Verifiable Registration-Based Encryption

Rishab Goyal
UT Austin
rgoyal@cs.utexas.edu*

Satyanarayana Vusirikala
UT Austin
satya@cs.utexas.edu†

Abstract

In a recent work, Garg, Hajiabadi, Mahmoody, and Rahimi [GHMR18] introduced a new encryption framework, which they referred to as Registration-Based Encryption (RBE). The central motivation behind RBE was to provide a novel methodology for solving the well-known *key-escrow* problem in Identity-Based Encryption (IBE) systems [Sha85]. Informally, in an RBE system there is no private-key generator unlike IBE systems, but instead it is replaced with a public *key accumulator*. Every user in an RBE system samples its own public-secret key pair, and sends the public key to the accumulator for registration. The key accumulator has no secret state, and is only responsible for compressing all the registered user identity-key pairs into a short public commitment. Here the encryptor only requires the compressed parameters along with the target identity, whereas a decryptor requires supplementary key material along with the secret key associated with the registered public key.

The initial construction in [GHMR18] based on standard assumptions only provided weak efficiency properties. In a follow-up work by Garg, Hajiabadi, Mahmoody, Rahimi, and Sekar [GHM+19], they gave an efficient RBE construction from standard assumptions. However, both these works considered the key accumulator to be honest which might be too strong an assumption in real-world scenarios. In this work, we initiate a formal study of RBE systems with *malicious* key accumulators. To that end, we introduce a strengthening of the RBE framework which we call *Verifiable RBE* (VRBE). A VRBE system additionally gives the users an extra capability to obtain *short* proofs from the key accumulator proving correct (and unique) registration for every registered user as well as proving non-registration for any yet unregistered identity.

We construct VRBE systems which provide succinct proofs of registration and non-registration from standard assumptions (such as CDH, Factoring, LWE). Our proof systems also naturally allow a much more efficient audit process which can be perfomed by any non-participating third party as well. A by-product of our approach is that we provide a more efficient RBE construction than that provided in the prior work of Garg et al. [GHM+19]. And, lastly we initiate a study on extension of VRBE to a wider range of access and trust structures.

1 Introduction

Public-key encryption (PKE) [DH76, RSA78, GM84] has remained a cornerstone in modern-day cryptography and has been one of the most widely used and studied cryptographic primitive. Traditionally, a public-key encryption system enables a one-to-one private communication channel between any two users over a public broadcast network as long as it is possible to disambiguate any user's public key information honestly. Over the last few decades, significant research effort has been made by the cryptographic community in re-envisioning the original goals of public-key encryption, in turn pushing towards more expressiveness from such systems. This effort has lead to introduction of encryption systems with better functionalities such as Identity-Based Encryption (IBE) [Sha85, Coc01, BF01], Attribute-Based Encryption (ABE) [SW05, GPSW06], and most notably Functional Encryption (FE) [SW08, BSW11] which is meant to encapsulate both IBE and ABE functionalities.

*Supported by IBM PhD Fellowship.
†Supported by NSF CNS-1908611, CNS-1414082, DARPA SafeWare and Packard Foundation Fellowship.

Very briefly, in FE systems there is a trusted authority which sets up the system by sampling public parameters pp along with a master secret key msk. The public parameters pp can be used by any party to encrypt a message m of its choice, while the master key msk enables the generation of certain private decryption keys sk_f for any function f in the associated function class. The most useful aspect of such systems is that decryption now leads to users either conditionally learning the full message (as in IBE/ABE where the condition is specified at encryption time) or learning some partial information about the message such as $f(m)$ (as in general FE). The security of all such systems guarantees that no computationally bounded adversary should be able to learn anything other than what can be uncovered using the private decryption keys in its possession.

Notably, in all such highly expressive systems it is crucial that the master key msk is never compromised as given the master key any adversary can arbitrarily sample private decryption keys to learn desired messages. Thus, an unfortunate consequence of adding such powerful functionalities to public-key cryptosystems is the introduction of a central trusted authority (or key generator) which is responsible for sampling the public parameters, distributing the private decryption keys to authorized users, and most importantly securely storing the master secret key. Now, this could be very worrisome for many applications, since the authority must be fully trustworthy, otherwise, it would turn out to be a single point of failure. While it would be quite reasonable to put some trust in the central authority, it so happens that even an honest-but-curious key generator can cause great havoc in such an environment. Specifically, any honest-but-curious key generator can arbitrarily decrypt ciphertexts that are intended for specific recipients since it has the master key. And, it could perform such an attack in an undetectable way. This problem is widely regarded as the "key-escrow" problem.

While many previous works ([BF01, CHSS02, ARP03, CCV04, Goy07, PS08, Cho09, KG10] to name a few) have suggested different approaches to solving the key-escrow problem, none of these solutions were able to resolve the key-escrow problem completely. Very recently, in a beautiful work by Garg, Hajiabadi, Mahmoody, and Rahimi [GHMR18], a novel approach for handling key-escrow was proposed. The central motivation of that work was to remove the requirement of private key generators completely from IBE systems, and to that end, they introduced the notion of Registration-Based Encryption (RBE). In an RBE system, each user samples its own public-secret key pair, and the private key generator is replaced with a public key accumulator. Every user registers their public key and identity information with the key accumulator, and the job of a key accumulator is to compress all these user identity-key pairs into a short public commitment with efficiently computable openings. Here the commitment is set as the public parameters of the RBE system, and the user-specific openings are used as supplementary key information during decryption. Now ideally one would expect the registration process to be time-unrestricted, that is users must be allowed to register at arbitrary time intervals. However, this would imply that public parameters will get updated after every registration, which could possibly lead to every registered user requesting fresh supplementary key information after another user registers. Thus, to make the notion more attractive, [GHMR18] required the following efficiency properties from an RBE system — (1) public parameters must be short, i.e. $|\mathsf{pp}| = \mathsf{poly}(\lambda, \log n)$ where λ is the security parameter and n is the number of users registered so far, (2) the registration process as well as the supplementary key generation process must be efficient, i.e. must run in time $\mathsf{poly}(\lambda, \log n)$, (3) number of times any user needs to request a fresh supplementary key from the accumulator (over the lifetime of the system) is also $\mathsf{poly}(\lambda, \log n)$. In short, an RBE system is meant to be a public key accumulation service which provides efficient and adaptive user registration while avoiding the problems associated with a private key generator.

In a sequence of two works [GHMR18, GHM+19], efficient construction of RBE systems were provided from a wide variety of assumptions (such as CDH, Factoring, LWE, iO). Specifically, [GHMR18] gave an efficient construction from indistinguishability obfuscation (iO) [BGI+01, GGH+13], and a weakly efficient construction from hash garbling scheme [GHMR18]. In a follow-up work by Garg et al. [GHM+19], a fully efficient RBE construction from hash garbling was provided. At first glance, it seems like efficient constructions for RBE systems fill the gap between regular PKE systems (which do not suffer from key-escrow but also do not provide any extra functionality) and IBE systems (which permit a simpler identity-based encryption paradigm but suffers from key-escrow). However, it turns out there is still a significant gap due to

which even RBE systems potentially could be surprisingly compromised due to a corrupt key accumulator. To better understand the gap, let us look back at the excerpt from Rogaway's essay [Rog15] which was one of the prompts behind the initial work on RBE in [GHMR18]:

> *"[. . .] But this convenience is enabled by a radical change in the trust model: Bob's secret key is no longer self-selected. It is issued by a trusted authority. That authority knows everyone's secret key in the system. IBE embeds key-escrow indeed a form of key-escrow where a single entity implicitly holds all secret keys even ones that haven't yet been issued. [. . .] "*

Now an RBE system solves the problem of self-selection of Bob's (or any user's) secret key faced in IBE systems, that is during honest registration every user samples its own public-secret key pair. However, the *embedding key-escrow problem* is still not directly prevented by the RBE abstraction. This is because a dishonest key accumulator could potentially add either certain trapdoors, or secretly register multiple keys for already registered users, or register any key for currently unregistered users. There could be many such scenarios in which malicious behavior of a key accumulator permits decryptability of ciphertexts intended towards arbitrary users by the key accumulator depending upon its adversarial strategy. Although such attack scenarios were not explicitly studied in the prior works [GHMR18, GHM+19], an extremely useful by-product of the approaches taken in those works was that the user registration process (and all the computations performed by the key accumulator) was completely deterministic. It thus leads to an extremely simple and elegant methodology for avoiding the embedding key-escrow problem by providing full public auditability. Basically, any user (or even a non-participating third party) could audit key accumulator and verify honest behavior by rebuilding the RBE public parameters and comparing that with the accumulated public parameters. As honestly generated public parameters do not have any trapdoors or faulty keys embedded by construction, thus public auditability solves the embedding key-escrow problem.

Although the above deterministic reconstructability feature of the RBE systems serves as a possible solution to embedding key-escrow problem, this is not at all efficient. Concretely, if any new (or even already registered) user wants to verify that the key accumulation has been done honestly, then that particular user needs to obtain a $O(n)$ (linear-sized) proof as well as spend $O(n)$ (linear amount of) time for verification, where n is the number of users registered until that point. In this work, we study the question of whether we can build RBE systems in which such verifications could be sped up. Specifically, we ask the following:

> *Do there exist efficient Registration-Based Encryption schemes in which any user can obtain short proofs of unique registration as well as short proofs of non-registration? Can such proof mechanisms be useful for speeding up the auditability process? Is it even possible to provide all such guarantees with only a* $\mathsf{poly}(\lambda, \log n)$ *cost incurred in the size of proof and running time of provers/verifiers?*

We answer the above questions in affirmative by introducing a notion of efficient verifiability for RBE systems and providing an instantiation from hash garbling schemes [GHMR18] thereby giving constructions based on standard assumptions (such as CDH, Factoring, LWE). Concretely, our contributions are described below.

Our results. In this work, we introduce a new notion for key accumulation which we call Verifiable Registration-Based Encryption (VRBE). Briefly, a VRBE system is simply a standard RBE system in which the key accumulator can also provide proofs of correct (and unique) registration for every registered user as well as proofs of non-registration for any yet unregistered identity. We give new constructions for VRBE from hash garbling schemes which provide succinct proofs of registration and non-registration, where the key accumulator can efficiently carry out the registration and proof generation processes. Our proof systems also naturally allow a much more efficient audit process which can be performed by any non-participating third party as well. A by-product of our approach is that we provide a more efficient RBE construction than that provided in the most recent work of Garg et al. [GHM+19], wherein the size of ciphertexts in

our construction is significantly smaller.[1] And, lastly we briefly discuss how the notion of VRBE can also be naturally extended to a wider range of access and trust structures, wherein the keys accumulated are no more associated with a PKE system, but for even more expressive encryption systems. Such systems might be practically more interesting in the future.

Next, we provide a detailed overview of our approach and describe the technical ideas. Later on, we discuss some related works.

1.1 Technical overview

We start by recalling the notion of RBE as it appears in prior works. We then discuss our proposed notions of efficient verifiability for such systems. Since the starting point of our construction is the RBE scheme proposed in [GHM+19], thus we first recall the main ideas and high-level structure of their approach. And, later we describe our construction and show how to provide succinct proofs of registration and non-registration for any user identity, thereby adding verifiability to the system.

The RBE abstraction. In an RBE system, there is a dedicated party which we call the key accumulator. A key accumulator runs the registration procedure indefinitely[2], where any user could make one of two types of queries — (1) registration query, where a new user sends in its identity and public key pair $(\mathsf{id}, \mathsf{pk})$ for registration, (2) update query, where an already registered user requests for supplementary key material u which is used for decryption. (The supplementary key material is usually referred to as the update information.) The key accumulator maintains the public parameters pp along with some auxiliary information aux throughout its execution. After each registration query of the form $(\mathsf{id}, \mathsf{pk})$, it updates the parameters to pp' and aux' to reflect addition and sends back the associated key material u to the corresponding user. For each update query made by a user with identity id, the accumulator extracts an update u from the auxiliary information aux and sends it over to the user. The encryption and decryption procedures are defined analogous to the IBE counterparts, except during decryption a user needs a piece of appropriate update information u to complete the operation.

At a high level, the correctness requirement states that any honestly registered user with identity id and key pair $(\mathsf{pk}, \mathsf{sk})$ must be able to decrypt a ciphertext encrypted for identity id under public parameters pp (which could have been updated after id was registered) using its own secret key sk as long as it also gets an update information u corresponding to pp from the accumulator. For efficiency, it is important that the size of public parameters pp, size of update information u, and the number of updates required by any user throughout its lifetime grow at most poly-logarithmically with the number of registered users n. Additionally, the registration process and update generation should run in time $\mathsf{poly}(\lambda, \log n)$. Lastly, for security, it is essential that a ciphertext encrypting a message m for identity id under parameters pp should hide the message as long as either id was not registered by the time pp was computed, or the key pair registered with identity id was honestly sampled and the corresponding secret key is unknown to an attacker.

Inadequacies of RBE and Workarounds. Now as we discussed before, the above abstraction still suffers from the embedding key-escrow problem. Specifically, the RBE system does not provide any abstraction for efficiently verifying whether a dishonest key accumulator — (1) secretly registers a public-secret key pair for any yet unregistered identity, (2) or while registering any new user (or even at any later point in time), also introduces a trapdoor (or register multiple keys for the same identity) that enables unauthorized decryption. For this specific reason, we study the possibility of efficient verifiability for RBE systems. Concretely, we consider two orthogonal notions of verifiability for an RBE scheme — *pre-registration* and *post-registration* proofs. Intuitively, the goal of pre-registration verifiability is to provide a short proof π validating that a given id has not yet been registered as per public parameters pp and any ciphertext ct encrypted towards

[1] Looking ahead, our efficiency gain is due to the fact that our construction takes a one-shot (single-step) approach whereas [GHM+19] takes a two-step approach. Here the outcome of a two-step approach is that the ciphertext consists of two layers of cascaded garbled circuits, while our solution consists of a single sequence of garbled circuits.

[2] It could run it sporadically as well, where it simply records all new registrations made in a certain time window, and later registers them all at once. For simplicity, here we consider the key accumulator is always online.

such an identity id will completely hide the plaintext even if all other secret keys are leaked. Similarly, the intuition behind post-registration verifiability is to provide a short proof π of *unique* accumulation, where the proof π guarantees that the key accumulator must not have added a trapdoor (or doubly registered) during a possibly dishonest registration which allows decryption of ciphertexts intended for that particular user. (Looking ahead, our formal definitions of pre/post-registration verifiability are stated in a much stronger way where we allow an adversary to completely control the key accumulator and still require the soundness/message-hiding property to hold.)

Defining Verifiable RBE. Formally, a verifiable RBE system is just like a regular RBE system with four additional (deterministic) algorithms — $\mathsf{PreProve}$, $\mathsf{PostProve}$, $\mathsf{PreVerify}$, and $\mathsf{PostVerify}$. The pre-registration prover takes as input a common reference string crs, public parameters pp, and a target identity id for which a proof π of pre-registration is provided. The post-registration prover on the other hand also takes a target public key pk as input. Both these provers are given random-access to the auxiliary information for time-efficient computation. Informally, the completeness of these proof systems states that the pre/post-registration verifier should always accept honestly generated proofs. And for soundness, the requirement is that if the pre-registration verifier accepts a proof π for an identity id w.r.t. parameters pp, then ciphertexts encrypted towards id must hide the message completely from a malicious key accumulator which computes the parameters pp and proof π. Similarly, for post-registration soundness, the property states that if the verifier accepts a proof π for identity-key pair $(\mathsf{id}, \mathsf{pk})$ w.r.t. parameters pp, then ciphertexts encrypted towards id must hide the message completely from a malicious key accumulator as long as the accumulator does not possess the secret key sk associated with the public key pk.

Stronger correctness guarantees. In addition to above-stated properties, we also define a very strong form of extractable correctness property for our post-registration proof. Concretely, the extractable correctness property states that there exists a deterministic update extraction algorithm such that if there exists an accepting post-registration proof π for identity-key pair $(\mathsf{id}, \mathsf{pk})$ w.r.t. parameters pp, then the extraction algorithm computes update information u from the proof π itself such that using update u, anybody could decrypt ciphertexts encrypted for identity id. Intuitively, extractable correctness states that completeness would still hold even for maliciously generated proofs. Our definitions are formally introduced later in Section 3.

A simple paradigm for efficient auditability. Looking back at our verifiability properties, one could interpret them as follows. The pre/post-registration proofs together help in ensuring that a key accumulator is behaving honestly at least *locally*. The idea behind a more global verification process is to perform the pre/post-registration verification on a randomly chosen (small) subset of users similar to what is done in probabilistically-checkable proof (PCP) literature. Specifically, suppose that a party claims that it has accumulated public parameters pp with the list of registered users R and non-registered users S. Any third party can efficiently audit the registration process by proceeding as follows — it samples a random subset of users in R and S, it requests post-registration and pre-registration proofs for users in those subsets respectively. If all the proofs are valid, then the auditor approves declaring that registration was done honestly. Note that depending upon the desired soundness threshold, the auditor can appropriately set the size of the subsets it samples. Thus, such randomized auditing would be more efficient than rebuilding the entire registration logs for most parameter regimes.

