization Phase: $\mathcal{A}$ on input 1^λ outputs 1^n . The challenger runs the setup algorithm of Appendix D.2, obtains

$$\text{mpk} = \{\text{ibe.mpk}_{i,b}\}_{i \in [\ell], b \in \{0,1\}}$$

and

$$\text{msk} = (\text{prf.k}, \{\text{ibe.msk}_{i,b}\}_{i \in [\ell], b \in \{0,1\}}),$$

and sends mpk to $\mathcal{A}$.

- Pre-Challenge Query Phase: In this phase, $\mathcal{A}$ is allowed to make multiple key queries id . For each such query, the challenger computes $z_{\text{id}} = \text{PRF.Eval}(\text{prf.k}, 0 \parallel \text{id})$. For every $i \in [\ell]$, it computes $r_{\text{Keygen},i} = \text{PRF.Eval}(\text{prf.k}, 1 \parallel \text{id} \parallel i)$ and derives $\text{ibe.sk}_{\text{id},i} \leftarrow \text{IBE.Keygen}(\text{ibe.msk}_{i,z_{\text{id}},i}, \text{id}; r_{\text{Keygen},i})$. It sends $\text{sk}_{\text{id}} = (z_{\text{id}}, \{\text{ibe.sk}_{\text{id},i}\}_{i \in [\ell]})$ to $\mathcal{A}$.
- Challenge Phase: The adversary $\mathcal{A}$ sends (m^*, id^*) , where $m^* = m_1^* \dots m_\ell^*$. If id^* was queried in the pre-challenge query phase, the challenger aborts. Otherwise, for every $i \in [\ell]$ and $b \in \{0,1\}$, it computes $\text{ibe.ct}_{i,b}^* \leftarrow \text{IBE.Enc}(\text{ibe.mpk}_{i,b}, \text{id}^*, m_i^*)$. It sends $\text{ct}^* = \{\text{ibe.ct}_{i,b}^*\}_{i \in [\ell], b \in \{0,1\}}$ to $\mathcal{A}$.
- Post-Challenge Query Phase: In this phase, $\mathcal{A}$ is allowed to make multiple key queries id . If $\text{id} = \text{id}^*$, the challenger aborts. Otherwise, it generates sk_{id} exactly as in the pre-challenge query phase and sends it to $\mathcal{A}$.
- Challenge-Key Reveal Phase: The challenger generates the challenge secret key exactly as in a normal key-generation query for id^* . Namely, it computes $z_{\text{id}^*} = \text{PRF.Eval}(\text{prf.k}, 0 \parallel \text{id}^*)$. For every $i \in [\ell]$, it computes $r_{\text{Keygen},i} = \text{PRF.Eval}(\text{prf.k}, 1 \parallel \text{id}^* \parallel i)$ and derives $\text{ibe.sk}_{\text{id}^*,i} \leftarrow \text{IBE.Keygen}(\text{ibe.msk}_{i,z_{\text{id}^*},i}, \text{id}^*; r_{\text{Keygen},i})$. It sends $\text{sk}_{\text{id}^*} = (z_{\text{id}^*}, \{\text{ibe.sk}_{\text{id}^*,i}\}_{i \in [\ell]})$ to $\mathcal{A}$.
- Final Query Phase: In this phase, $\mathcal{A}$ may continue to make key queries $\text{id} \neq \text{id}^*$. The challenger answers each such query exactly as in the pre-challenge and post-challenge query phases.
- Response Phase: $\mathcal{A}$ outputs a bit.

2. **Hybrid $\mathcal{H}_2$:** This hybrid proceeds exactly as $\mathcal{H}_1$, except that the challenger no longer computes PRF values using prf.k . Instead, it samples the outputs of lazy-sampled random functions and reuses them consistently across repeated queries. More precisely, the challenger maintains two tables T_z and T_{Keygen} .

Whenever the challenger needs the string associated with an identity id , if $T_z[\text{id}]$ is undefined it samples a fresh uniform string in $\{0, 1\}^\ell$ and stores it as $T_z[\text{id}]$; otherwise it reuses the stored value. Similarly, whenever it needs the key-generation randomness associated with (id, i) , if $T_{\text{Keygen}}[\text{id}, i]$ is undefined it samples fresh uniform randomness of the appropriate length and stores it as $T_{\text{Keygen}}[\text{id}, i]$; otherwise it reuses the stored value.