Reviewing prior RBE systems [GHMR18, GHM+19]. Before outlining our approach, we quickly recall the high-level structure used in prior works [GHMR18, GHM+19] since our construction uses similar building blocks. Let us first look at the weakly efficient RBE construction provided in [GHMR18] since it lays major groundwork for the follow-up works (including ours). At a high level, the ideas behind their construction can be summarized as follows. The key accumulator stores all the registered identity-key pairs $\{(\mathsf{id}_i, \mathsf{pk}_i)\}_{i\in[n]}$ using a shortlist of Merkle tree $\mathsf{Tree}_1, \mathsf{Tree}_2, \ldots, \mathsf{Tree}_k$, where every tree Tree_i is at least twice as large as Tree_{i+1}. Here the leaves of each tree encode one of the registered identity-key pairs $(\mathsf{id}, \mathsf{pk})$, while the internal nodes are like standard Merkle tree nodes (which is that they encode the hash of its children) except each node also stores the largest identity registered in its left sub-tree as well. In words, each tree

Tree_i is binary search Merkle tree, with all the leaves are lexicographically sorted as per the identities. Consequently, the public key of any registered identity can be obtained efficiently via a binary search, and the root values of each Merkle tree serve as a short commitment to the entire registry tree. To encrypt a message m to identity id, encryptor needs to search the Merkle trees to obtain id's public key pk. However, the public parameters only contain the root node, not the entire tree. To overcome this issue, the [GHMR18] construction uses the ideas developed in a long line of works [CDG+17, DG17a, DG17b, BLSV17, DGHM18] of deferring the binary search to the decryption side by sending a set of garbled circuits as part of the ciphertext. Basically, for decryption, a user needs to obtain an opening (i.e., path of nodes from root to leaf) in one of the merkle trees to its registered key, and this corresponds to the supplementary key material. Now, what makes the registration process only weakly efficient is that in order to register an identity-key pair $(\mathsf{id}, \mathsf{pk})$, the key accumulator creates a new tree consisting of only node $(\mathsf{id}, \mathsf{pk})$, and then merges all merkle trees of equal size. This helps in keeping the size of the public parameters short, but since the leaves of the merkle trees have to be sorted, thus tree merging process is quite inefficient which results in only a weakly efficient system.

In the follow-up work [GHM+19], the authors observed that the weakly efficient RBE construction described above is fully efficient if the identities being registered are already coming in sorted order. They call RBE schemes with these restrictions as Timed-RBE (T-RBE). Starting with this observation, they suggest a powerful two-step approach for building an efficient RBE system without this restriction, i.e. they provide a nice bootstrapping construction from T-RBE to general (non-timed) RBE with full efficiency. In their construction, the key accumulator associates every identity id with a timestamp t_{id} as well, where t_{id} is an internal counter incrementally maintained by the accumulator. The idea is that since timestamps t_{id} will be accumulated in a sorted order, thus for storing the association between the timestamp t_{id} and public key $\mathsf{pk}_{\mathsf{id}}$ one could simply use T-RBE scheme. And, for storing the association between identity id and timestamp t_{id}, the accumulator maintains a *balanced* merkle tree $\mathsf{TimeTree}$. The leaves of $\mathsf{TimeTree}$ encode the identity-timestamp pairs $(\mathsf{id}, t_{\mathsf{id}})$ for all registered users, and are sorted as per the identities. The most crucial aspect of $\mathsf{TimeTree}$ is that it is balanced (for instance, they use a red-black tree). Let us look into more details about how such an additional balanced merkle tree is useful in improving efficiency.

The key accumulator stores all the registered identity-timestamp pairs $\{(\mathsf{id}_i, i)\}_{i\in[n]}$ using a balanced merkle $\mathsf{TimeTree}$, and stores the timestamp-key association using a short list of (standard) merkle trees $\{\mathsf{Tree}_j\}_j$ as in [GHMR18]. The public parameters consists of multiple versions of the root node of the $\mathsf{TimeTree}$ along with the root nodes for $\{\mathsf{Tree}_j\}_j$. (Specifically, the public parameters store the root node and depth information of $\mathsf{TimeTree}$ for all timestamps whenever the underlying T-RBE merkle tree was updated.) To register an $(\mathsf{id}, \mathsf{pk})$ pair, the key accumulator inserts the identity-timestamp pair $(\mathsf{id}, t_{\mathsf{id}})$ into $\mathsf{TimeTree}$, and timestamp-key $(t_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}})$ to the sequence of T-RBE trees. The most important aspect of the construction is that if the T-RBE trees storing timestamp-key associations are merged, then the versions of root nodes being stored in the public parameters are updated as well. Next, let us look at how encryption and decryption are performed since the efficiency of registration follows almost immediately.

While encrypting message m for identity id, the encryptor now provides two levels of garbled circuit sequences, where the first level of garbled circuit sequence is used to find the timestamp t_{id} associated with id, and in next level one simply uses the T-RBE garbled circuit sequence to encrypt m under the corresponding public key pk. For building both levels of garbled circuit sequences, they employ the same approach of deferring binary search to decryption. The supplementary key material (or update) consists of two distinct paths, where the first path is w.r.t. the $\mathsf{TimeTree}$ and the second path is as per the T-RBE system which is w.r.t. one of the merkle trees in $\{\mathsf{Tree}_j\}_j$. The most important component of this extended construction is that in order to tightly bound the number of updates (for any user identity id), the first portion of key material/update u (required for evaluating the first level of garbled circuits) are only issued whenever the first T-RBE merkle tree in which identity id was registered gets merged. It turns out that executing the above idea formally leads to an efficient RBE scheme.

Our Verifiable RBE solution. The starting point of our construction is the [GHM+19] RBE scheme described above. As a first step, we start by simplifying their construction and present a one-shot (single-

step) approach to building efficient RBE systems. Later we describe how the simplified system can be made verifiable, both in pre-registration and post-registration settings, without making any additional assumptions. Lastly, we provide some comparisons and discuss potential black-box methods for making existing RBE schemes verifiable.

Although the basic principles behind our simplified construction and the one provided in [GHM+19] are quite similar, there are significant structural differences in both the approaches. Therefore, we provide a direct outline of our construction instead of going through the [GHM+19] construction and explaining the differences. Later on, we briefly compare our construction with theirs. Below we sketch the main ideas behind our construction. The actual construction is a little more complicated but follows quite naturally from the following outline. A detailed description appears later in Section 4.

In our construction, the key accumulator maintains a single *balanced* merkle tree which directly stores the mapping between identities and their respective public keys. Concretely, the key accumulator stores a balanced merkle tree which we call $\mathsf{EncTree}$ and it consists of two types of nodes — leaf and intermediate. Similar to existing works, a leaf node stores a identity-key pair $(\mathsf{id}, \mathsf{pk})$ for every registered identity, whereas an intermediate node stores a tuple of the form $(h_{\mathsf{left}}, \mathsf{id}, h_{\mathsf{right}})$, where h_{left} and h_{right} are hash values of its left and right child (respectively) and id is the largest identity in its left sub-tree. Since $\mathsf{EncTree}$ is balanced and the nodes are ordered as per the registered identities, therefore given an identity id the key accumulator could both efficiently search its associated public key (if id has been registered) and efficiently insert a new identity-key pair. The key accumulator stores $\mathsf{EncTree}$ as auxiliary information aux, and publishes root value rt and depth d of the tree as part of public parameters pp. The registration algorithm inserts given identity-key pair $(\mathsf{id}, \mathsf{pk})$ as a leaf in the $\mathsf{EncTree}$, balances the tree, and updates the hash values stored in all the ancestors of the newly inserted leaf. The registration algorithm then updates the public parameters pp to store the root value and depth of the updated $\mathsf{EncTree}$.

Encryption and Decryption. The encryption and decryption procedures follow the aforementioned 'deferred binary search' approach in which the ciphertext for identity id contains a sequence of d garbled circuits which work as follows. Given a path (a sequence of nodes from root to a leaf) in $\mathsf{EncTree}$ as input, the sequence of garbled circuits jointly check that the path is well-formed, and the leaf node encodes the identity id, and outputs a PKE ciphertext under the public key encoded in the leaf node. Individually, the i^{th} garbled circuit performs the local well-formedness check on the path and outputs the garbled input for $(i+1)^{th}$ garbled circuit. For decryption, the decryptor needs to obtain a valid path u from the accumulator which can be efficiently generated by the accumulator by performing a binary search on the $\mathsf{EncTree}$. Given a well-formed path, the decryptor can sequentially evaluate the garbled circuits and eventually obtain a PKE ciphertext which it decrypts using its secret key.

How to get the desired efficiency? The snapshotting trick. The above scheme is highly inefficient since updates must be issued each time a new user joins. At a very high level, we visualize our approach to improve efficiency as that of storing multiple *'snapshots'* of the registration process, where an older snapshot is deleted only after new user registration leads to a new snapshot that is used by as many number of users as those using the older snapshot.[3] The intuition is to split the registered user space into disjoint groups of sizes — $1, 2, 4, \ldots, 2^{\lambda}$. For each group size, the public parameters will consist of *at most* one snapshot which consists of root node and depth information of (a possibly older version of) the balanced merkle tree $\mathsf{EncTree}$.

Concretely, the public parameters look like $\{(j_1, \mathsf{snapshot}_{j_1}), \ldots, (j_\ell, \mathsf{snapshot}_{j_\ell})\}$ where every $j_i \in \{1, 2, \ldots, 2^{\lambda}\}$, and $j_i > j_{i+1}$, and $\mathsf{snapshot}_{j_i}$ consists of a root node and tree depth. These public parameters are interpreted as follows:

(1) the first j_1 users who registered refer to the $\mathsf{EncTree}$ corresponding to $\mathsf{snapshot}_{j_1}$ for decryption/obtaining update information;

[3]The snapshotting trick was implicitly used in [GHM+19] for similar reasons which is to build an *efficient* RBE scheme, but their construction instead highlighted the notion of explicitly mapping identities to corresponding timestamps as the more important aspect. Here we instead choose to focus mostly on the snapshotting principle since it is the major contributor in improving efficiency.

(2) next j_2 users who registered refer to the $\mathsf{EncTree}$ corresponding to $\mathsf{snapshot}_{j_2}$;

$\vdots$

(ℓ) and similarly the last j_ℓ users to register refer to the $\mathsf{EncTree}$ corresponding to $\mathsf{snapshot}_{j_\ell}$.

Basically, the key accumulator still adds new users to the single balanced merkle tree $\mathsf{EncTree}$ defined before, but now it also stores older snapshots of the $\mathsf{EncTree}$ (thereby older snapshots of the registration process). When a new user is added then a tuple $(1, \mathsf{snapshot})$ is added to list of parameters, where $\mathsf{snapshot}$ is the latest description of $\mathsf{EncTree}$. Now older snapshots get replaced with latest snapshots, after new user registration, if there exist two different snapshots but for same group size. By careful analysis and non-trivial execution of the above idea, we were able to show that the resulting RBE scheme is efficient. (Hereby non-trivial execution we mean that a straightforward implementation/black-box usage of balanced merkle trees lead to an RBE system which is only efficient in the amortized sense, but if the balanced merkle tree are *lazily* created then we obtain a fully efficient RBE scheme as desired. More details are provided in the main body.)

Making RBE Verifiable. It turns out that our simplified RBE construction is already very well suited for providing succinct proofs of pre/post-registration. This is due to the fact that the underlying technology being used is a merkle tree for which we know how to provide succinct proofs of membership, and since the merkle trees we are building are balanced and sorted, thus we also can provide succinct proofs of non-membership. Looking ahead, the proofs of pre-registration will consist of proofs of non-membership, and proofs of post-registration would be a combination of proofs of membership and non-membership.

Pre-Registration Proofs. For ease of exposition, consider that the public parameters contain exactly one root node and depth value (rt, d). The idea behind pre-registration proof readily extends to the general case when the public parameters contain more than one root node and depth value pairs. Recall that for soundness of pre-registration verifiability we need to argue that if the adversary produces an accepting pre-registration proof π for an identity id and parameters pp, then any ciphertext ct encrypted towards id under parameters pp must hide the plaintext. Now we know that in our construction, in order to decrypt such a ciphertext ct the adversary must be able to generate a *well-formed* path in the encryption tree $\mathsf{EncTree}$ such that the leaf node contains the identity id.

Here well-formedness of a path (a sequence of nodes from root to a leaf) is formally defined as follows. Let the path under consideration be $\mathsf{path} = (\mathsf{node}_1, \ldots, \mathsf{node}_d)$ where $\mathsf{node}_i = (h_{i,\mathsf{left}}, \mathsf{id}_i, h_{i,\mathsf{right}})$ for all i. We say path is well-formed if the following conditions are satisfied:

1. All the adjacent nodes obey the merkle tree hash constraints, i.e. either $h_{i,\mathsf{left}} = \mathsf{Hash}(\mathsf{hk}, \mathsf{node}_{i+1})$ or $h_{i,\mathsf{right}} = \mathsf{Hash}(\mathsf{hk}, \mathsf{node}_{i+1})$ for all i,

   (this also tells whether node_{i+1} is a left child or a right child of node_i)

2. If node_{i+1} is the left child of node_i, then it must be that $\mathsf{id}_j \leq \mathsf{id}_i$; otherwise $\mathsf{id}_j > \mathsf{id}_i$, (for all $j > i$)

3. Root rt is same as node_1.

Similarly, we define the notion of adjacent paths. For $b \in \{0, 1\}$, consider two paths $\mathsf{path}^{(b)} = (\mathsf{node}_1^{(b)}, \ldots, \mathsf{node}_d^{(b)})$ where $\mathsf{node}_i^{(b)} = (h_{i,\mathsf{left}}^{(b)}, \mathsf{id}_i^{(b)}, h_{i,\mathsf{right}}^{(b)})$. For two distinct paths $\mathsf{path}^{(0)}$ and $\mathsf{path}^{(1)}$, we say they are adjacent if the following conditions are satisfied:

1. Paths $\mathsf{path}^{(0)}$ and $\mathsf{path}^{(1)}$ are well-formed,

2. Nodes $\mathsf{node}_{k+1}^{(0)}$ and $\mathsf{node}_{k+1}^{(1)}$ are left and right child (respectively) of nodes $\mathsf{node}_k^{(0)}$ and $\mathsf{node}_k^{(1)}$

   (where k is the largest index such that first k nodes in paths $\mathsf{path}^{(0)}$ and $\mathsf{path}^{(1)}$ are identical)

3. For all $j > k + 1$, nodes $\mathsf{node}_j^{(0)}$ and $\mathsf{node}_j^{(1)}$ are right and left child of their respective parent nodes

   (where k is as defined above).

At this point, the pre-registration proofs follow from a natural observation which is that — if some identity id has not yet been registered as per the encryption tree $\mathsf{EncTree}$ (maintained by the key accumulator), then there must exist two identities $\mathsf{id}_{\mathsf{lwr}}$ and $\mathsf{id}_{\mathsf{upr}}$ such that $\mathsf{id}_{\mathsf{lwr}} < \mathsf{id} < \mathsf{id}_{\mathsf{upr}}$ and paths from the root node to leaf nodes containing $\mathsf{id}_{\mathsf{lwr}}$ and $\mathsf{id}_{\mathsf{upr}}$ are adjacent. That is, a pre-registration proof consists of two adjacent paths $\mathsf{path}_{\mathsf{lwr}}$ and $\mathsf{path}_{\mathsf{upr}}$ with identity relations as described above.[4] Now such proofs can be very efficiently computed by performing an extended binary search for id, where the extension corresponds to finding the closest registered identities both larger and smaller than id. Note that a verifier can perform the adjacency-check along with the check that the identities are arranged as $\mathsf{id}_{\mathsf{lwr}} < \mathsf{id} < \mathsf{id}_{\mathsf{upr}}$ for verifying the pre-registration proof.

In summary, the idea is that proof of pre-registration for an identity id can be provided using *structured* proofs of membership for two identities $\mathsf{id}_{\mathsf{lwr}}$ and $\mathsf{id}_{\mathsf{upr}}$, where the structure is formalized by the concept of adjacency as described above. The proof of soundness and correctness builds upon the aforementioned intuition and is provided in detail later in the main body.

Post-Registration Proofs. As in the case for pre-registration proofs, let us focus on the case where the public parameters contain a single root node and depth pair. Recall that an accepting post-registration proof π for identity-key pair $(\mathsf{id}, \mathsf{pk})$ w.r.t parameters pp must guarantee that a key accumulator uniquely added the identity-key pair $(\mathsf{id}, \mathsf{pk})$ to accumulated list of registered users. The post-registration proofs in our construction can also be visualized similar to the pre-registration proofs.