Thus, in every phase of the experiment, the objects are generated exactly as in $\mathcal{H}_1$, except that every occurrence of $\text{PRF.Eval}(\text{prf.k}, 0 \parallel \text{id})$ is replaced by $T_z[\text{id}]$, and every occurrence of $\text{PRF.Eval}(\text{prf.k}, 1 \parallel \text{id} \parallel i)$ is replaced by $T_{\text{Keygen}}[\text{id}, i]$.

3. **Hybrid $\mathcal{H}_3$:** This hybrid proceeds exactly as $\mathcal{H}_2$, except that the challenge ciphertext is modified on the branch that will not be opened by the challenge secret key. More precisely, at the challenge phase the challenger first lazy-samples $z_{\text{id}^*} = T_z[\text{id}^*]$. Then, for every $i \in [\ell]$, it computes

$$\text{ibe.ct}_{i, z_{\text{id}^*, i}}^* \leftarrow \text{IBE.Enc}(\text{ibe.mpk}_{i, z_{\text{id}^*, i}}, \text{id}^*, m_i^*)$$

and

$$\text{ibe.ct}_{i, 1 - z_{\text{id}^*, i}}^* \leftarrow \text{IBE.Enc}(\text{ibe.mpk}_{i, 1 - z_{\text{id}^*, i}}, \text{id}^*, 1 - m_i^*).$$

All other phases remain unchanged from $\mathcal{H}_2$.

We now show that consecutive hybrids are computationally indistinguishable.

Lemma D.2 *Assuming PRF is a secure pseudorandom function, then for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all λ ,*

$$|\Pr[\mathcal{H}_1(\lambda, \mathcal{A}) = 1] - \Pr[\mathcal{H}_2(\lambda, \mathcal{A}) = 1]| \leq \text{negl}(\lambda).$$

proof The only difference between $\mathcal{H}_1$ and $\mathcal{H}_2$ is how the values z_{id} and $r_{\text{Keygen}, i}$ are generated. In $\mathcal{H}_1$, they are computed as outputs of the PRF on inputs $0 \parallel \text{id}$ and $1 \parallel \text{id} \parallel i$, respectively. In $\mathcal{H}_2$, they are replaced by outputs of lazy-sampled truly random functions. Hence, any adversary that distinguishes $\mathcal{H}_1$ from $\mathcal{H}_2$ can be transformed into a PRF distinguisher by using its oracle to answer all such queries consistently throughout the simulation. If the oracle is the real PRF, the simulation is exactly $\mathcal{H}_1$. If the oracle is truly random, the simulation is exactly $\mathcal{H}_2$. Therefore, the distinguishing gap between $\mathcal{H}_1$ and $\mathcal{H}_2$ is bounded by the PRF advantage, which is negligible.

Lemma D.3 *Assuming IBE is an adaptively secure IBE scheme, then for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all λ ,*

$$|\Pr[\mathcal{H}_2(\lambda, \mathcal{A}) = 1] - \Pr[\mathcal{H}_3(\lambda, \mathcal{A}) = 1]| \leq \text{negl}(\lambda).$$

proof We introduce intermediate hybrids $\mathcal{H}_{2,0}, \dots, \mathcal{H}_{2,\ell}$. The hybrid $\mathcal{H}_{2,j}$ is defined as follows. For the first j indices, the challenge ciphertext is generated as in $\mathcal{H}_3$, while for the remaining indices it is generated as in $\mathcal{H}_2$. More explicitly, for every $i \leq j$,

$$\text{ibe.ct}_{i, z_{\text{id}^*, i}}^* \leftarrow \text{IBE.Enc}(\text{ibe.mpk}_{i, z_{\text{id}^*, i}}, \text{id}^*, m_i^*)$$

and

$$\text{ibe.ct}_{i,1-z_{\text{id}^*,i}}^* \leftarrow \text{IBE.Enc}(\text{ibe.mpk}_{i,1-z_{\text{id}^*,i}}, \text{id}^*, 1 - m_i^*),$$

whereas for every $i > j$ and $b \in \{0, 1\}$,

$$\text{ibe.ct}_{i,b}^* \leftarrow \text{IBE.Enc}(\text{ibe.mpk}_{i,b}, \text{id}^*, m_i^*).$$

Thus, $\mathcal{H}_{2,0}$ is exactly $\mathcal{H}_2$, and $\mathcal{H}_{2,\ell}$ is exactly $\mathcal{H}_3$.

It remains to show that $\mathcal{H}_{2,i^*-1}$ and $\mathcal{H}_{2,i^*}$ are computationally indistinguishable for every $i^* \in [\ell]$. These two hybrids differ only in the ciphertext $\text{ibe.ct}_{i^*,1-z_{\text{id}^*,i^*}}^*$, which is changed from an IBE encryption of $m_{i^*}^*$ to an IBE encryption of $1 - m_{i^*}^*$ under the master public key $\text{ibe.mpk}_{i^*,1-z_{\text{id}^*,i^*}}$ and identity id^* .

Suppose there exists an adversary $\mathcal{A}$ that distinguishes $\mathcal{H}_{2,i^*-1}$ from $\mathcal{H}_{2,i^*}$ with non-negligible advantage. We construct an adversary $\mathcal{B}$ against the adaptive security of the underlying IBE scheme. The reduction $\mathcal{B}$ guesses a bit $\gamma \leftarrow \{0, 1\}$ and embeds its IBE challenge master public key as $\text{ibe.mpk}_{i^*,\gamma}$. It generates all other IBE master public keys and master secret keys honestly. It then runs $\mathcal{A}$ with the resulting master public key of the weak RNC-IBE scheme.

The reduction answers all key queries, including those made after the challenge-key reveal phase, exactly as in $\mathcal{H}_2$. For the embedded IBE instance (i^*, γ) , whenever it needs a secret key for an identity id , it obtains this key by making an extraction query to its IBE challenger. All other secret-key components are generated locally using the corresponding master secret keys.

When $\mathcal{A}$ outputs the challenge pair (m^*, id^*) , the reduction lazy-samples $z_{\text{id}^*} = T_z[\text{id}^*]$ if necessary. If $\gamma \neq 1 - z_{\text{id}^*,i^*}$, the reduction aborts and outputs a uniform bit. Otherwise, it proceeds as follows. For every challenge ciphertext component except $\text{ibe.ct}_{i^*,\gamma}^*$, the reduction generates the component exactly according to the current hybrid. To generate $\text{ibe.ct}_{i^*,\gamma}^*$, it submits the identity id^* and the two messages $(m_{i^*}^*, 1 - m_{i^*}^*)$ to its IBE challenger. It receives the challenge ciphertext and uses it as $\text{ibe.ct}_{i^*,\gamma}^*$.

The extraction queries made by $\mathcal{B}$ are valid. Indeed, in the response phase for the challenge identity id^* , the weak RNC-IBE secret key contains the IBE secret key for branch z_{id^*,i^*} , whereas the embedded IBE challenge ciphertext is under branch $\gamma = 1 - z_{\text{id}^*,i^*}$. Therefore, $\mathcal{B}$ never asks for an extraction key for the IBE challenge identity id^* in the embedded IBE instance.

Conditioned on the event $\gamma = 1 - z_{\text{id}^*,i^*}$, if the IBE challenger encrypts $m_{i^*}^*$, then $\mathcal{B}$ perfectly simulates $\mathcal{H}_{2,i^*-1}$ towards $\mathcal{A}$. If the IBE challenger encrypts $1 - m_{i^*}^*$, then $\mathcal{B}$ perfectly simulates $\mathcal{H}_{2,i^*}$ towards $\mathcal{A}$. Since the event $\gamma = 1 - z_{\text{id}^*,i^*}$ occurs with probability $1/2$, any non-negligible distinguishing gap between the two hybrids yields a non-negligible advantage against the adaptive security of IBE. Hence each adjacent pair $\mathcal{H}_{2,i^*-1}$ and $\mathcal{H}_{2,i^*}$ is computationally indistinguishable. By a hybrid argument over $i^* \in [\ell]$, and since ℓ is polynomially bounded, $\mathcal{H}_2$ and $\mathcal{H}_3$ are computationally indistinguishable.