Specifically, observe that if some identity id has been registered as per the encryption tree $\mathsf{EncTree}$ (maintained by the key accumulator), then there must exist two identities $\mathsf{id}_{\mathsf{lwr}}$ and $\mathsf{id}_{\mathsf{upr}}$ such that $\mathsf{id}_{\mathsf{lwr}} < \mathsf{id} < \mathsf{id}_{\mathsf{upr}}$ and paths from the root node to leaf nodes containing $\mathsf{id}_{\mathsf{lwr}}$ and id, and id and $\mathsf{id}_{\mathsf{upr}}$ are adjacent. In other words, if identity id was uniquely registered, then there must exist three disjoint paths $\mathsf{path}_{\mathsf{lwr}}, \mathsf{path}_{\mathsf{mid}}$ and $\mathsf{path}_{\mathsf{upr}}$ such that $\mathsf{path}_{\mathsf{lwr}}, \mathsf{path}_{\mathsf{mid}}$ are adjacent as well as $\mathsf{path}_{\mathsf{mid}}, \mathsf{path}_{\mathsf{upr}}$ with the identities in their respective leaf nodes are related as described above.[5] As for pre-registration proofs, the aforementioned post-registration proof can be computed analogously in an efficient manner. The verification procedure can also be naturally extended from pre-registration proof.

There is however one important distinction in the case of post-registration proofs. Note that a pre-registration proof w.r.t. public parameters that contain multiple root node and depth pairs simply consist of independently computed pre-registration proofs for each root node and depth pair. This is because each sub-proof would guarantee that id was not registered as per that corresponding encryption tree snapshot. Thus, together all these sub-proof would guarantee that id was not registered as per any existing encryption tree snapshot. On the other hand, a post-registration proof w.r.t. public parameters with multiple root node and depth pairs will *not* consist of independently computed post-registration proofs for each root node and depth pair. This is because it is possible that the identity-key pair $(\mathsf{id}, \mathsf{pk})$ is registered as per only one root node and depth pair (say the latest snapshot), whereas it is not registered as per remaining (older) snapshots. Therefore, a post-registration proof, in this case, will consist of a mixture of post-registration and pre-registration proofs depending upon whether $(\mathsf{id}, \mathsf{pk})$ was registered as per that encryption tree snapshot.

A black-box approach to verifiability? A natural question a reader might ask is whether it would be possible to provide proofs of pre/post-registration verifiability generically for any RBE scheme by using a succinct non-interactive proof system such as SNARGs/SNARKs [Mic94, Gro10, GW11, Lip12, BC12] for instance. One possible approach along these lines could be to maintain an external sorted hash tree of registered identities, and for providing a pre/post-registration proof the accumulator would generate (non-)membership proofs for the hash tree along with a SNARK for proving the consistency of the external tree w.r.t. the RBE public parameters. Such a black-box approach seems possible, but would require maintaining additional data structures for consistency checks. More importantly, this approach necessitates making

[4] In case id is either smaller or larger than all registered identities, then the proof will consist of exactly one path instead of two. Here we ignore that for simplicity.

[5] As before, in case id is either the smallest or largest registered identity, then the proof will consist of exactly two paths instead of three. Here we ignore that for simplicity.

additional assumptions as for most succinct non-interactive proof systems we either need to make certain non-falsifiable assumptions [Nao03, GW11], or work in the Random Oracle model [Mic94, BR93]. Our construction and the proofs of verifiability do not rely on any extra assumptions other than what is already required in existing RBE systems [GHMR18, GHM+19] themselves, thus our results show that verifiability comes for free.

Also, note that SNARKs are usually defined for a family of circuits, thus the running time of prover is always as large as the size of the circuit, whereas in this case our provers already have random-access over the auxiliary information and we were able to provide highly efficient provers in which the running time of prover grows only poly-logarithmically with the number of users. Therefore, our non-black-box approach is more interesting both theoretically as well as practically, since we do not make any non-standard assumptions, nor do we incur an additional overhead in the efficiency.

1.2 Related Work and Future Directions

The problem of reducing the amount of trust put behind private key generators (PKGs) in an IBE environment has been well studied. Initial attempts were made by Boneh and Franklin [BF01] who suggested the use of multiple PKGs, instead of just one, with the goal of decentralizing the PKG thereby preventing PKGs from being a single target of corruption. This idea was further explored in many subsequent works ([CHSS02, LBD+04, PS08, KG10] to name a few). Later on, Goyal [Goy07] considered an alternate approach in which he studied the notion of accountable authority IBE. In such a system, the goal is to make the PKG accountable by requiring that any malicious activity by the PKG (which could involve distributing decryption keys to unauthorized users) can be traced back to the cheating authority. Goyal [Goy07] was able to show that Gentry's scheme [Gen06] was already an accountable IBE scheme, and additionally provided a different construction based on DBDH assumption. In follow-up work, [GLSW08] gave a construction of black-box accountable authority IBE from DBDH Assumption where tracing guarantee was much stronger.

Al-Riyami and Paterson [ARP03] put forward the notion of "Certificateless" Public Key Cryptography in which the key generation process is split between a central authority and the user. The authority first generates a key pair, where the private key is now the partial private key of the user. The remainder of the key is a random value generated by the user and is never revealed to anyone, not even the authority. A very crucial bottleneck in such systems is that an encryptor needs to obtain additional information about the user from the authority before encrypting a message. Building on this, [CCV04] proposed a similar setting and gave a construction based on [BF01] scheme.

Another line of work has been centered around trying to solve the key escrow problem by relying on anonymous IBE [BCOP04]. This approach was originally studied in [Cho09], where the intuition was that since in an anonymous IBE system ciphertexts hide the recipient identity, therefore key generation authority's ability to decrypt the ciphertext is adversely affected whenever there is some min-entropy in the space of identities (e.g. biometric identities). A similar approach was later taken in [WQT18] as well.

The most recent approach to solving the key escrow problem is of registration-based encryption (RBE) [GHMR18] which is the starting point of this work. The initial solutions [GHMR18] for RBE from standard assumptions could only provide a weak notion of efficiency, where the weakness was quantified using the running time of the key accumulator (entity responsible for user registration) which in the worst-case grows linearly with the number of registered users n. In a follow-up work [GHM+19], the authors were able to get around weak efficiency restriction, thereby giving the first fully efficient RBE scheme from standard assumptions. In [GHM+19] the authors also studied the anonymous version of RBE problem and were able to show interesting connections between anonymous RBE and secure messaging tasks. They were able to build a new primitive which they called anonymous board communication (ABC) generically from any anonymous RBE scheme. The ABC primitive proposed follows in the long line of work on practical scenarios of secure messaging [RMS18, CF10, CBM15, BSJ+17, AKTZ17, UDB+15, CCD+17, CB95].

In this work, we extend the notion of RBE by introducing the concept of efficient verifiability of registration. In addition to studying Verifiable RBE, we also propose an extension of RBE to a possibly more natural and broader setting which we call Universal RBE. We formally define it in Appendix A. At a high level, a URBE system is meant to extend the notion of registration-based encryption to more expressive encryption

systems as well. Informally, the idea is that a URBE system allows users to register public keys for more expressive encryption systems such as IBE, ABE, etc. It turns out that existing (V)RBE constructions very naturally extend to the setting of (V)URBE as well. We will elaborate more on this in the full version. Also, unlike the work of [GHM+19], we do not study blindness of our VRBE schemes. Although it seems that our VRBE construction might already satisfy the blindness properties as defined in [GHM+19], we chose to mainly focus on studying verifiability.

Lastly, throughout this paper we work in the common reference string (CRS) model, that is we assume that a hash key is sampled honestly by some trusted party and included as part of the common reference string crs. Thus, in our setting a malicious key accumulator is allowed to arbitrarily perform corruptions, except being allowed to choose the hash key (i.e., the crs). It is a very interesting question whether Verifiable RBE systems are possible in the standard model. At first glance, it might seem like Verifiable RBE in the standard model might be impossible and it might be possible to extend the impossibility results such as the impossibility of key-less collision-resistant hash functions, and SNARGS in the standard model [GW11] to the VRBE setting as well. But this needs more research and we leave it as an interesting open problem.

2 Preliminaries

Notations. Let PPT denote probabilistic polynomial-time. We denote the set of all positive integers upto n as $[n] := \{1, 2, \ldots, n\}$. Throughout this paper, unless specified, all polynomials we consider are positive polynomials. For any finite set S, $x \leftarrow S$ denotes a uniformly random element x from the set S. Similarly, for any distribution $\mathcal{D}$, $x \leftarrow \mathcal{D}$ denotes an element x drawn from distribution $\mathcal{D}$. The distribution $\mathcal{D}^n$ is used to represent a distribution over vectors of n components, where each component is drawn independently from the distribution $\mathcal{D}$. Additionally, whenever we define a summation of the form $\mathsf{sum}_i = \sum_{j \in [i]} f(j)$ for some function f, then we always define $\mathsf{sum}_0 = 0$. That is, summation over an empty set is denoted with 0.

2.1 Public Key Encryption

A public key encryption (PKE) scheme PKE for message spaces $\mathcal{M} = \{\mathcal{M}_\lambda\}_\lambda$ consists of the following polynomial-time algorithms.

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

$\mathsf{Enc}(\mathsf{pk}, m \in \mathcal{M}_\lambda) \to \mathsf{ct}$. The encryption algorithm takes as input a public key pk and a message m, and outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to m$. The decryption algorithm takes as input a secret key sk and a ciphertext ct, and outputs a message m.

Correctness. A PKE scheme $\mathcal{E}$ for message spaces $\mathcal{M}$ is said to be correct if for all λ, $m \in \mathcal{M}_\lambda$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$, and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$, we have that $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = m$.

Security. The standard security notion for PKE schemes is $\mathsf{IND\text{-}CPA}$ security. Formally, it is defined as follows.

Definition 2.1. *A public key encryption scheme* $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ *is IND-CPA secure if for every stateful PPT adversary* $\mathcal{A}$*, there exists a negligible function* $negl(\cdot)$*, such that the following holds*

$$\Pr\left[\mathcal{A}(\mathsf{ct}) = b \;:\; \begin{array}{c}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda); b \leftarrow \{0,1\} \\ (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk}); \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)\end{array}\right] \leq \frac{1}{2} + negl(\lambda).$$

2.2 Hash Garbling

We now review the notion of hash garbling scheme introduced in [GHMR18].

$\mathsf{Setup}(1^\lambda, 1^\ell) \to \mathsf{hk}$. The setup algorithm takes as input the security parameter λ, an input length parameter ℓ, and outputs a hash key hk.

$\mathsf{Hash}(\mathsf{hk}, x) \to y$. This is a deterministic algorithm that takes as input a hash key hk and a value $x \in \{0,1\}^\ell$ and outputs a value $y \in \{0,1\}^\lambda$.

$\mathsf{GarbleCkt}(\mathsf{hk}, C, \mathsf{state}) \to \tilde{C}$. It takes as input hash key hk, a circuit C, a secret state $\mathsf{state} \in \{0,1\}^\lambda$ and outputs a garbled circuit $\tilde{C}$.

$\mathsf{GarbleInp}(\mathsf{hk}, y, \mathsf{state}) \to \tilde{y}$. It takes as input hash key hk, a value $y \in \{0,1\}^\lambda$, a secret state $\mathsf{state} \in \{0,1\}^\lambda$ and outputs a garbled value $\tilde{y}$.

$\mathsf{Eval}(\tilde{C}, \tilde{y}, x) \to z$. This takes as input a garbled circuit $\tilde{C}$, a garbled value $\tilde{y}$, a value $x \in \{0,1\}^\ell$ and outputs a value z.

We now state the correctness and security requirements for a hash garbling scheme.

Correctness. A hash garbling scheme is said to be correct if for all $\lambda \in \mathbb{N}$, $\ell \in \mathbb{N}$, hash key $\mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$, circuit C, input $x \in \{0,1\}^\ell$, $\mathsf{state} \in \{0,1\}^\lambda$, garbled circuit $\tilde{C} \leftarrow \mathsf{GarbleCkt}(\mathsf{hk}, C, \mathsf{state})$ and a garbled value $\tilde{y} \leftarrow \mathsf{GarbleInp}(\mathsf{hk}, \mathsf{Hash}(\mathsf{hk}, x), \mathsf{state})$, we have $\mathsf{Eval}(\tilde{C}, \tilde{y}, x) = C(x)$.

Security. We now define the security requirement for hash garbling scheme.

Definition 2.2. *A hash garbling scheme is said to be secure if there exists a PPT simulator* Sim *such that for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for every $\lambda, \ell \in \mathbb{N}$, we have*

$$\Pr\left[\mathcal{A}(\tilde{C}_b, \tilde{y}_b) = b \;:\; \begin{array}{c} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell); (C, x) \leftarrow \mathcal{A}(\mathsf{hk}); \mathsf{state} \leftarrow \{0,1\}^\lambda \\ \tilde{C}_0 \leftarrow \mathsf{GarbleCkt}(\mathsf{hk}, C, \mathsf{state}); \tilde{y}_0 \leftarrow \mathsf{GarbleInp}(\mathsf{hk}, \mathsf{Hash}(\mathsf{hk}, x), \mathsf{state}) \\ (\tilde{C}_1, \tilde{y}_1) \leftarrow \mathsf{Sim}(\mathsf{hk}, x, 1^{|C|}, C(x)); b \leftarrow \{0,1\} \end{array}\right] \leq \frac{1}{2} + negl(\lambda).$$

3 Verifiable Registration Based Encryption

In this section, we define the notion of Verifiable Registration Based Encryption (VRBE). First, we recall the definition of Registration Based Encryption (RBE) as introduced in [GHMR18]. For message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_\lambda$ and identity space $\mathcal{ID} = \{\mathcal{ID}_\lambda\}_\lambda$, an RBE system consists of the following algorithms —

$\mathsf{CRSGen}(1^\lambda) \to \mathsf{crs}$. The CRS generation algorithm takes as input the security parameter λ, and outputs a common reference string crs.

$\mathsf{Gen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$. The key generation algorithm takes as input the security parameter 1^λ, and outputs a public-secret key pair $(\mathsf{pk}, \mathsf{sk})$. (Note that these are only public and secret keys, not the encryption/decryption keys.)

$\mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk}) \to \mathsf{pp}'$. The registration algorithm is a deterministic algorithm, that takes as input the common reference string crs, current public parameter pp, an identity id to be registered, and a corresponding public key pk. It maintains auxiliary information aux, and outputs the updated parameters pp'. The registration algorithm is modelled as a RAM program where it can read/write to arbitrary locations of the auxiliary information aux. (The system is initialized with pp and aux set to ϵ.)

$\mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, m) \to \mathsf{ct}$. The encryption algorithm takes as input the common reference string crs, public parameters pp, a recipient identity id, and a plaintext message m. It outputs a ciphertext ct.

$\mathsf{Upd}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{id}) \to u$. The key update algorithm is a deterministic algorithm, that takes as input the public parameters pp and an identity id. Given the auxiliary information aux, it generates the key update $u \in \{0,1\}^*$. Similar to the registration algorithm, this is also modelled as a RAM program, but it is only given read access to arbitrary locations of the auxiliary information aux.

$\mathsf{Dec}(\mathsf{sk}, u, \mathsf{ct}) \to m/\mathsf{GetUpd}/\perp$. The decryption algorithm takes as input a secret key sk, a key update u, and a ciphertext ct, and it outputs either a message $m \in \mathcal{M}$, or a special symbol in $\{\perp, \mathsf{GetUpd}\}$. (Here GetUpd indicates that a key update might be needed for decryption.)

Next, we introduce the notion of verifiability for an RBE system. Here we consider the notions of pre-registration as well as post-registration verifiability. Intuitively, the goal of pre-registration verifiability is to provide a short proof validating that a given id has not yet been registered and any ciphertext encrypted towards such an identity will completely hide the message even if all other secret keys are leaked. Similarly, the intuition behind post-registration verifiability is to provide a short proof of unique addition, where the proof guarantees that the key accumulator (i.e., the party responsible for registration) must not have added a trapdoor during a possibly dishonest registration which allows decryption of ciphertexts intended for that particular user. Formally, we introduce four new algorithms — $\mathsf{PreProve}, \mathsf{PreVerify}, \mathsf{PostProve}, \mathsf{PostVerify}$ with the following syntax:

$\mathsf{PreProve}^{\mathsf{aux}}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}) \to \pi$. The pre-registration prover algorithm is a deterministic algorithm, that takes as input the common reference string crs, public parameters pp, and an identity id. Given the auxiliary information aux, it outputs a pre-registration proof π. Similar to the registration algorithm, this is also modeled as a RAM program, but it is only given read access to arbitrary locations of the auxiliary information aux.

$\mathsf{PreVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \pi) \to 0/1$. The pre-registration verifier algorithm takes as input the common reference string crs, public parameter pp, an identity id, and a proof π. It outputs a single bit $0/1$ denoting whether the proof is accepted or not.

$\mathsf{PostProve}^{\mathsf{aux}}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk}) \to \pi$. The post-registration prover algorithm is a deterministic algorithm, that takes as input the common reference string crs, public parameters pp, an identity id, and a public key pk. Given the auxiliary information aux, it outputs a post-registration proof π. Similar to the registration algorithm, this is also modeled as a RAM program, but it is only given read access to arbitrary locations of the auxiliary information aux.

$\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk}, \pi) \to 0/1$. The post-registration verifier algorithm takes as input the common reference string crs, public parameter pp, an identity id, a public key pk, and a proof π. It outputs a single bit $0/1$ denoting whether the proof is accepted or not.