It remains to observe that $\mathcal{H}_3$ is exactly the simulated experiment. In $\mathcal{H}_3$, define $x^* = z_{\text{id}^*} \oplus m^*$. Since z_{id^*} is uniformly random and independent of m^* , the string x^* is also uniformly random. Moreover, for every $i \in [\ell]$, if $m_i^* = 0$, then $x_i^* = z_{\text{id}^*,i}$, and the ciphertext components are encryptions of 0 under branch x_i^* and 1 under branch $1 - x_i^*$. If $m_i^* = 1$, then $x_i^* = 1 - z_{\text{id}^*,i}$, and again the ciphertext components are encryptions of 0 under branch x_i^* and 1 under branch $1 - x_i^*$. Thus the challenge ciphertext in $\mathcal{H}_3$ is distributed exactly as the output of Sim_3 .

Finally, in the challenge-key reveal phase of $\mathcal{H}_3$, the challenge secret key uses $z_{\text{id}^*} = x^* \oplus m^*$, which is exactly how Sim_4 opens the simulated ciphertext to m^* . Any subsequent key queries for

identities $\text{id} \neq \text{id}^*$ are still answered exactly as in the simulated experiment using Sim_2 . Therefore, $\mathcal{H}_3$ is exactly the simulated experiment with $\text{Sim}_1, \dots, \text{Sim}_4$ defined in Appendix D.2.

Combining Lemmas D.2 and D.3, and using the triangle inequality, we conclude that for every PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathcal{H}_1(\lambda, \mathcal{A}) = 1] - \Pr[\mathcal{H}_3(\lambda, \mathcal{A}) = 1]| \leq \text{negl}(\lambda).$$

This proves that the construction is a secure weak RNC-IBE scheme.

D.4 Correctness of the Epoch-Based Construction

We prove Lemma 4.3.

proof Let (pk, sk) be honestly generated, let τ be any epoch, let $(\text{CT}_1, \dots, \text{CT}_m)$ be ciphertexts encrypting messages $m_1, \dots, m_m$ under identities $\text{id}_1, \dots, \text{id}_m$, and let $i^* \in [m]$. We show that decryption of CT_{i^*} using the succinct batch key output by PartDec yields m_{i^*} except with negligible probability.

Write

$$\mathbf{h} \leftarrow \text{RBE.Hash}(\text{pp}, (\text{id}_1, \dots, \text{id}_m)).$$

For every $j \in [k]$, the algorithm PartDec computes

$$\sigma_j \leftarrow \text{IBE.Keygen}(\text{msk}_{j, \mathbf{h}_j}, \tau).$$

On the other hand, during encryption of CT_{i^*} , the garbling algorithm produces labels $\text{lab}_{j,0}, \text{lab}_{j,1}$ for the circuit $\text{C}[\text{pp}, \text{id}_{i^*}, m_{i^*}, r_{i^*}]$, and the construction encrypts each label as

$$\text{ct}_{i^*, j, b} \leftarrow \text{IBE.Enc}(\text{mpk}_{j, b}, \tau, \text{lab}_{j, b}).$$

In the decryption algorithm, we compute

$$(\mathbf{h}', \eta) \leftarrow \text{RBE.Hint}(\text{pp}, \text{id}_{i^*}, (\text{id}_1, \dots, \text{id}_m)).$$

By the consistency assumption on RBE.Hash and RBE.Hint , we have $\mathbf{h}' = \mathbf{h}$. Hence, by correctness of the IBE scheme, for every $j \in [k]$,

$$\hat{\text{lab}}_j = \text{IBE.Dec}(\sigma_j, \text{ct}_{i^*, j, \mathbf{h}_j}) = \text{lab}_{j, \mathbf{h}_j}$$

except with negligible probability.

Therefore, the decryption algorithm evaluates the garbled circuit GC_{i^*} on exactly the labels corresponding to the input $\mathbf{h}$. By correctness of the garbling scheme, this yields

$$c = \text{Eval}(\text{GC}_{i^*}, \{\hat{\text{lab}}_j\}_{j \in [k]}) = \text{RBE.Enc}(\text{pp}, \mathbf{h}, \text{id}_{i^*}, m_{i^*}; r_{i^*})$$

except with negligible probability. Finally, by correctness of the RBE scheme,

$$\text{RBE.Dec}(\text{pp}, \eta, c) = m_{i^*}$$

except with negligible probability.