Note that if one does not impose any succinctness requirements on the pre/post-registration proofs, then the above algorithms are directly implied by the fact that the registration process is deterministic. This is because the proofs themselves can set to be the auxiliary information aux, and one could perform verification by simply rebuilding the public parameters given in aux. This is quite inefficient, thus we impose succinctness restrictions along with completeness and soundness restrictions on the pre/post-registration.

Remark 3.1. *In the above abstraction, we consider deterministic provers both for pre-registration as well as post-registration proofs. Although one could instead consider randomized proving algorithms, we avoid it since our construction already achieves deterministic proving and this also leads to simpler correctness and security definitions. Looking ahead, in all our security and correctness definitions, we do not provide the adversary any oracle queries to the* $\mathsf{PreProve}$ *and* $\mathsf{PostProve}$ *algorithms since they are deterministic given access to the auxiliary information* aux*. Since the adversary can itself maintain auxiliary information, thus due to proving algorithms being deterministic oracle queries are redundant.*

3.1 Correctness

Below we first recall the definition of completeness, compactness, and efficiency for RBE systems as studied previously. After that, we introduce the definitions for completeness, compactness, and efficiency of the pre/post-registration processes.

Definition 3.1 (Completeness, compactness, and efficiency of RBE)**.** *For any (stateful) interactive computationally unbounded adversary $\mathcal{A}$ that has a $\mathsf{poly}(\lambda)$ round complexity, consider the following game $\mathsf{Comp}^{\mathsf{RBE}}_{\mathcal{A}}(\lambda)$.*

1. *(Initialization) The challenger initializes parameters as $(\mathsf{pp}, \mathsf{aux}, u, S_{\mathsf{ID}}, \mathsf{id}^*, t) = (\epsilon, \epsilon, \epsilon, \emptyset, \bot, 0)$, samples $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$, and sends the crs to $\mathcal{A}$.*
2. *(Query Phase) $\mathcal{A}$ makes polynomially many queries of the following form, where each query is considered as a single round of interaction between the challenger and the adversary.*
   (a) ***Registering new (non-target) identity.*** *On a query of the form $(\mathsf{regnew}, \mathsf{id}, \mathsf{pk})$, the challenger checks that $\mathsf{id} \notin S_{\mathsf{ID}}$, and registers $(\mathsf{id}, \mathsf{pk})$ by running the registration procedure as $\mathsf{pp} := \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk})$. It adds id to the set as $S_{\mathsf{ID}} := S_{\mathsf{ID}} \cup \{\mathsf{id}\}$. (Note that the challenger updates the parameters $\mathsf{pp}, \mathsf{aux}, S_{\mathsf{ID}}$ after each query. Also, it aborts if the check fails.)*
   (b) ***Registering target identity.*** *On a query of the form $(\mathsf{regtgt}, \mathsf{id})$, the challenger first checks that $\mathsf{id}^* = \bot$. If the check fails, it aborts. Else, it sets $\mathsf{id}^* := \mathsf{id}$, samples challenge key pair $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{Gen}(1^\lambda)$, updates public parameters as $\mathsf{pp} := \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*)$, and sets $S_{\mathsf{ID}} := S_{\mathsf{ID}} \cup \{\mathsf{id}^*\}$. Finally, it sends the challenge public key pk^* to $\mathcal{A}$. (Here the challenger stores the secret key sk^* in addition to updating all other parameters. Also, note that the adversary here is restricted to make such a query at most once, since the challenger would abort otherwise.)*
   (c) ***Target identity encryptions.*** *On a query of the form (enctgt, m), the challenger checks if $\mathsf{id}^* \neq \bot$. It aborts if the check fails. Otherwise, it sets $t := t + 1$, $\widetilde{m}_t := m$, and computes ciphertext $\mathsf{ct}_t \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \widetilde{m}_t)$. It stores the tuple $(t, \widetilde{m}_t, \mathsf{ct}_t)$, and sends the ciphertext ct_t to $\mathcal{A}$.*[6]
   (d) ***Target identity decryptions.*** *On a query of the form (dectgt, j), the challenger checks if $\mathsf{id}^* \neq \bot$ and $j \in [t]$. It aborts if the check fails. Otherwise, it computes $y_j = \mathsf{Dec}(\mathsf{sk}^*, u, \mathsf{ct}_j)$. If $y_j = \mathsf{GetUpd}$, then it generates the fresh update $u := \mathsf{Upd}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{id}^*)$ and then re-computes $y_j = \mathsf{Dec}(\mathsf{sk}^*, u, \mathsf{ct}_j)$. Finally, the challenger stores the tuple (j, y_j).*
3. *(Output Phase) We say that the adversary $\mathcal{A}$ wins the game if there is some $j \in [t]$ for which $\widetilde{m}_j \neq y_j$.*

Let $n = |S_{\mathsf{ID}}|$ denote the number of identities registered until any specific round in the above game. We say that an RBE scheme is complete, compact, and efficient if for every (stateful) interactive computationally unbounded adversary $\mathcal{A}$ that has a $\mathsf{poly}(\lambda)$ round complexity, there exists polynomials p_1, p_2, p_3, p_4, p_5 and a negligible function $negl(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds:

Completeness. $\Pr[\mathcal{A}$ *wins in* $\mathsf{Comp}^{\mathsf{RBE}}_{\mathcal{A}}(\lambda)] \leq negl(\lambda)$.

Compactness of public parameters and updates. $|\mathsf{pp}| \leq p_1(\lambda, \log n)$ *and* $|u| \leq p_2(\lambda, \log n)$.

Efficiency of registration and update. *The running time of each invocation of* Reg *and* Upd *algorithms is at most $p_3(\lambda, \log n)$ and $p_4(\lambda, \log n)$, respectively. (Note that this implies the above compactness property.)*

Efficiency of the number of updates. *The total number of invocations of* Upd *for identity* id^* *during* target identity decryption phase *(i.e., Step* $\mathsf{2d}$ *of game* $\mathsf{Comp}^{\mathsf{RBE}}_{\mathcal{A}}(\lambda)$*) is at most $p_5(\lambda, \log n)$ for every n.*

[6] Here and throughout, whenever we write the challenger stores the tuple, we mean that it appends this to its local state such that these could be obtained by the challenger when referred to later in the game.

Next, we introduce the completeness, compactness, and efficiency conditions we require from the pre/post-registration procedures of a Verifiable RBE system. Briefly, the completeness of $(\mathsf{PreProve}, \mathsf{PreVerify})$ algorithms states that for any identity id^* that has not yet been registered, the key accumulator should be able to compute a proof π (by running the $\mathsf{PreProve}$ algorithm) such that proof π guarantees id^* has not yet been registered. Similarly for the post-registration verification, the completeness of $(\mathsf{PostProve}, \mathsf{PostVerify})$ algorithms states that for any identity id^* that has been (honestly) registered, the key accumulator should be able to compute a proof π (by running the $\mathsf{PostProve}$ algorithm) such that proof π guarantees id^* has been registered.

In addition to the above natural completeness definition for the post-registration verification, we also define a stronger completeness property that provides certain extractability guarantee. Informally, it states that if there exists a post-registration proof π for identity-key pair $(\mathsf{id}^*, \mathsf{pk}^*)$ that is accepted by the $\mathsf{PostVerify}$ algorithm, then every honestly generated ciphertext intended towards id^* can be decrypted by the corresponding secret key sk^* and some update u. Here the update u is (publicly, efficiently and deterministically) computable from the proof π itself, instead of the auxiliary information aux.

Definition 3.2 (Completeness, compactness, and efficiency of Pre/Post-Registration Proofs)**.** *For any (stateful) interactive computationally unbounded adversary $\mathcal{A}$ that has a $\mathsf{poly}(\lambda)$ round complexity, consider the following game $\mathsf{Comp}^{\mathsf{VRBE}}_{\mathcal{A}}(\lambda)$.*

1. *(Initialization) The challenger initializes parameters as $(\mathsf{pp}, \mathsf{aux}, S_{\mathsf{ID}}, S_{\mathsf{ID},\mathsf{PK}}, t) = (\epsilon, \epsilon, \emptyset, \emptyset, 0)$, samples $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$, and sends the crs to $\mathcal{A}$.*
2. *(Query Phase) $\mathcal{A}$ makes polynomially many queries of the following form, where each query is considered as a single round of interaction between the challenger and the adversary.*
   (a) ***Registering new identity.*** *On a query of the form $(\mathsf{regnew}, \mathsf{id}, \mathsf{pk})$, the challenger checks that $\mathsf{id} \notin S_{\mathsf{ID}}$, and registers $(\mathsf{id}, \mathsf{pk})$ by running the registration procedure as $\mathsf{pp} := \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk})$. It adds id and pk to the sets $S_{\mathsf{ID}}, S_{\mathsf{ID},\mathsf{PK}}$ as $S_{\mathsf{ID}} := S_{\mathsf{ID}} \cup \{\mathsf{id}\}$ and $S_{\mathsf{ID},\mathsf{PK}} := S_{\mathsf{ID},\mathsf{PK}} \cup \{(\mathsf{id}, \mathsf{pk})\}$.*
   (b) ***Pre-registration proofs.*** *On a query of the form $(\mathsf{prereg}, \mathsf{id})$, the challenger checks if $\mathsf{id} \notin S_{\mathsf{ID}}$. It aborts if the check fails. Otherwise, it sets $t := t + 1$, and computes the proof $\pi_t^{\mathsf{pre}} = \mathsf{PreProve}^{\mathsf{aux}}(\mathsf{crs}, \mathsf{pp}, \mathsf{id})$. Next, it verifies the proof π_t^{pre} as $\beta_t = \mathsf{PreVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \pi_t^{\mathsf{pre}})$, and stores the tuple $(\mathsf{prereg}, t, \beta_t)$.*
   (c) ***Post-registration proofs.*** *On a query of the form $(\mathsf{postreg}, \mathsf{id}, \mathsf{pk})$, the challenger checks if $(\mathsf{id}, \mathsf{pk}) \in S_{\mathsf{ID},\mathsf{PK}}$. It aborts if the check fails. Otherwise, it sets $t := t + 1$, and computes the proof $\pi_t^{\mathsf{post}} = \mathsf{PostProve}^{\mathsf{aux}}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk})$. Next, it verifies the proof π_t^{post} as $\beta_t = \mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk}, \pi_t^{\mathsf{post}})$, and stores the tuple $(\mathsf{postreg}, t, \beta_t)$.*
3. *(Output Phase) We say that the adversary $\mathcal{A}$ wins the game if for some $j \in [t]$ there exists a tuple of the form $(\mathsf{prereg}, j, 0)$ or $(\mathsf{postreg}, j, 0)$. (That is, there exists a proof that does not verify.)*

Let $n = |S_{\mathsf{ID}}|$ denote the number of identities registered until any specific round in the above game. We say that a VRBE scheme achieves pre/post-registration completeness, compactness, and efficiency if for every (stateful) interactive computationally unbounded adversary $\mathcal{A}$ that has a $\mathsf{poly}(\lambda)$ round complexity, there exists polynomials p_1, p_2, p_3, p_4, p_5 and a negligible function $negl(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds:

Completeness. $\Pr[\mathcal{A}$ *wins in* $\mathsf{Comp}^{\mathsf{VRBE}}_{\mathcal{A}}(\lambda)] \leq negl(\lambda)$.

Compactness of proofs. $|\pi_t^{\mathsf{pre}}| \leq p_1(\lambda, \log n)$ *and* $|\pi_t^{\mathsf{post}}| \leq p_2(\lambda, \log n)$.

Efficiency of provers. *The running time of each invocation of* $\mathsf{PreProve}$ *and* $\mathsf{PostProve}$ *algorithms is at most $p_3(\lambda, \log n)$ and $p_4(\lambda, \log n)$, respectively. (Note that this implies the above compactness property.)*

Next, we define the stronger extractable completeness for post-registration proofs.

Definition 3.3 (Efficiently Extractable Completeness for Post-Registration)**.** *A VRBE scheme satisfies efficiently extractable completeness property for post-registration if there exists a deterministic polynomial-time algorithm* UpdExt *such that for any computationally unbounded admissible adversary* $\mathcal{A}$*, there exists a negligible function* $negl(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*, we have*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, u, \mathsf{ct}) \neq m \ : \ \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda), (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ (\mathsf{pp}, \mathsf{id}, \pi, m) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pk}), u \leftarrow \mathsf{UpdExt}(\pi, \mathsf{id}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, m) \end{array}\right] \leq negl(\lambda)$$

where an adversary $\mathcal{A}$ *is said to be admissible if it always produces a valid post-registration proof* π *i.e.,* $\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk}, \pi) = 1$*.*

3.2 Security

Below we first recall the definition of security for RBE systems as studied previously. After that, we introduce the definitions for soundness of the pre/post-registration proofs.

Definition 3.4 (Message Hiding Security)**.** *For any (stateful) interactive PPT adversary* $\mathcal{A}$*, consider the following game* $\mathsf{Sec}^{\mathsf{RBE}}_{\mathcal{A}}(\lambda)$*.*

1. *(Initialization) The challenger initializes parameters as* $(\mathsf{pp}, \mathsf{aux}, u, S_{\mathsf{ID}}, \mathsf{id}^*) = (\epsilon, \epsilon, \epsilon, \emptyset, \perp)$*, samples* $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$*, and sends the* crs *to* $\mathcal{A}$*.*
2. *(Query Phase)* $\mathcal{A}$ *makes polynomially many queries of the following form:*
   (a) ***Registering new (non-target) identity.*** *On a query of the form* $(\mathsf{regnew}, \mathsf{id}, \mathsf{pk})$*, the challenger checks that* $\mathsf{id} \notin S_{\mathsf{ID}}$*, and registers* $(\mathsf{id}, \mathsf{pk})$ *by running the registration procedure as* $\mathsf{pp} := \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk})$*. It adds* id *to the set as* $S_{\mathsf{ID}} := S_{\mathsf{ID}} \cup \{\mathsf{id}\}$*.*
   (b) ***Registering target identity.*** *On a query of the form* $(\mathsf{regtgt}, \mathsf{id})$*, the challenger first checks that* $\mathsf{id}^* = \perp$*. If the check fails, it aborts. Else, it sets* $\mathsf{id}^* := \mathsf{id}$*, samples challenge key pair* $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{Gen}(1^\lambda)$*, updates public parameters as* $\mathsf{pp} := \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*)$*, and sets* $S_{\mathsf{ID}} := S_{\mathsf{ID}} \cup \{\mathsf{id}^*\}$*. Finally, it sends the challenge public key* pk^* *to* $\mathcal{A}$*.*
3. *(Challenge Phase) On a query of the form* $(\mathsf{chal}, \mathsf{id}, m_0, m_1)$*, then the challenger checks if* $\mathsf{id} \notin S_{\mathsf{ID}} \backslash \{\mathsf{id}^*\}$*. It aborts if the check fails. Otherwise, it samples a bit* $b \leftarrow \{0, 1\}$ *and computes challenge ciphertext* $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, m_b)$*.*
4. *(Output Phase) The adversary* $\mathcal{A}$ *outputs a bit* b' *and wins the game if* $b' = b$*.*

We say that an RBE scheme is message-hiding secure if for every (stateful) interactive PPT adversary $\mathcal{A}$*, there exists a negligible function* $negl(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*,* $\Pr[\mathcal{A}$ *wins in* $\mathsf{Sec}^{\mathsf{RBE}}_{\mathcal{A}}(\lambda)] \leq \frac{1}{2} + negl(\lambda)$*.*

Finally, we define the soundness requirements for our pre/post-registration proof systems. Informally, the pre-registration soundness states that any adversarial key accumulator must not be able to simultaneously — 1) provide a valid (acceptable) proof of pre-registration for some identity id, 2) able to break semantic security for (honestly generated) ciphertexts intended towards identity id. Intuitively, this says that even a corrupt key accumulator must not be able to decrypt ciphertexts intended for unregistered users while being able to provide an accepting pre-registration proof. Thus, any new user can ask for a pre-registration proof to verify that the key accumulator has not inserted any trapdoor that enables the accumulator to decrypt ciphertexts encrypted for that user.

In a similar vein, the post-registration soundness informally states that any adversarial key accumulator must not be able to simultaneously — 1) provide a valid (acceptable) proof of post-registration for some identity-key pair $(\mathsf{id}, \mathsf{pk})$ (where pk has honestly generated and the associated secret key was not revealed), 2) able to break semantic security for (honestly generated) ciphertexts intended towards identity id. Intuitively, this says that even a corrupt key accumulator must not be able to decrypt ciphertexts intended for registered

users while being able to provide an accepting post-registration proof. Thus, any registered user can ask for a post-registration proof to verify that the key accumulator has not inserted any trapdoor that enables the accumulator to decrypt ciphertexts encrypted for that user. Now we give the formal definitions.

Definition 3.5 (Soundness of Pre-Registration Verifiability)**.** *A VRBE scheme satisfies soundness of pre-registration verifiability if for every stateful* admissible *PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds*

$$\Pr\left[\mathcal{A}(\mathsf{ct}) = b \ : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda) \\ (\mathsf{pp}, \mathsf{id}, \pi, m_0, m_1) \leftarrow \mathcal{A}(\mathsf{crs}) \\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, m_b) \end{array}\right] \leq \frac{1}{2} + negl(\lambda),$$

where $\mathcal{A}$ is admissible if and only if π is a valid pre-registration proof, i.e. $\mathsf{PreVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \pi) = 1$.

Definition 3.6 (Soundness of Post-Registration Verifiability)**.** *A VRBE scheme satisfies soundness of post-registration verifiability if for every stateful* admissible *PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds*

$$\Pr\left[\mathcal{A}(\mathsf{ct}) = b \ : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda);\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ (\mathsf{pp}, \mathsf{id}, \pi, m_0, m_1) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pk}) \\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, m_b) \end{array}\right] \leq \frac{1}{2} + negl(\lambda),$$

where $\mathcal{A}$ is admissible if and only if π is a valid post-registration proof, i.e. $\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk}, \pi) = 1$.

4 Verifiable RBE from Standard Assumptions

In this section, we present our VRBE construction. Our construction relies on two primitives — a regular PKE scheme $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$, and a hash garbling scheme $\mathsf{HG} = (\mathsf{HG.Setup}, \mathsf{HG.Hash}, \mathsf{HG.GarbleCkt}, \mathsf{HG.GarbleInp}, \mathsf{HG.Eval})$. Below we provide a detailed outline of our construction.

4.1 Construction

For ease of exposition, we assume that the length of identities supported, length of public keys generated by Gen algorithm, the output length of the hash is λ-bits, and the input length of the hash function is $(3\lambda + 1)$-bits. Note that this can be avoided by simply selecting parameters accordingly. Below we define some useful notation that we will reuse throughout the sequel. Additionally, we describe how to interpret the auxiliary information and the public parameters in our construction.

Abstractions, Trees, and Notations. In our construction, the key accumulator maintains two types of balanced binary trees. The first tree which we refer to as the IDTree is a balanced binary tree in which each node has a label of the form $(\mathsf{id}, t) \in \{0,1\}^{2\lambda}$, and the nodes are basically being sorted as per the first tuple entry which is id. (Concretely, $(\mathsf{id}_1, t_1) \prec (\mathsf{id}_2, t_2)$ iff $\mathsf{id}_1 < \mathsf{id}_2$, where $\prec$ denotes the node ordering.) This tree is simply used as an internal storage object (which provides fast node insertion/lookup) by the key accumulator. Here id denotes the registered identity and t denotes the timestamp (i.e., number of users already registered $+1$).

The second family of trees which we refer to are the *encryption* trees $\{\mathsf{EncTree}_i\}_{i \in [\ell_n]}$ for some $\ell_n > 0$. Each such tree consists of two-types of nodes — (1) leaf nodes which store a registered identity-key pair, (2) non-leaf nodes which store the hash values of its children and largest registered identity in its left sub-tree. Concretely, each node in the tree has a label of the form $(\mathsf{flag}||a||\mathsf{id}||b) \in \{0,1\}^{3\lambda+1}$. For a leaf node $\mathsf{flag} = 1$, $a = 0^\lambda$, $b = \mathsf{pk}$ and $(\mathsf{id}, \mathsf{pk})$ is identity-key pair of the corresponding registered user. For a non-leaf node, $\mathsf{flag} = 0$, and id denotes the largest registered identity in its left sub-tree, a and b are the hash value of its left and right child's label (respectively). The leaf nodes are inserted as per their registered identity (i.e., the nodes are ordered with an increasing ordering amongst the identities). Concretely, a new leaf node $(1||0^\lambda||\mathsf{id}||\mathsf{pk})$ is added as follows —

1. Perform a binary search, by using the 'largest registered identity in the left sub-tree' information stored in the label of each intermediate node, to find the leaf node with the smallest identity $\widetilde{\mathsf{id}}$ such that $\widetilde{\mathsf{id}} > \mathsf{id}$. (Let $\widetilde{\mathsf{pk}}$ be the key associated with $\widetilde{\mathsf{id}}$.)

2. Delete the leaf node associated with $\widetilde{\mathsf{id}}$, and replace it with a new intermediate node such that $(1||0^\lambda||\mathsf{id}||\mathsf{pk})$ and $(1||0^\lambda||\widetilde{\mathsf{id}}||\widetilde{\mathsf{pk}})$ are its left and right children (respectively).

3. Perform the *re-balance* operation on the binary tree.[7]

4. Re-compute the labels for all intermediate nodes which have been re-balanced (i.e., moved around). This involves updating the largest registered identity in the left sub-tree information as well as re-computing the corresponding hash values.

Looking ahead, here the first $\ell_n - 1$ encryption trees $\mathsf{EncTree}_1, \ldots, \mathsf{EncTree}_{\ell_n - 1}$ represent the older snapshots of the registration process, whereas $\mathsf{EncTree}_{\ell_n}$ represents the latest encryption tree which contains all the identities registered so far. Also, the above tree insertion operation is efficient ($O(\log n)$ updates and running time) as long as the underlying tree abstraction provides efficient lookup and insertion. Since we use a balanced tree as the underlying abstraction, thus efficiency follows.

A very useful piece of notation in our scheme is the notion of *'paths'* from the root node to a leaf node in some encryption tree $\mathsf{EncTree}$. Concretely, throughout this section, we will define a *path* w.r.t. a tree $\mathsf{EncTree}$ (with root rt and depth d) as a sequence of (at most) d nodes where the first node is the root node of the tree and last node is a leaf node with certain specific properties. Concretely, any path path will look like $\mathsf{path} = (\mathsf{node}_1, \ldots, \mathsf{node}_{d-1}, \mathsf{node}_d)$, where for $i < d$, $\mathsf{node}_i = (0||a_i||\mathsf{id}_i||b_i)$ for some hash values a_i, b_i and identity id_i. Similarly, $\mathsf{node}_d = (1||0^\lambda||\mathsf{id}_d||\mathsf{pk})$ for some identity-key pair $\mathsf{id}_d, \mathsf{pk}$, and the remaining intermediate nodes are such that for every i, $a_i = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i+1})$ if node_{i+1} is left child of node_i, else $b_i = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i+1})$. Also, if node_{i+1} is left child of node_i then $\mathsf{id}_i \geq \mathsf{id}_{i+1}$, else $\mathsf{id}_i < \mathsf{id}_{i+1}$. Now note that such a path can be efficiently computed for every identity id, which has been added to encryption tree $\mathsf{EncTree}$, by simply performing an extended binary search. We will be re-using this fact many times throughout the sequel.

Lastly, we define a notion which we refer to as *'adjacent'* paths. This is extremely useful for verifiability of our scheme. Note that if during binary search in any balanced search tree, if the node/label that is being searched does not exist, then one could prove that efficiently by giving two paths to nodes with labels that are just bigger than and smaller than the label as per the ordering defined in the tree. More formally, for any two paths path_1 and path_2 in an encryption tree, we can perform an adjacency check efficiently as follows. Let $\mathsf{path}_j = (\mathsf{node}_{1,j}, \ldots, \mathsf{node}_{d-1,j}, \mathsf{node}_{d,j})$ for $j \in [2]$ where $\mathsf{node}_{i,j} = (0||a_{i,j}||\mathsf{id}_{i,j}||b_{i,j})$ for $j < d$ and $\mathsf{node}_{d,j} = (1||0^\lambda||\mathsf{id}_{d,j}||\mathsf{pk}_{d,j})$: (1) First, check that both paths are valid. Note that path validity is checked as that either $\mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i+1,j})$ is equal to $a_{i,j}$ or $b_{i,j}$.[8] (2) Next, the verifier first computes the largest common prefix of nodes in paths path_1 and path_2. That is, let k be the largest index such that $\mathsf{node}_{i,1} = \mathsf{node}_{i,2}$ for all $i \leq k$. Now if $\mathsf{id}_{d,1} < \mathsf{id}_{d,2}$, then check that $\mathsf{node}_{k+1,1}$ and $\mathsf{node}_{k+1,2}$ are left and right children of $\mathsf{node}_{k,1} = \mathsf{node}_{k,2}$. Next, it must check that, for all $i > k+1$, $\mathsf{node}_{i,1}$ is always the right child of its parent and $\mathsf{node}_{i,2}$ is always the left child of its parent. Basically, this is done to make sure that these two paths are adjacent and there does not exist any intermediate registered identity between these.

Construction. The key accumulator initializes the public parameters pp and auxiliary information aux as empty strings ϵ. And, afterwards at any point, the auxiliary information will contain the IDTree and (at most a λ number of) encryption trees $\mathsf{EncTree}_i$ along with a number n_i.[9] And, the public parameters pp consists

[7] Note that the tree re-balancing operation has to be carefully performed as in our abstraction (as well as the [GHM+19] abstraction) the leaf nodes and intermediate nodes are not exchangeable. Thus, the leaf-nodes must always stay the leaf nodes. Roughly one might consider that the re-balancing operation is only performed on the tree obtained by removing all leaf-nodes. This is not completely accurate but captures the underlying intuition.

[8] Note that this also tells whether $\mathsf{node}_{i+1,j}$ is a left child of $\mathsf{node}_{i,j}$, or right child.

[9] Looking ahead, the number n_i signifies the number of users who will refer to the tree $\mathsf{EncTree}_i$ for decryption. The significance of n_i will become clear in the construction.

of root value and depth pairs (rt_i, d_i) for each encryption tree $\mathsf{EncTree}_i$ present in auxiliary information aux. Here rt_i is the root node and d_i is the depth of $\mathsf{EncTree}_i$. We now formally describe our construction.

$\mathsf{CRSGen}(1^\lambda) \to \mathsf{crs}$. The CRS generation algorithm samples a hash key for the hash garbling scheme as $\mathsf{hk} \leftarrow \mathsf{HG.Setup}(1^\lambda, 1^{3\lambda+1})$, and outputs $\mathsf{crs} = \mathsf{hk}$.

$\mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk}) \to \mathsf{pp}'$. Let $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_{i\in[\ell_n]}$ and $\mathsf{aux} = \Big(\mathsf{IDTree}, \{(\mathsf{EncTree}_i, n_i)\}_{i\in[\ell_n]}\Big)$. Also, let $n = \sum_i n_i + 1$. The key accumulator performs the following operations:

1. It creates a leaf node with the label $(1||0^\lambda||\mathsf{id}\ ||\mathsf{pk})$, and update the current (latest) encryption tree $\mathsf{EncTree}_{\ell_n}$ by inserting the new leaf node. (Note that the insertion is performed as described above, and it involves balancing the tree and updating the hash values accordingly.)
2. Let $\mathsf{NewTree}$ be the new encryption tree. It continues by adding (id, n) to the IDTree, and the tuple $(\mathsf{EncTree}_{\ell_n+1}, 1) := (\mathsf{NewTree}, 1)$ to current auxiliary information aux. (This new tuple should be interpreted as signifying that only one user (which is the current, i.e. n^{th}, user with identity id) would refer to the latest encryption tree $\mathsf{NewTree}$ during decryption.)
3. Next it modifies the list of encryption trees as follows. Let $\mathsf{aux} = \Big(\mathsf{IDTree}, \{(\mathsf{EncTree}_i, n_i)\}_{i\in[\ell_n+1]}\Big)$, and
$$\delta = \max\left(\{0\} \cup \left\{i \in [\ell_n - 1] \ :\ \forall\ j \in [i],\ n_{\ell_n+1-j} = 2^{j-1}\right\}\right).$$
It modifes the auxilliary information as $\mathsf{aux} = \Big(\mathsf{IDTree}, \big\{(\mathsf{EncTree}'_i, n'_i)\big\}_{i\in[\ell_n+1-\delta]}\Big)$, where
$$(\mathsf{EncTree}'_i, n'_i) := \begin{cases} (\mathsf{EncTree}_i, n_i) & \text{if } i < \ell_n + 1 - \delta, \\ (\mathsf{NewTree}, 2 \cdot n_i) & \text{otherwise.} \end{cases}$$
In words, the accumulator removes all the old versions of the encryption trees as long as it could replace all of them with the latest tree until the number of users which would then refer to the latest tree stays a power of 2. To illustrate this operation, we give a detailed running example of the Reg algorithm in Figure 1.
4. Lastly, the accumulator modifes the public parameters to $\mathsf{pp}' = \{(\mathsf{rt}'_i, d'_i)\}_{i\in[\ell_n+1-\delta]}$, where rt'_i, d'_i are root node and depth of the encryption tree $\mathsf{EncTree}'_i$ (respectively).
Note. At a high level, the accumulator maintains the invariant that the i^{th} encryption tree $\mathsf{EncTree}'_i$ is an accumulation of the identity-key pairs for exactly the first $\sum_{j\le i} n'_j$, and this tree is intended to be precisely used during decryption by those n'_i users who registered just after the first $\sum_{j\le i-1} n'_i$ users. Additionally, the n_i values for the last and the second last encryption trees are more than a factor of 2 apart. (The last point is quite crucial in ensuring that number of updates grows only logarithmically.)

$\mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, m) \to \mathsf{ct}$. Let $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_{i\in[\ell_n]}$ and $\mathsf{crs} = \mathsf{hk}$. The encryptor proceeds as follows:

1. First, it samples $\mathsf{state}_{i,j} \leftarrow \{0,1\}^\lambda$ and $r_{i,j} \leftarrow \{0,1\}^\lambda$ for each $i \in [\ell_n]$, and $j \in [d_i + 1]$.
2. Next, for each encryption tree $\mathsf{EncTree}_i$, it computes a sequence of d_i hash-garbled circuits as follows:
For $i \in [\ell_n]$:
   — For $j \in [d_i]$: It constructs a step-circuit $\mathsf{Enc\text{-}Step}_{i,j}$ as defined in Fig. 2 with $\mathsf{hk}, \mathsf{id}, m$, $\mathsf{state}_{i,j+1}$, $r_{i,j+1}$ hardwired. It then garbles the circuit as $\widetilde{\mathsf{Enc\text{-}Step}}_{i,j} \leftarrow \mathsf{HG.GarbleCkt}(\mathsf{hk}, \mathsf{Enc\text{-}Step}_{i,j}, \mathsf{state}_{i,j})$.
   — It computes the hash value of root node as $h_i = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{rt}_i)$, and computes the input garbling as $\widetilde{y}_{i,1} = \mathsf{HG.GarbleInp}(\mathsf{hk}, h_i, \mathsf{state}_{i,1}; r_{i,1})$.

Sample Execution of Reg Algorithm

Consider the scenario where 7 users ($\mathsf{id}_1, \mathsf{id}_2, \mathsf{id}_3, \mathsf{id}_4, \mathsf{id}_5, \mathsf{id}_6, \mathsf{id}_7$) are registered into the system. The auxilliary information aux now stores IDTree and 3 versions of $\mathsf{EncTree}$. IDTree consists of all the identites along with their timestamps. $\mathsf{EncTree}_1, \mathsf{EncTree}_2$ are the versions of $\mathsf{EncTree}$ when only 4 users and 6 users were registered respectively. $\mathsf{EncTree}_3$ is the latest version of the $\mathsf{EncTree}$ when all 7 users are registered in the system. More precisely, the list of identities present in each $\mathsf{EncTree}_i$ is as follows.

$$\mathsf{aux} = \{\mathsf{IDTree}, (\mathsf{EncTree}_1, 4) : [\mathsf{id}_1, \ldots, \mathsf{id}_4],\ (\mathsf{EncTree}_2, 2) : [\mathsf{id}_1, \ldots, \mathsf{id}_6],\ (\mathsf{EncTree}_3, 1) : [\mathsf{id}_1, \ldots, \mathsf{id}_7]\}$$

Let us now look at when we register a new identity id_8. The key accumulator sets $n = 8$, inserts id_8 into IDTree, creates $\mathsf{NewTree}$ by inserting id_8 into $\mathsf{EncTree}_3$, and sets $(\mathsf{EncTree}_4, 1) = (\mathsf{NewTree}, 1)$. To compute δ, the key accumulator observes that $n_{\ell_n+1-j} = n_{4-j} = 2^{j-1}$ for all $j \in [3]$, and sets $\delta = 3$. The key accumulator now deletes $\mathsf{EncTree}_i$ for each $i \geq \ell_n + 1 - \delta = 1$, and sets $(\mathsf{EncTree}'_1, n'_1) = (\mathsf{NewTree}, 2 \cdot n_1 = 8)$. So, now the updated auxilliary information is $\mathsf{aux} = \{\mathsf{IDTree}, (\mathsf{EncTree}'_1, 8) : [\mathsf{id}_1, \ldots, \mathsf{id}_8]\}$.

Figure 1: An example demonstrating aux being updated during registration

Circuit $\mathsf{Enc\text{-}Step}_{i,j}$

Constants: $\mathsf{hk}, \mathsf{id}, m, \mathsf{state}_{i,j+1}, r_{i,j+1}$.
Input: $\mathsf{flag}||a||\mathsf{id}^*||b \in \{0,1\}^{3\lambda+1}$.

1. If $\mathsf{flag} = 1$ and $\mathsf{id}^* = \mathsf{id}$, output $1||\mathsf{PKE.Enc}(b, m; r_{i,j+1})$.
2. If $\mathsf{flag} = 1$ and $\mathsf{id}^* \neq \mathsf{id}$, output $1||\perp$.
3. If $\mathsf{id} > \mathsf{id}^*$, output $0||\mathsf{HG.GarbleInp}(\mathsf{hk}, b, \mathsf{state}_{i,j+1}; r_{i,j+1})$
   Else, output $0||\mathsf{HG.GarbleInp}(\mathsf{hk}, a, \mathsf{state}_{i,j+1}; r_{i,j+1})$.