Combining the above and using a union bound over the negligible failure probabilities of the underlying primitives, we conclude that

$$\text{Dec}(\text{sbk}, i^*, \text{CT}_1, \dots, \text{CT}_m) = m_{i^*}$$

except with negligible probability. This proves correctness of the construction.

D.5 Security of the Epoch-Based Construction

The following lemmas justify the hybrid transitions in the proof of Theorem 4.4.

Lemma D.4 *Assume that IBE is a secure weak RNC-IBE scheme in the sense of Definition D.1. Then for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathcal{H}_0(\lambda, \mathcal{A}) = 1] - \Pr[\mathcal{H}_1(\lambda, \mathcal{A}) = 1]| \leq \text{negl}(\lambda).$$

proof We distinguish three exhaustive cases according to when the unique succinct batch key for the challenge epoch τ^* is obtained: either no such batch key is ever obtained, or it is obtained after the challenge, or it is obtained before the challenge. It suffices to bound the distinguishing gap in each of these cases separately.

Consider first the cases in which no succinct batch key for τ^* is ever obtained, or in which it is obtained only after the challenge. In both cases we may switch the weak RNC-IBE ciphertexts contained in CT^* one instance at a time. Order the pairs $(j, b) \in [k] \times \{0, 1\}$ lexicographically and let $\mathcal{H}_{0,j,b}$ denote the experiment in which all challenge components preceding (j, b) have already been switched to simulated mode, while $\text{ct}_{j,b}^*$ and all later components are still generated honestly. Thus $\mathcal{H}_{0,1,0} = \mathcal{H}_0$, and the final hybrid in this sequence is exactly $\mathcal{H}_1$.

We claim that each adjacent pair of hybrids is computationally indistinguishable. Indeed, fix (j, b) and assume that some adversary distinguishes the two adjacent hybrids in which only the component $\text{ct}_{j,b}^*$ changes. We build an adversary $\mathcal{B}_{j,b}$ against the weak RNC-IBE security of the instance indexed by (j, b) . The reduction $\mathcal{B}_{j,b}$ generates all other weak RNC-IBE instances honestly and simulates the batch decryption experiment for $\mathcal{A}$. It uses the challenge public key of its weak RNC-IBE challenger as $\text{mpk}_{j,b}$, while all other challenge components are generated exactly as required by the current hybrid. If the unique succinct batch key for τ^* is requested after the challenge and satisfies $h_j^* = b$, then $\mathcal{B}_{j,b}$ forwards the challenge-key-reveal key of its weak RNC-IBE challenger as the j th component of the succinct batch key. Any subsequent **PartDec** queries at epochs different from τ^* are answered using the final query phase of the weak RNC-IBE experiment. If $h_j^* \neq b$, or if no such batch key is ever requested, the challenge-key-reveal key is ignored. Therefore, when the weak RNC-IBE challenger is in the real world, $\mathcal{B}_{j,b}$ simulates the earlier hybrid, and when it is in the simulated world it simulates the later hybrid. By Definition D.1, each adjacent gap is negligible, and a union bound over the $2k$ challenge components shows that $\mathcal{H}_0$ and $\mathcal{H}_1$ are indistinguishable in these two cases.

It remains to consider the case in which the unique succinct batch key for τ^* is obtained before the challenge. Let

$$h^* = \text{RBE.Hash}(\text{pp}, (\text{id}_1, \dots, \text{id}_m))$$

be the hash value of the unique pre-challenge batch queried at epoch τ^* . In this case the adversary already holds, before seeing CT^* , the weak RNC-IBE secret keys corresponding to the instances (j, h_j^*) , and therefore will only ever be able to decrypt the challenge components $\text{ct}_{j,h_j^*}^*$. Hence it suffices to modify only the sibling ciphertexts $\text{ct}_{j,1-h_j^*}^*$.

For $j \in [k]$, let $\tilde{\mathcal{H}}_{0,j}$ denote the experiment obtained from $\mathcal{H}_0$ by replacing, for every wire index up to j , the sibling ciphertext $\text{ct}_{i,1-h_i^*}^*$ by a weak RNC-IBE challenge ciphertext generated in simulated mode. The final hybrid $\tilde{\mathcal{H}}_{0,k}$ is exactly the branch of $\mathcal{H}_1$ relevant for the present case, since every undecryptable sibling ciphertext has been switched to simulated mode while every decryptable component remains honest.