Figure 2: Description of the step-circuit $\mathsf{Enc\text{-}Step}_{i,j}$

3. Finally, it outputs the ciphertext ct as $\mathsf{ct} = \left(\{(\mathsf{rt}_i, d_i)\}_i, \left\{\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}\right\}_{i,j}, \{\widetilde{y}_{i,1}\}_i\right)$.

$\mathsf{Upd}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{id}) \to u$. Let $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_{i \in [\ell_n]}$ and $\mathsf{aux} = \left(\mathsf{IDTree}, \{(\mathsf{EncTree}_i, n_i)\}_{i \in [\ell_n]}\right)$. The update computation is a two-step approach. In the first step, the algorithm performs a binary search over the IDTree to obtain the timestamp associated with the identity id. As IDTree is a balanced binary search tree, thus this can done efficiently. Let t be the timestamp associated with id that the binary search outputs. (It aborts if no such timestamp exists.) In the second phase, the update generator computes the index $i^* \in [\ell_n]$ such that $\sum_{j \in [i^*-1]} n_j < t \leq \sum_{j \in [i^*]} n_j$. Index i^* corresponds to the smallest index of the encryption tree in which id has been registered. Now the algorithm performs a binary search for identity id in the encryption tree $\mathsf{EncTree}_{i^*}$. It stores the path of nodes traversed from root rt_{i^*} to leaf node containing identity id. Let path be the searched path in tree $\mathsf{EncTree}_{i^*}$. Finally, it outputs the update u as $u = \mathsf{path}$. (Again, it aborts if no such index or a path to a leaf node containing identity id exists.)

$\mathsf{Dec}(\mathsf{sk}, u, \mathsf{ct}) \to m/\perp/\mathsf{GetUpd}$. The decryption algorithm first parses the inputs as:

$$\mathsf{ct} = \left(\{(\mathsf{rt}_i, d_i)\}_i, \left\{\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}\right\}_{i,j}, \{\widetilde{y}_{i,1}\}_i\right), \text{ and } u = \mathsf{path} = (\mathsf{node}_1, \ldots, \mathsf{node}_{d-1}, \mathsf{node}_d).$$

It then proceeds as follows:

1. Let i be the smallest index $i \in [\ell_n]$ such that $\mathsf{node}_1 = \mathsf{rt}_i$. If such an i does not exist, then it outputs GetUpd. Otherwise, it continues.
2. Now the decryptor iteratively runs the hash garbling evaluation algorithms as follows.
   For $j \in [d_i]$:

— It evaluates the j^{th} step-circuit as $(\mathsf{flag}||\widetilde{y}_{i,j+1}) \leftarrow \mathsf{HG.Eval}(\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}, \widetilde{y}_{i,j}, \mathsf{node}_i)$.

— If $\mathsf{flag} = 1$ and $\widetilde{y}_{i,j+1} = \perp$, the algorithm outputs $\perp$.

— Otherwise, if $\mathsf{flag} = 1$ and $\widetilde{y}_{i,j+1} \neq \perp$, then interpret $\widetilde{y}_{i,j+1}$ as a PKE ciphertext, and decrypt it as $\widetilde{y}_{i,j+1}$ using key sk to obtain the message as $m \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, \widetilde{y}_{i,j+1})$. And, it outputs the message m.

3. If the algorithm did not terminate, then it outputs $\perp$.

$\mathsf{PreProve}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{id}) \rightarrow \pi$. Let $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_{i\in[\ell_n]}$ and $\mathsf{aux} = \left(\mathsf{IDTree}, \{(\mathsf{EncTree}_i, n_i)\}_{i\in[\ell_n]}\right)$. The pre-registration proof consists of ℓ_n sub-proofs π_i for $i \in [\ell_n]$, where each sub-proof π_i consist of two[10] adjacent paths in the i^{th} encryption tree $\mathsf{EncTree}_i$. Concretely, the algorithm proceeds as follows:

For $i \in [\ell_n]$:

— It runs a binary search on tree $\mathsf{EncTree}_i$ to find identity id. If id is contained in $\mathsf{EncTree}_i$, then it outputs $\perp$. Otherwise, it continues.

— It runs an extended binary search on tree $\mathsf{EncTree}_i$ to find two adjacent paths $\mathsf{path}_{i,\mathsf{lwr}}$ and $\mathsf{path}_{i,\mathsf{upr}}$ for identities $\mathsf{id}_{i,\mathsf{lwr}}$ and $\mathsf{id}_{i,\mathsf{upr}}$, respectively. (Here $\mathsf{id}_{i,\mathsf{lwr}}$ is the largest identity in $\mathsf{EncTree}_i$ such that $\mathsf{id}_{i,\mathsf{lwr}} < \mathsf{id}$ and similarly $\mathsf{id}_{i,\mathsf{upr}}$ is the smallest identity in $\mathsf{EncTree}_i$ such that $\mathsf{id}_{i,\mathsf{upr}} > \mathsf{id}$.)
If $\mathsf{id}_{i,\mathsf{lwr}}$ is the largest identity registered in the tree $\mathsf{EncTree}_i$, that is no such $\mathsf{id}_{i,\mathsf{upr}}$ exists, then path $\mathsf{path}_{i,\mathsf{upr}}$ is set as $\mathsf{path}_{i,\mathsf{upr}} = \epsilon$. Similarly, if $\mathsf{id}_{i,\mathsf{upr}}$ is the smallest identity, that is no such $\mathsf{id}_{i,\mathsf{lwr}}$ exists, then path $\mathsf{path}_{i,\mathsf{lwr}}$ is set as $\mathsf{path}_{i,\mathsf{lwr}} = \epsilon$.

— It sets sub-proof π_i as $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{upr}})$.

Finally, it outputs the pre-registration proof as $\pi = (\pi_1, \ldots, \pi_{\ell_n})$.

$\mathsf{PreVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \pi) \rightarrow 0/1$. Let $\mathsf{crs} = \mathsf{hk}$, $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_{i\in[\ell_n]}$, $\pi = (\pi_i)_{i\in[\ell_n]}$.[11] Also, let each sub-proof be $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{upr}})$ for $i \in [\ell_n]$.

The pre-registration proof verification procedure proceeds as follows. For every $i \in [\ell_n]$, it runs the pre-registration sub-proof verification procedure which is described in Fig. 3.

If the pre-registration sub-proof verification procedure rejects for any index $i \in [\ell_n]$, then the main verification algorithm also rejects and outputs 0. Otherwise, if all sub-proof verification routines accept, then the main verification algorithm also accepts and outputs 1.

$\mathsf{PostProve}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{id}, \mathsf{pk}) \rightarrow \pi$. Let $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_{i\in[\ell_n]}$ and $\mathsf{aux} = \left(\mathsf{IDTree}, \{(\mathsf{EncTree}_i, n_i)\}_{i\in[\ell_n]}\right)$. The post-registration proof consists of ℓ_n sub-proofs π_i for $i \in [\ell_n]$, where each sub-proof π_i consist of either two or *three* adjacent paths in the i^{th} encryption tree $\mathsf{EncTree}_i$.[12] (Very briefly, having 3 adjacent paths w.r.t. an encryption tree will correspond to the proof of uniqueness of decryptability by the registered user's secret key; whereas 2 adjacent paths will mostly correspond to a proof of non-decryptability.) Concretely, the algorithm proceeds as follows:

Initialize $\ell = \perp$, where ℓ will eventually denote the index of the first encryption tree $\mathsf{EncTree}_\ell$ in which identity id was registered. For $i \in [\ell_n]$:

— It runs a binary search on tree $\mathsf{EncTree}_i$ to find identity id. If the tree contains a leaf node of the form $1||0^\lambda||\mathsf{id}||\mathsf{pk}'$ for some key $\mathsf{pk}' \neq \mathsf{pk}$, then the algorithm simply outputs $\perp$. Otherwise, it continues as follows.

[10] Sometimes one of the paths might just be an empty path.

[11] If the number of sub-proofs and number of encryption trees are distinct, then the verifier rejects. Here we simply consider that while parsing the inputs, the verifier verifies that the crs and pp are consistent which simply corresponds to checking that the number of trees and their depths are consistent.

[12] Sometimes one of the paths might just be an empty path.

Verification procedure for pre-registration sub-proof

For simplicity of exposition, suppose that none of paths $\mathsf{path}_{i,\mathsf{lwr}}$, $\mathsf{path}_{i,\mathsf{upr}}$ are empty. Towards the end, we explain how the verification handles the case if either of these paths is ϵ.

Non-empty paths. It interprets every path $\mathsf{path}_{i,\mathsf{tag}}$ as $(\mathsf{node}_{i,1,\mathsf{tag}}, \ldots, \mathsf{node}_{i,d_i,\mathsf{tag}})$ for $i \in [\ell_n]$ and $\mathsf{tag} \in \{\mathsf{lwr}, \mathsf{upr}\}$. And every node $\mathsf{node}_{i,j,\mathsf{tag}}$, is interpreted as $(\mathsf{flag}_{i,j,\mathsf{tag}}||a_{i,j,\mathsf{tag}}||\mathsf{id}_{i,j,\mathsf{tag}}||b_{i,j,\mathsf{tag}})$.

1. First, it checks that both paths $\mathsf{path}_{i,\mathsf{lwr}}$ and $\mathsf{path}_{i,\mathsf{upr}}$ are well-formed. That is, $\mathsf{node}_{i,1,\mathsf{tag}} = \mathsf{rt}_i$ for both $\mathsf{tag} \in \{\mathsf{lwr}, \mathsf{upr}\}$. Also, it checks that $\mathsf{node}_{i,j+1,\mathsf{tag}}$ is either left child of $\mathsf{node}_{i,j,\mathsf{tag}}$ (i.e., $a_{i,j,\mathsf{tag}} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\mathsf{tag}})$ and $\mathsf{id}_{i,j,\mathsf{tag}} \geq \mathsf{id}_{i,j+1,\mathsf{tag}}$), or right child of $\mathsf{node}_{i,j,\mathsf{tag}}$ (i.e., $b_{i,j,\mathsf{tag}} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\mathsf{tag}})$ and $\mathsf{id}_{i,j,\mathsf{tag}} < \mathsf{id}_{i,j+1,\mathsf{tag}}$). If $\mathsf{node}_{i,j+1,\mathsf{tag}}$ is left child of $\mathsf{node}_{i,j,\mathsf{tag}}$, then it checks that $\mathsf{id}_{i,k,\mathsf{tag}} \leq \mathsf{id}_{i,j,\mathsf{tag}}$ for each $k > j$. Similarly, If $\mathsf{node}_{i,j+1,\mathsf{tag}}$ is right child of $\mathsf{node}_{i,j,\mathsf{tag}}$, then it checks that $\mathsf{id}_{i,k,\mathsf{tag}} > \mathsf{id}_{i,j,\mathsf{tag}}$ for each $k > j$. And, it checks that $\mathsf{flag}_{i,j,\mathsf{tag}} = 0$ for $j < d_i$, and $\mathsf{flag}_{i,d_i,\mathsf{tag}} = 1$, $a_{i,d_i,\mathsf{tag}} = 0^\lambda$. (Note that during this validity check, the verifier also stores whether that node is left child or right child.).
2. Next, it checks that $\mathsf{id}_{i,d_i,\mathsf{lwr}} < \mathsf{id} < \mathsf{id}_{i,d_i,\mathsf{upr}}$, that is the identity in the *lower* path is less than that in the *upper* path, and the identity id whose non-registration is being proven lies between both these identities.
3. It then computes the largest common prefix of nodes in paths $\mathsf{path}_{i,\mathsf{lwr}}$ and $\mathsf{path}_{i,\mathsf{upr}}$. That is, let k be the largest index such that $\mathsf{node}_{i,j,\mathsf{lwr}} = \mathsf{node}_{i,j,\mathsf{upr}}$ for all $j \leq k$. It checks that $\mathsf{id}_{i,k,\mathsf{lwr}} = \mathsf{id}_{i,d_i,\mathsf{lwr}}$. Also, it checks:
   (a) It checks that $\mathsf{node}_{i,k+1,\mathsf{lwr}}$ and $\mathsf{node}_{i,k+1,\mathsf{upr}}$ are *left* and *right* children of $\mathsf{node}_{i,k,\mathsf{lwr}} = \mathsf{node}_{i,k,\mathsf{upr}}$. That is, $a_{i,k,\mathsf{lwr}} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,k+1,\mathsf{lwr}})$ and $b_{i,k,\mathsf{upr}} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,k+1,\mathsf{upr}})$.
   (b) For every index $j > k$, $\mathsf{node}_{i,j+1,\mathsf{lwr}}$ and $\mathsf{node}_{i,j+1,\mathsf{upr}}$ are *right* and *left* children of $\mathsf{node}_{i,j,\mathsf{lwr}}$ and $\mathsf{node}_{i,j,\mathsf{upr}}$, respectively. That is, $b_{i,j,\mathsf{lwr}} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\mathsf{lwr}})$ and $a_{i,j,\mathsf{upr}} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\mathsf{upr}})$.

It rejects, i.e. outputs 0, if any of these checks fails. Otherwise, it accepts and outputs 1.

One empty path. Suppose $\mathsf{path}_{i,\mathsf{lwr}} = \epsilon$. The verifier checks first well-formedness of $\mathsf{path}_{i,\mathsf{upr}}$ as in Step 1 (above). Next, it checks that $\mathsf{id} < \mathsf{id}_{i,d_i,\mathsf{upr}}$, and lastly verifies that $\mathsf{id}_{i,d_i,\mathsf{upr}}$ is the smallest registered node in $\mathsf{EncTree}_i$. For the last check, the verifier check that for every index j, $\mathsf{node}_{i,j+1,\mathsf{upr}}$ is the *left* child of $\mathsf{node}_{i,j,\mathsf{upr}}$. It rejects, i.e. outputs 0, if any of these checks fails. Otherwise, it accepts and outputs 1.

Similarly, if $\mathsf{path}_{i,\mathsf{upr}} = \epsilon$, then it proceeds as above, except it checks that $\mathsf{id}_{i,d_i,\mathsf{lwr}}$ is the largest identity in $\mathsf{EncTree}_i$ instead.

Figure 3: Conditions for verifying a proof $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{upr}})$ that id is NOT registered as per $\mathsf{EncTree}_i$

— If id is *not* contained in $\mathsf{EncTree}_i$, then it first checks that $\ell = \perp$. If the check fails, it aborts. Otherwise, it proceeds as for the pre-registration sub-proof which is to run an extended binary search on tree $\mathsf{EncTree}_i$ to find two adjacent paths $\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{upr}}$ for identities $\mathsf{id}_{i,\mathsf{lwr}}, \mathsf{id}_{i,\mathsf{upr}}$ (respectively). Here $\mathsf{id}_{i,\mathsf{lwr}}$ is the largest identity in $\mathsf{EncTree}_i$ such that $\mathsf{id}_{i,\mathsf{lwr}} < \mathsf{id}$ and similarly $\mathsf{id}_{i,\mathsf{upr}}$ is the smallest identity in $\mathsf{EncTree}_i$ such that $\mathsf{id}_{i,\mathsf{upr}} > \mathsf{id}$. And, it sets sub-proof π_i as $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{upr}})$. (Recall that one of these paths might be empty.)

— If id is contained in $\mathsf{EncTree}_i$, then it proceeds as follows:

  — If $\ell = \perp$, then it sets $\ell = i$ (i.e., sets ℓ as the first tree where id was found).

  — It runs an extended binary search on tree $\mathsf{EncTree}_i$ to find three adjacent paths $\mathsf{path}_{i,\mathsf{lwr}}$, $\mathsf{path}_{i,\mathsf{mid}}$, $\mathsf{path}_{i,\mathsf{upr}}$ for identities $\mathsf{id}_{i,\mathsf{lwr}}$, id, $\mathsf{id}_{i,\mathsf{upr}}$ (respectively). Here $\mathsf{id}_{i,\mathsf{lwr}}$ is the largest identity in $\mathsf{EncTree}_i$ such that $\mathsf{id}_{i,\mathsf{lwr}} < \mathsf{id}$ and similarly $\mathsf{id}_{i,\mathsf{upr}}$ is the smallest identity in $\mathsf{EncTree}_i$ such that $\mathsf{id}_{i,\mathsf{upr}} > \mathsf{id}$.
  If id is the largest identity registered in the tree $\mathsf{EncTree}_i$, that is no such $\mathsf{id}_{i,\mathsf{upr}}$ exists, then

path $\mathsf{path}_{i,\mathsf{upr}}$ is set as $\mathsf{path}_{i,\mathsf{upr}} = \epsilon$. Similarly, if id is the smallest identity, that is no such $\mathsf{id}_{i,\mathsf{lwr}}$ exists, then path $\mathsf{path}_{i,\mathsf{lwr}}$ is set as $\mathsf{path}_{i,\mathsf{lwr}} = \epsilon$.

— It sets sub-proof π_i as $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{mid}}, \mathsf{path}_{i,\mathsf{upr}})$.