We claim that each adjacent pair $\tilde{\mathcal{H}}_{0,j-1}$ and $\tilde{\mathcal{H}}_{0,j}$ is computationally indistinguishable. Fix $j \in [k]$ and suppose some adversary distinguishes these two hybrids. We build an adversary $\mathcal{B}_j$ against the weak RNC-IBE security of a single embedded instance. The reduction $\mathcal{B}_j$ first guesses a bit $\gamma \leftarrow \{0, 1\}$ and uses the challenge public key of its weak RNC-IBE challenger as $\text{mpk}_{j,\gamma}$, while generating all other weak RNC-IBE instances honestly. It answers all **PartDec** queries on epochs different from τ^* by forwarding the corresponding epoch as a key query to its weak RNC-IBE challenger for the embedded instance and generating all other components locally.

Since the unique succinct batch key for τ^* is obtained before the challenge, the reduction learns the bit h_j^* before the challenge phase is reached. If $\gamma \neq 1 - h_j^*$, the reduction aborts and outputs a uniform bit. Otherwise, the embedded instance is exactly the sibling instance $(j, 1 - h_j^*)$ that never needs to be opened by the pre-challenge batch key. Indeed, that batch key only requires keys for the instances (i, h_i^*) , and no further **PartDec** query on epoch τ^* is answered after the challenge.

Conditioned on the event $\gamma = 1 - h_j^*$, at the challenge phase $\mathcal{B}_j$ forwards the pair $(\text{lab}_{j,1-h_j^*}, \tau^*)$ to its weak RNC-IBE challenger and receives a challenge ciphertext. It uses this ciphertext as $\text{ct}_{j,1-h_j^*}^*$, while all other challenge components are generated exactly as required by the hybrid being simulated. If the weak RNC-IBE challenger is in the real world, then $\mathcal{B}_j$ simulates $\tilde{\mathcal{H}}_{0,j-1}$. If the challenger is in the simulated world, then $\mathcal{B}_j$ simulates $\tilde{\mathcal{H}}_{0,j}$. The challenge-key-reveal key of the weak RNC-IBE experiment is never needed and may be ignored. Therefore, by Definition D.1, the distinguishing gap between the two hybrids, conditioned on $\gamma = 1 - h_j^*$, is negligible. Since this event occurs with probability $1/2$, the unconditional gap is negligible as well. Another union bound over the k wire indices shows that $\mathcal{H}_0$ and the branch of $\mathcal{H}_1$ relevant for the present case are computationally indistinguishable.

Combining the three cases proves the lemma.

Lemma D.5 *Assume that GC is an adaptive secure garbling scheme. Then for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathcal{H}_1(\lambda, \mathcal{A}) = 1] - \Pr[\mathcal{H}_2(\lambda, \mathcal{A}) = 1]| \leq \text{negl}(\lambda).$$

proof Again we distinguish the three exhaustive cases according to whether the unique succinct batch key for the challenge epoch τ^* is obtained never, after the challenge, or before the challenge.

We first consider the cases in which no succinct batch key for τ^* is ever obtained, or in which it is obtained only after the challenge. Let

$$C^*(h) = \text{RBE.Enc}(\text{pp}, h, \text{id}^*, m_b^*)$$

be the circuit underlying the challenge ciphertext. We construct an adversary $\mathcal{B}$ against adaptive security of the garbling scheme. The reduction $\mathcal{B}$ simulates the batch-decryption experiment for $\mathcal{A}$ as follows. It generates all non-challenge ciphertexts honestly. For every pair (j, b) it runs the weak RNC-IBE simulator to obtain a public key and the state needed to later explain the challenge ciphertext component $\text{ct}_{j,b}^*$. It forwards the circuit C^* to its garbling challenger and receives a garbled circuit that is either honestly generated or simulated. This garbled circuit is used as the challenge component GC^* , while every weak RNC-IBE challenge ciphertext component is generated in simulated mode.