Finally, it outputs the post-registration proof as $\pi = (\pi_1, \ldots, \pi_{\ell_n}, \ell)$. (Note that the cut-off index ℓ in included as part of the proof.)

$\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk}, \pi) \to 0/1$. Let $\mathsf{crs} = \mathsf{hk}$, $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_{i\in[\ell_n]}$, $\pi = (\pi_1, \ldots, \pi_{\ell_n}, \ell)$.[13] Now each sub-proof either is interpreted as 3 adjacent paths $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{mid}}, \mathsf{path}_{i,\mathsf{upr}})$, or as 2 adjacent paths $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{upr}})$ for every i.

The post-registration proof verification procedure proceeds as follows. For every $i \in [\ell]$, it runs the *pre-registration* sub-proof verification procedure which is described in Fig. 3. Now, for every $i \in \{\ell, \ell+1, \ldots, \ell_n\}$, it runs the *post-registration* sub-proof verification procedure which is described in Fig. 4.

If any of the pre-registration or post-registration sub-proof verification procedure rejects for any index $i \in [\ell_n]$, then the main verification algorithm also rejects and outputs 0. Otherwise, if all sub-proof verification routines accept, then the main verification algorithm also accepts and outputs 1.

Remark 4.1. *In the above construction, we make the key accumulator maintain a special balanced tree* IDTree *privately. It turns out this is not necessary, and one could easily remove it from our construction, thereby only leaving the list of encryption trees* $\{\mathsf{EncTree}_i\}_i$ *as part of the auxiliary information. However, for ease of exposition, we include* IDTree *explicitly as part of the description.*

4.2 Efficiency

In this section, we prove that our scheme satisfies compactness and efficiency requirements of Definitions 3.1 and 3.2. The analysis of Reg and Upd algorithms is similar to the analysis presented in [GHM+19].

Time Complexity of Reg Algorithm. Given an identity-key pair $(\mathsf{id}, \mathsf{pk})$, the registration algorithm first updates its timestep counter $n := n+1$, and inserts (id, n) tuple into the balanced binary tree IDTree, which is sorted as per identities. This insertion takes $O(\log n)$ time.

The algorithm then inserts $(\mathsf{id}, \mathsf{pk})$ tuple into a balanced binary tree $\mathsf{EncTree}_{\ell_n}$. This involves performing binary search on id in $\mathsf{EncTree}_{\ell_n}$ to identify the leaf at which $1||0^\lambda||\mathsf{id}||\mathsf{pk}$ is to be inserted. As the tree is balanced, the binary search takes $O(\log n)$ time. The registration algorithm then inserts the leaf and balances the tree as per the rules of the underlying data structure. This step also takes $O(\log n)$ time. As inserting leaf and balancing the tree takes $O(\log n)$ time, at most $O(\log n)$ node labels can change during the process. For all such modified nodes, the hash values stored in their ancestors are updated. As the depth of the tree is $O(\log n)$, this involves updating labels of $O(\log^2 n)$ nodes. [14]

The registration algorithm then computes δ and deletes some old versions of encryption trees. In the construction, the key accumulator maintains an invariant that the number of identities n_i associated with various trees $\mathsf{EncTree}_i$ in auxiliary information is a different power of 2. As there are n registered identities, there can be at most $\log n$ trees in auxiliary information. Therefore, computing δ and merging trees takes only $O(\log n)$ time. Overall, the registration process takes $O(\log^2 n)$ time.

Size of Public Parameters. The public parameters consists of root and depth of each tree in auxiliary information. As described earlier, there can be at most $\log n$ trees in auxiliary information. Therefore, this size of public parameters is at most $O(\log n)$.

[13] If the number of sub-proofs and number of encryption trees are distinct, then the verifier rejects. Here we simply consider that while parsing the inputs, the verifier verifies that the crs and pp are consistent which simply corresponds to checking that the number of trees and their depths are consistent.

[14] Note that the registration algorithm needs to first make a copy of $\mathsf{EncTree}_{\ell_n}$, then insert the leaf $1||0^\lambda||\mathsf{id}||\mathsf{pk}$ into the copied tree to create $\mathsf{NewTree}$. Copying a tree naively takes $O(n)$ time. However, this overhead can be easily avoided by not storing the entire $\mathsf{NewTree}$. Basically, $\mathsf{NewTree}$ only stores values of the $O(\log^2 n)$ nodes that are different from $\mathsf{EncTree}_{\ell_n}$.

Verification procedure for post-registration sub-proof

For simplicity of exposition, suppose that none of paths $\mathsf{path}_{i,\mathsf{lwr}}$, $\mathsf{path}_{i,\mathsf{upr}}$ are empty. Towards the end, we explain how the verification handles the case if either of these paths are ϵ.

Non-empty paths. It interprets every path $\mathsf{path}_{i,\mathsf{tag}}$ as $(\mathsf{node}_{i,1\mathsf{tag}}, \ldots, \mathsf{node}_{i,d_i\mathsf{tag}})$ for $i \in [\ell_n]$ and $\mathsf{tag} \in \{\mathsf{lwr}, \mathsf{mid}, \mathsf{upr}\}$. And every node $\mathsf{node}_{i,j,\mathsf{tag}}$, is interpreted as $(\mathsf{flag}_{i,j,\mathsf{tag}}||a_{i,j,\mathsf{tag}}||\mathsf{id}_{i,j,\mathsf{tag}}||b_{i,j,\mathsf{tag}})$.

1. First, it checks that both paths $\mathsf{path}_{i,\mathsf{lwr}}$, $\mathsf{path}_{i,\mathsf{mid}}$ and $\mathsf{path}_{i,\mathsf{upr}}$ are well-formed. That is, $\mathsf{node}_{i,1,\mathsf{tag}} = \mathsf{rt}_i$ for both $\mathsf{tag} \in \{\mathsf{lwr}, \mathsf{mid}, \mathsf{upr}\}$. Also, it checks that $\mathsf{node}_{i,j+1,\mathsf{tag}}$ is either a left child of $\mathsf{node}_{i,j,\mathsf{tag}}$ (i.e., $a_{i,j,\mathsf{tag}} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\mathsf{tag}})$ and $\mathsf{id}_{i,j,\mathsf{tag}} \geq \mathsf{id}_{i,j+1,\mathsf{tag}}$), or is a right child of $\mathsf{node}_{i,j,\mathsf{tag}}$ (i.e., $b_{i,j,\mathsf{tag}} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\mathsf{tag}})$ and $\mathsf{id}_{i,j,\mathsf{tag}} < \mathsf{id}_{i,j+1,\mathsf{tag}}$). If $\mathsf{node}_{i,j+1,\mathsf{tag}}$ is left child of $\mathsf{node}_{i,j,\mathsf{tag}}$, then it checks that $\mathsf{id}_{i,k,\mathsf{tag}} \leq \mathsf{id}_{i,j,\mathsf{tag}}$ for each $k > j$. Similarly, If $\mathsf{node}_{i,j+1,\mathsf{tag}}$ is right child of $\mathsf{node}_{i,j,\mathsf{tag}}$, then it checks that $\mathsf{id}_{i,k,\mathsf{tag}} > \mathsf{id}_{i,j,\mathsf{tag}}$ for each $k > j$. And, it checks that $\mathsf{flag}_{i,j,\mathsf{tag}} = 0$ for $j < d_i$, and $\mathsf{flag}_{i,d_i,\mathsf{tag}} = 1$, $a_{i,d_i,\mathsf{tag}} = 0^\lambda$. (Note that during this validity check, the verifier also stores whether that node is left child or right child.)
2. Next, it checks that $\mathsf{id}_{i,d_i,\mathsf{lwr}} < \mathsf{id} = \mathsf{id}_{i,d_i,\mathsf{mid}} < \mathsf{id}_{i,d_i,\mathsf{upr}}$, that is the identity in the *lower* path is less than that in the *upper* path, and the identity id whose non-registration is being proven is equal to the identity in the *middle* path and lies between the other two identities. It also checks that $b_{i,d_i,\mathsf{mid}} = \mathsf{pk}$.
3. For both tag pairs $(\mathsf{tag}_1, \mathsf{tag}_2) \in \{(\mathsf{lwr}, \mathsf{mid}), (\mathsf{mid}, \mathsf{upr})\}$, it proceeds as follows:
   It computes the largest common prefix of nodes in paths $\mathsf{path}_{i,\mathsf{tag}_1}$ and $\mathsf{path}_{i,\mathsf{tag}_2}$. That is, let k be the largest index such that $\mathsf{node}_{i,j,\mathsf{tag}_1} = \mathsf{node}_{i,j,\mathsf{tag}_2}$ for all $j \leq k$. It checks that $\mathsf{id}_{i,k,\mathsf{tag}_1} = \mathsf{id}_{i,d_i,\mathsf{tag}_1}$. Also, it checks:
   (a) It checks that $\mathsf{node}_{i,k+1,\mathsf{tag}_1}$ and $\mathsf{node}_{i,k+1,\mathsf{tag}_2}$ are *left* and *right* children of $\mathsf{node}_{i,k,\mathsf{tag}_1} = \mathsf{node}_{i,k,\mathsf{tag}_2}$. That is, $a_{i,k,\mathsf{tag}_1} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,k+1,\mathsf{tag}_1})$ and $b_{i,k,\mathsf{tag}_2} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,k+1,\mathsf{tag}_2})$.
   (b) For every index $j > k$, $\mathsf{node}_{i,j+1,\mathsf{tag}_1}$ and $\mathsf{node}_{i,j+1,\mathsf{tag}_2}$ are *right* and *left* children of $\mathsf{node}_{i,j,\mathsf{tag}_1}$ and $\mathsf{node}_{i,j,\mathsf{tag}_2}$, respectively. That is, $b_{i,j,\mathsf{tag}_1} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\mathsf{tag}_1})$ and $a_{i,j,\mathsf{tag}_2} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\mathsf{tag}_2})$.

It rejects, i.e. outputs 0, if any of these checks fails. Otherwise, it accepts and outputs 1.

One empty path. Suppose $\mathsf{path}_{i,\mathsf{lwr}} = \epsilon$. The verifier checks first well-formedness of $\mathsf{path}_{i,\mathsf{mid}}, \mathsf{path}_{i,\mathsf{upr}}$ as in Step 1 (above). Next, it checks that $\mathsf{id} = \mathsf{id}_{i,d_i,\mathsf{mid}} < \mathsf{id}_{i,d_i,\mathsf{upr}}$ as in Step 2 (above). And lastly, it performs the Step 3 verification checks as described above only for the tag pair $(\mathsf{tag}_1, \mathsf{tag}_2) = (\mathsf{mid}, \mathsf{upr})$. Lastly verifies that $\mathsf{node}_{i,d_i,\mathsf{mid}}$ is the smallest registered node in $\mathsf{EncTree}_i$ i.e., the verifier checks that for every index j, $\mathsf{node}_{i,j+1,\mathsf{mid}}$ is the *left* child of $\mathsf{node}_{i,j,\mathsf{mid}}$. It rejects, i.e. outputs 0, if any of these checks fail. Otherwise, it accepts and outputs 1.

Similarly, if $\mathsf{path}_{i,\mathsf{upr}} = \epsilon$, then it proceeds complementarily to above which is to check well-formedness of $\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{mid}}$, range check $\mathsf{id}_{i,d_i,\mathsf{lwr}} < \mathsf{id} = \mathsf{id}_{i,d_i,\mathsf{mid}}$, and lastly it performs the Step 3 verification checks only for the tag pair $(\mathsf{tag}_1, \mathsf{tag}_2) = (\mathsf{lwr}, \mathsf{mid})$. Lastly verifies that $\mathsf{node}_{i,d_i,\mathsf{mid}}$ is the largest registered node in $\mathsf{EncTree}_i$ i.e., the verifier checks that for every index j, $\mathsf{node}_{i,j+1,\mathsf{mid}}$ is the *right* child of $\mathsf{node}_{i,j,\mathsf{mid}}$. It rejects, i.e. outputs 0, if any of these checks fail. Otherwise, it accepts and outputs 1.

Figure 4: Conditions for verifying a proof $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{mid}}, \mathsf{path}_{i,\mathsf{upr}})$ that id is registered as per $\mathsf{EncTree}_i$

Time Complexity of Upd Algorithm. During the update process, the Upd algorithm first computes the timestep t_{id} at which the given identity id is registered by performing a binary search through balanced binary tree IDTree. As IDTree is balanced, the binary search requires only $O(\log n)$ time. After computing t_{id}, the Upd algorithm identifies the balanced binary tree $\mathsf{EncTree}_i$ that is associated with timestep t_{id}. As described earlier, there can be at most $\log n$ trees, and therefore this operation takes $O(\log n)$ time. The key accumulator then performs a binary search on id in the associated tree $\mathsf{EncTree}_i$ and outputs the sequence

of nodes traversed during the binary search. As $\mathsf{EncTree}_i$ is balanced, this step also requires $O(\log n)$ time. Overall, the update algorithm runs in $O(\log n)$ time.

Size of an Update. The Upd algorithm performs a binary search on given identity id in a balanced binary tree $\mathsf{EncTree}_i$ which contains registered identity-key pairs. The algorithm then outputs the sequence of nodes traversed during the binary search. As the tree is balanced and as there are at most n identities in the tree, the depth of the tree is $O(\log n)$. Therefore, the size of an update output by Upd algorithm is $O(\log n)$.

Number of Updates. In our construction, we say that a registered identity id with timestamp t is associated with $\mathsf{EncTree}_j$ if $\sum_{k<j} n_k < t \leq \sum_{k\leq j} n_k$. A registered identity id needs to invoke Upd algorithm only when the tree $\mathsf{EncTree}_j$ associated with id gets deleted during Reg algorithm. By our construction, any pair $(\mathsf{EncTree}_i, n_i)$ gets deleted only when n_i new identites are registered after $(\mathsf{EncTree}_i, n_i)$ is added to aux. Therefore for any identity id, its i^{th} update happens when 2^i new identities are registered after its $(i-1)^{th}$ update. As there are at most n registrations, the number of updates received by any identity is at most $\log n$.

Time Complexity of $\mathsf{PreProve}$/$\mathsf{PostProve}$ Algorithms. Given an identity id, the $\mathsf{PreProve}$/$\mathsf{PostProve}$ algorithms perform binary search on at most 3 identities in each balanced binary tree $\mathsf{EncTree}_i$ present in the aux. As each tree in aux is balanced, contains at most n leaves and is ordered as per identities, a binary search on each tree takes $O(\log n)$ time. In the construction, the key accumulator maintains an invariant that the number of identities n_i associated with various trees $\mathsf{EncTree}_i$ in auxiliary information is a different power of 2. As there are n registered identities, there can be at most $\log n$ trees in auxiliary information. Therefore, the time complexity of $\mathsf{PreProve}$ and $\mathsf{PostProve}$ algorithms is at most $O(\log^2 n)$.

Size of Pre/Post-Registration Proofs. The pre/post-registration sub-proofs consist of at most 3 paths (sequence of nodes from root to a leaf) for each tree in auxiliary information. As each tree in aux is balanced and contains at most n leaves, the depth of each tree is $O(\log n)$. In the construction, the key accumulator maintains an invariant that the number of identities n_i associated with various trees $\mathsf{EncTree}_i$ in auxiliary information is a different power of 2. As there are n registered identities, there can be at most $\log n$ trees in auxiliary information. Therefore, the size of pre/post-registration proofs is $O(\log^2 n)$.

4.3 Completeness

We now show that the above scheme satisfies completenesss requirements described in Definitions 3.1 to 3.3.

4.3.1 Completeness of VRBE

Lemma 4.1. *Assuming* HG *and* PKE *are perfectly correct, the above construction satisfies completeness property (Definition 3.1)*[15].