If the adversary never obtains a succinct batch key for τ^* , then the reduction never needs any garbled labels. Otherwise, let h^* be the hash of the unique batch queried at epoch τ^* . At that

point, $\mathcal{B}$ asks its garbling challenger for the labels corresponding to the input h^* . For every $j \in [k]$, it then invokes the weak RNC-IBE simulator of the instance (j, h_j^*) on the label associated with the j th input wire and obtains the challenge-identity key needed to answer the succinct batch-key query. No other challenge-identity key is ever needed. Consequently, if the garbling challenger returns the real distribution, then $\mathcal{B}$ simulates the relevant branch of $\mathcal{H}_1$, while if it returns the simulated distribution, then $\mathcal{B}$ simulates the corresponding branch of $\mathcal{H}_2$. By adaptive security of GC , the distinguishing gap is negligible.

It remains to consider the case in which the unique succinct batch key for τ^* is obtained before the challenge. In this case the queried batch, and hence its hash value h^* , are fixed before the challenge phase. The relevant branch of $\mathcal{H}_1$ is the one identified in the proof of Lemma D.4: the decryptable challenge components $\text{ct}_{j,h_j^*}^*$ remain unchanged, while the sibling ciphertexts are already in simulated mode. Therefore the adversary effectively sees one full input-label set corresponding to the single input h^* and no useful labels for any other input.

We now reduce directly to adaptive security of the garbling scheme. The reduction forwards the circuit C^* to its garbling challenger and requests the labels for the fixed input h^* . It uses the returned labels to form the challenge weak RNC-IBE ciphertexts that are decryptable under the unique pre-challenge succinct batch key, while the undecryptable sibling positions are generated via the weak RNC-IBE simulator exactly as prescribed by the relevant branch of $\mathcal{H}_1$. If the garbling challenger returns an honest garbling and the corresponding labels, the resulting simulation is exactly the present branch of $\mathcal{H}_1$. If it returns a simulated garbling and simulated labels, then the resulting simulation is exactly the corresponding branch of $\mathcal{H}_2$. Thus the distinguishing gap in this case is negligible as well.

Combining the three cases proves the lemma.

Lemma D.6 *Assume that RBE is a non-inclusion secure registration based encryption scheme. Then for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathcal{H}_2(\lambda, \mathcal{A}) = 1] - \Pr[\mathcal{H}_3(\lambda, \mathcal{A}) = 1]| \leq \text{negl}(\lambda).$$

proof We prove the lemma by a direct reduction to the non-inclusion security of RBE. As above, we distinguish three cases according to whether the unique succinct batch key for the challenge epoch τ^* is obtained never, after the challenge, or before the challenge.

Let $\mathcal{B}$ be an adversary against the RBE security experiment. On input the public parameters pp from its challenger, $\mathcal{B}$ simulates the entire batch-decryption experiment for $\mathcal{A}$. It generates all weak RNC-IBE instances honestly or in simulated mode exactly as prescribed by $\mathcal{H}_2$ and $\mathcal{H}_3$, and it also generates the simulated garbled circuit component GC^* using the garbling simulator. The only object that $\mathcal{B}$ does not generate by itself is the RBE ciphertext underlying the simulated output of GC^* .

We now explain how $\mathcal{B}$ determines the corresponding hash value h^* . If no succinct batch key for τ^* is ever obtained, $\mathcal{B}$ fixes any tuple of pairwise distinct identities $(\hat{\text{id}}_1, \dots, \hat{\text{id}}_m)$ that does not contain id^* and sets

$$h^* = \text{RBE.Hash}(\text{pp}, (\hat{\text{id}}_1, \dots, \hat{\text{id}}_m)).$$

If the unique succinct batch key for τ^* is obtained after the challenge, then $\mathcal{B}$ lets h^* be the hash of that queried batch. Finally, if the unique succinct batch key for τ^* is obtained before the challenge, then $\mathcal{B}$ lets h^* be the hash of the unique pre-challenge batch queried at epoch τ^* .

In all three cases, $\mathcal{B}$ submits the tuple defining h^* together with the challenge identity id^* and messages $(m_b^*, 0)$ to its RBE challenger. The crucial point is that this always constitutes a valid non-inclusion challenge. In the first case, this is immediate by construction. In the second case, it follows from the post-challenge restriction in the batch decryption security game, which requires the unique admissible query at τ^* to exclude id^* . In the third case, it follows from the challenge-phase abort condition, which rules out the possibility that the unique pre-challenge batch queried at τ^* already contains a ciphertext under identity id^* .

Let ct_{RBE}^* be the ciphertext returned by the RBE challenger. The reduction now uses ct_{RBE}^* as the output value with which it explains the simulated garbling. Concretely, when the unique succinct batch key for τ^* is needed, $\mathcal{B}$ invokes the garbling input simulator on the value ct_{RBE}^* and uses the resulting labels to finalize the simulated weak RNC-IBE instances exactly as in $\mathcal{H}_2$ and $\mathcal{H}_3$. All remaining components are distributed identically in both hybrids.

If the RBE challenger encrypts m_b^* , then ct_{RBE}^* is distributed exactly as $\text{RBE.Enc}(\text{pp}, h^*, \text{id}^*, m_b^*)$, and therefore $\mathcal{B}$ simulates $\mathcal{H}_2$ for $\mathcal{A}$. If the RBE challenger encrypts 0, then ct_{RBE}^* is distributed exactly as $\text{RBE.Enc}(\text{pp}, h^*, \text{id}^*, 0)$, and therefore $\mathcal{B}$ simulates $\mathcal{H}_3$ for $\mathcal{A}$. Hence any distinguisher between $\mathcal{H}_2$ and $\mathcal{H}_3$ yields a distinguisher against the non-inclusion security of RBE, and the distinguishing gap is negligible.

D.6 Alternative Instantiation Paths

Beyond the CDH route used in Corollary 4.5, the closely related laconic-encryption viewpoint yields efficient lattice-based realizations of the restricted RBE interface, for instance from LWE/RLWE [DKL⁺23]. More generally, Theorem 4.4 applies whenever adaptively secure garbling, non-inclusion secure RBE, and weak RNC-IBE are available under a desired assumption. Theorem 4.2 supplies the last ingredient from standard IBE and pseudorandom functions.

D.7 Simultaneous Multi-Challenge Security

The guarantee of Theorem 4.4 concerns the single-challenge experiment of Definition 4.1.

We also define a multi-challenge security notion, in which the adversary specifies all challenge tuples simultaneously.

Definition D.7 *An identity-based encryption scheme with batch decryption is said to be multi-challenge secure if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that, for all $\lambda \in \mathbb{N}$,*

$$\Pr[b' = b] \leq \frac{1}{2} + \text{negl}(\lambda),$$

*where the probability is defined with respect to the experiment of Definition 4.1, except that in the challenge phase the adversary submits an epoch τ^* together with a set $\{(m_{i,0}^*, m_{i,1}^*, \text{id}_i^*)\}_{i \in [\ell]}$, where the identities $\text{id}_1^*, \dots, \text{id}_\ell^*$ are pairwise distinct. The challenger aborts if more than one **PartDec** query on epoch τ^* was made in the pre-challenge phase, or if the unique such query, when it exists, contains a ciphertext CT such that $\text{Id}(\text{CT}) \in \{\text{id}_1^*, \dots, \text{id}_\ell^*\}$. Otherwise, it samples a random bit $b \leftarrow \{0, 1\}$, computes $\text{CT}_i^* \leftarrow \text{Enc}(\text{pk}, \tau^*, \text{id}_i^*, m_{i,b}^*)$ for all $i \in [\ell]$, returns the challenge ciphertexts $(\text{CT}_1^*, \dots, \text{CT}_\ell^*)$ to $\mathcal{A}$, and answers post-challenge queries exactly as in the single-challenge experiment except that it aborts whenever $\tau = \tau^*$ and either a **PartDec** query on epoch τ^* was already answered in the pre-challenge phase or a previous post-challenge query was already made on epoch τ^* , or one of the*

challenge ciphertexts $\text{CT}_1^*, \dots, \text{CT}_\ell^*$ appears in the queried tuple, or one of the challenge identities $\text{id}_1^*, \dots, \text{id}_\ell^*$ appears in $\{\text{Id}(\text{CT}_1), \dots, \text{Id}(\text{CT}_\ell)\}$. Finally, it receives a bit b' from $\mathcal{A}$.