Proof. Consider any computationally unbounded adversary $\mathcal{A}$. The challenger first initializes $(\mathsf{pp}, \mathsf{aux}, u, S_{\mathsf{ID}}, \mathsf{id}^*) = (\epsilon, \epsilon, \epsilon, \emptyset, \perp)$, samples $\mathsf{crs} = \mathsf{hk}$ and sends it to $\mathcal{A}$. The adversary requests for polynomially many registration queries of the form $(\mathsf{regnew}, \mathsf{id}, \mathsf{pk})$ or $(\mathsf{regtgt}, \mathsf{id}^*)$. For each $(\mathsf{regnew}, \mathsf{id}, \mathsf{pk})$ query, the challenger registers $(\mathsf{id}, \mathsf{pk})$. For $(\mathsf{regtgt}, \mathsf{id}^*)$ query, the challenger samples PKE pair $(\mathsf{pk}^*, \mathsf{sk}^*)$, registers $(\mathsf{id}^*, \mathsf{pk}^*)$ and sends back pk^* to the adversary. The adversary then makes an encryption query (enctgt, m) and obtains $\mathsf{ct} = (\mathsf{pp}, \{\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}\}_{i,j}, \{\tilde{y}\}_{i,1}) \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{id}^*, m)$, where id^* is the challenge identity and $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_i$ is public parameters at the time of encryption query. The adversary then makes few more registration

[15]For the sake of simplicity, we consider the completeness game in which the adversary makes only one encryption and decryption query. Note that if an adversary can break completeness property with access to polynomially many encryption and decryption queries (as in Definition 3.1), it can also break the completeness property with access to a single encryption and decryption query.

queries and then requests to decrypt the ciphertext ct. The challenger then runs the decryption algorithm $x = \mathsf{Dec}(\mathsf{sk}^*, u, \mathsf{ct})$. We know that x is either GetUpd or $\perp$ or a message. If $x = \mathsf{GetUpd}$, the challenger runs the update algorithm $u \leftarrow \mathsf{Upd}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{id}^*)$ and then again runs decryption algorithm $x = \mathsf{Dec}(\mathsf{sk}^*, u, \mathsf{ct})$. Assuming $x \neq \mathsf{GetUpd}$, we now prove that x is always equals to m. Let u be $(\mathsf{node}_1, \mathsf{node}_2, \cdots \mathsf{node}_d)$ and $\mathsf{node}_i = \mathsf{flag}_i||a_i||\mathsf{id}_i||b_i$ for each i. We first make the following observations. (1) $\mathsf{flag}_i = 1$ iff $i = d$, (2) $\mathsf{id}_d = \mathsf{id}^*$, $b_d = \mathsf{pk}^*$, (3) $\mathsf{node}_1 = \mathsf{rt}_\kappa$ for some $\kappa \in [\ell_n]$, and (4) for each $i < d$, $\mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i+1}) = \begin{cases} a_i & \text{if } \mathsf{id}^* \leq \mathsf{id}_i \\ b_i & \text{Otherwise} \end{cases}$. The last observation is due to the fact that u is obtained by performing binary search on id^* in an $\mathsf{EncTree}$. Based on the above observations, the decryption algorithm runs $\mathsf{HEval}(\widetilde{\mathsf{Enc\text{-}Step}}_{\kappa,1}, \tilde{y}_{\kappa,1}, \mathsf{node}_1)$. By the perfect correctness of hash garbling scheme, it obtains $0||\tilde{y}_{\kappa,2} = 0||\mathsf{HG.GarbleInp}(\mathsf{hk}, \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_2), \mathsf{state}_{\kappa,2}; r_{\kappa,2})$. The algorithm then runs $\mathsf{HEval}(\widetilde{\mathsf{Enc\text{-}Step}}_{\kappa,2}, \tilde{y}_{\kappa,2}, \mathsf{node}_2)$ and obtains $0||\mathsf{HG.GarbleInp}(\mathsf{hk}, \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_3), \mathsf{state}_{\kappa,3}; r_{\kappa,3})$ by perfect correctness of hash garbling scheme. On continuing the process, the final circuit outputs $1||\mathsf{Enc}(\mathsf{pk}^*, m)$. The decryption algorithm then decrypts the PKE ciphertext $\mathsf{Enc}(\mathsf{pk}^*, m)$ and obtains m by perfect correctness of PKE scheme. ■

Remark 4.2 (Handling missing updates)**.** *Note that the above completeness proof relies on the fact that the user has the update u associated w.r.t. the public parameters pp used during encryption. However, this does not capture the scenario when the challenger registers many identities after ct is created, and due to delete operations in Reg algorithm, the $\mathsf{EncTree}$ associated with identity id is changed. That is, in this case, it might have happened that even after Upd algorithm is executed, the root value of the update u may not match with any root value rt_i used while creating the ciphertext. Thus, the decryption algorithm outputs GetUpd again and therefore it is not possible to decrypt such a well-formed ciphertext. Intuitively, the issue is that the updates could get* lost *during the registration process. Identical issues were observed in the prior works [GHMR18, GHM+19] as well. Here we sketch a simple and efficient mechanism to resolve the issue. For ease of exposition, we ignore this detail in the formal description of our scheme.*

The idea is to simply store a list of 'old' updates for every user in the IDTree that we already maintain. In order to preserve efficiency, an update is added to the list if it is not already present, that is only after an existing encryption tree is removed. More formally, the tree IDTree now stores tuples of the form $(\mathsf{id}, (t, L))$, where t denotes the timestamp as before and L denotes the list of all the past updates (path from the root of an encryption tree to the leaf containing the identity id). Now during the registration, when an encryption tree $\mathsf{EncTree}$ associated with an identity id changes (i.e., is removed), then the key accumulator for every such identity id appends the update information (i.e., path from the root to leaf containing id) to the tuple $(\mathsf{id}, (t, L))$ in the tree IDTree. Note that this can be efficiently done by performing a binary search on IDTree, and does not blow up the size of public parameters. For generating updates, the algorithm searches for the given identity id in IDTree to obtain the tuple $(\mathsf{id}, (t, L))$ and sends the list L containing all the past updates in addition to the current update information. The decryption algorithm on receiving the list of all the past updates, decrypts the ciphertext with each of the updates and then outputs the message if any update works.

Lastly, such a scenario could be more easily handled in real-world applications by issuing out updates after every deletion instead of storing them. That is, when the key accumulator possibly deletes existing encryption trees (Step (3) in Reg algorithm), then it runs the update algorithm $\mathsf{Upd}(\mathsf{pp}, \mathsf{id})$ and sends the update for each identity id with timestamp greater than $\sum_{i<\ell_n+1-\delta} n_i$.

4.3.2 Completeness of Pre/Post-Registration Verifiability

Lemma 4.2. *The above construction satisfies completeness of pre/post-Registration Verifiability property (Definition 3.2)*[16]*.*

[16]For the sake of simplicity, we consider the completeness game in which the adversary makes only one prereg or $\mathsf{postreg}$ query. Note that if an adversary can break completeness property with access to polynomially many prereg and $\mathsf{postreg}$ queries (as in Definition 3.2), it can also break the completeness property with access to single prereg or $\mathsf{postreg}$ query.

Proof. Consider any computationally unbounded adversary $\mathcal{A}$. The challenger first initializes $(\mathsf{pp}, \mathsf{aux}, u, S_{\mathsf{ID}}, \mathsf{id}^*) = (\epsilon, \epsilon, \epsilon, \emptyset, \perp)$, samples $\mathsf{crs} = \mathsf{hk}$ and sends it to $\mathcal{A}$. The adversary requests for polynomially many registration queries of the form $(\mathsf{regnew}, \mathsf{id}, \mathsf{pk})$. For each $(\mathsf{regnew}, \mathsf{id}, \mathsf{pk})$ query, the challenger registers $(\mathsf{id}, \mathsf{pk})$ as $\mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk})$. The adversary then either queries $(\mathsf{prereg}, \mathsf{id}^*)$ on any unregistered identity id^* or $(\mathsf{postreg}, \mathsf{id}^*, \mathsf{pk}^*)$ on any registered key-pair $(\mathsf{id}^*, \mathsf{pk}^*)$.

Suppose $\mathcal{A}$ makes query $(\mathsf{prereg}, \mathsf{id}^*)$ on any unregistered identity id^*. The challenger computes $\pi = \mathsf{PreProve}^{\mathsf{aux}}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*)$. We argue that $\mathsf{PreVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \pi) = 1$ with probability 1. Let $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_{i \in [\ell_n]}$ and $\pi = (\pi_1, \cdots, \pi_{\ell_n})$, where $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{upr}})$ for each i. For each i and $\mathsf{tag} \in \{\mathsf{lwr}, \mathsf{upr}\}$, let $\mathsf{path}_{i,\mathsf{tag}} = (\mathsf{node}_{i,1,\mathsf{tag}}, \cdots, \mathsf{node}_{i,d_i,\mathsf{tag}})$, where $\mathsf{node}_{i,j,\mathsf{tag}} = \mathsf{flag}_{i,j,\mathsf{tag}} || a_{i,j,\mathsf{tag}} || \mathsf{id}_{i,j,\mathsf{tag}} || b_{i,j,\mathsf{tag}}$. As $\mathsf{path}_{i,\mathsf{lwr}}$ and $\mathsf{path}_{i,\mathsf{upr}}$ are obtained by performing binary search in $\mathsf{EncTree}_i$, they satisfy well-formedness conditions (Point 1 in Fig. 3). As id^* is an unregistered user, the extended binary search on id^* in $\mathsf{EncTree}_i$ leads to identities $\mathsf{id}_{i,d_i,\mathsf{lwr}}$ and $\mathsf{id}_{i,d_i,\mathsf{upr}}$ s.t. $\mathsf{id}_{i,d_i,\mathsf{lwr}} < \mathsf{id}^* < \mathsf{id}_{i,d_i,\mathsf{upr}}$. As $\mathsf{EncTree}_i$ is ordered as per identites and as there are no registered identities between $\mathsf{id}_{i,d_i,\mathsf{lwr}}$ between $\mathsf{id}_{i,d_i,\mathsf{upr}}$, the paths $\mathsf{path}_{i,\mathsf{lwr}}$ and $\mathsf{path}_{i,\mathsf{upr}}$ are 'adjacent' and therefore satisfies condition 3 in Fig. 3.

On the other hand, suppose $\mathcal{A}$ makes query $(\mathsf{postreg}, \mathsf{id}^*, \mathsf{pk}^*)$ on any registered identity-key pair $(\mathsf{id}^*, \mathsf{pk}^*)$. The challenger computes $\pi = \mathsf{PostProve}^{\mathsf{aux}}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*)$. We argue that $\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*, \pi) = 1$ with probability 1. Let $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_{i \in [\ell_n]}$ and $\pi = (\pi_1, \cdots, \pi_{\ell_n}, \alpha)$. For any $i < \alpha$, let $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{upr}})$. For any $i \geq \alpha$, let $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{mid}}, \mathsf{path}_{i,\mathsf{upr}})$. For each i and $\mathsf{tag} \in \{\mathsf{lwr}, \mathsf{mid}, \mathsf{upr}\}$, let $\mathsf{path}_{i,\mathsf{tag}} = (\mathsf{node}_{i,1,\mathsf{tag}}, \cdots, \mathsf{node}_{i,d_i,\mathsf{tag}})$, where $\mathsf{node}_{i,j,\mathsf{tag}} = \mathsf{flag}_{i,j,\mathsf{tag}} || a_{i,j,\mathsf{tag}} || \mathsf{id}_{i,j,\mathsf{tag}} || b_{i,j,\mathsf{tag}}$. As the paths are obtained by performing binary search in $\mathsf{EncTree}_i$, they satisfy well-formedness conditions (Point 1 in Figs. 3 and 4). For each $i < \alpha$, we know that id^* is unregistered in $\mathsf{EncTree}_i$, and therefore extended binary search on id^* in $\mathsf{EncTree}_i$ leads to identities $\mathsf{id}_{i,d_i,\mathsf{lwr}}$ and $\mathsf{id}_{i,d_i,\mathsf{upr}}$ s.t. $\mathsf{id}_{i,d_i,\mathsf{lwr}} < \mathsf{id}^* < \mathsf{id}_{i,d_i,\mathsf{upr}}$. Moreover, as $\mathsf{EncTree}_i$ is ordered as per identites and as there are no registered identities between $\mathsf{id}_{i,d_i,\mathsf{lwr}}$ between $\mathsf{id}_{i,d_i,\mathsf{upr}}$, the paths $\mathsf{path}_{i,\mathsf{lwr}}$ and $\mathsf{path}_{i,\mathsf{upr}}$ are 'adjacent' and therefore satisfies condition 3 in Fig. 3. For each $i \geq \alpha$, we know that $(\mathsf{id}^*, \mathsf{pk}^*)$ is registered in $\mathsf{EncTree}_i$, and therefore extended binary search on id^* in $\mathsf{EncTree}_i$ leads to identities $\mathsf{id}_{i,d_i,\mathsf{lwr}}, \mathsf{id}_{i,d_i,\mathsf{mid}}, \mathsf{id}_{i,d_i,\mathsf{upr}}$ s.t. $\mathsf{id}_{i,d_i,\mathsf{lwr}} < \mathsf{id}_{i,d_i,\mathsf{mid}} = \mathsf{id}^* < \mathsf{id}_{i,d_i,\mathsf{upr}}$. Moreover, as $\mathsf{EncTree}_i$ is ordered as per identites and as there are no registered identities between $\mathsf{id}_{i,d_i,\mathsf{lwr}}, \mathsf{id}_{i,d_i,\mathsf{mid}}$, the paths $\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{mid}}$ are 'adjacent' and therefore satisfy condition 3 in Fig. 4. Similarly, the paths $\mathsf{id}_{i,d_i,\mathsf{mid}}, \mathsf{id}_{i,d_i,\mathsf{upr}}$, are 'adjacent' and satisfy condition 3 in Fig. 4.

■

4.3.3 Efficiently Extractable Completeness for Post-Registration

We now prove a lemma that is useful for proving soundness of pre/post-registration properties. Consider any $\mathsf{crs} = \mathsf{hk} \leftarrow \mathsf{HG.Setup}(1^\lambda, 1^{3\lambda+1})$, public parameters $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_i$, identity id^* and a sub-proof $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{upr}})$. Suppose, $(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \pi_i)$ satisfies the verification conditions described in Fig. 3. We prove that if $\mathsf{path}_{i,\mathsf{upr}} \neq \epsilon$, then $\mathsf{path}_{i,\mathsf{upr}}$ resembles a sequence of nodes obtained by performing binary search on id^* in some tree $\mathsf{EncTree}$ with root value rt_i. If $\mathsf{path}_{i,\mathsf{upr}} = \epsilon$, we prove that $\mathsf{path}_{i,\mathsf{lwr}}$ resembles a sequence of nodes obtained by performing binary search on id^* in some tree $\mathsf{EncTree}$ with root value rt_i. Concretely, we prove the following lemma.

Lemma 4.3. *Consider any hash key* hk*, public parameters* $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_i$*, identity* id^* *and a sub pre-registration proof* $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{upr}})$*. For any* $\mathsf{tag} \in \{\mathsf{lwr}, \mathsf{upr}\}$*, let* $\mathsf{path}_{i,\mathsf{tag}} = (\mathsf{node}_{i,1,\mathsf{tag}}, \cdots, \mathsf{node}_{i,d_i,\mathsf{tag}})$*, where* $\mathsf{node}_{i,j,\mathsf{tag}} = \mathsf{flag}_{i,j,\mathsf{tag}} || a_{i,j,\mathsf{tag}} || \mathsf{id}_{i,j,\mathsf{tag}} || b_{i,j,\mathsf{tag}}$*. If* $(\mathsf{hk}, \mathsf{pp}, \mathsf{id}^*, \pi_i)$ *satisfies the pre-registration sub-proof verification procedure of Fig. 3, then* π_i *satisfies the following conditions.*

- $\mathsf{node}_{i,1,\widetilde{\mathsf{tag}}} = \mathsf{rt}_i$, $\mathsf{id}_{i,d_i,\widetilde{\mathsf{tag}}} \neq \mathsf{id}^*$, $\mathsf{flag}_{i,j,\widetilde{\mathsf{tag}}} = 1$ *iff* $j = d_i$ *and*
- *For each* $j < d_i$, $\mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\widetilde{\mathsf{tag}}}) = \begin{cases} a_{i,j,\widetilde{\mathsf{tag}}} & \text{if } \mathsf{id}^* \leq \mathsf{id}_{i,j,\widetilde{\mathsf{tag}}} \\ b_{i,j,\widetilde{\mathsf{tag}}} & \text{if } \mathsf{id}^* > \mathsf{id}_{i,j,\widetilde{\mathsf{tag}}} \end{cases}$

where $\widetilde{\mathsf{tag}} = \mathsf{upr}$ *and* $\mathsf{path}_{i,\mathsf{upr}} \neq \epsilon$*, or* $\widetilde{\mathsf{tag}} = \mathsf{lwr}$ *and* $\mathsf{path}_{i,\mathsf{upr}} = \epsilon$*.*

Proof. The fact that subproof π_i satisfies the first condition directly from the checks performed by $\mathsf{PreVerify}$ algorithm. We now prove that π_i satisfies the second condition. Let us first consider the case where $\mathsf{path}_{i,\mathsf{upr}} \neq \epsilon$ and $\widetilde{\mathsf{tag}} = \mathsf{upr}$. We prove that if the above condition is violated for any j, then the sub-proof π_i violates one of the conditions in Fig. 3.

1. Case 1 (There exists j s.t. $\mathsf{id}^* > \mathsf{id}_{i,j,\mathsf{upr}}$ and $\mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\mathsf{upr}}) \neq b_{i,j,\mathsf{upr}}$): As π_i satisfies the well-formedness condition, we know that $\mathsf{node}_{i,j+1,\mathsf{upr}}$ is left child of $\mathsf{node}_{i,j,\mathsf{upr}}$ (i.e., $\mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\mathsf{upr}}) = a_{i,j,\mathsf{upr}}$ and $\mathsf{id}_{i,j+1,\mathsf{upr}} \leq \mathsf{id}_{i,j,\mathsf{upr}}$). Consequently, for every $k > j$, we have $\mathsf{id}_{i,k,\mathsf{upr}} \leq \mathsf{id}_{i,j,\mathsf{upr}}$. Therefore, $\mathsf{id}_{i,d_i,\mathsf{upr}} \leq \mathsf{id}_{i,j,\mathsf{upr}} < \mathsf{id}^*$ and the sub-proof π_i violates condition 2 in Fig. 3.
2. Case 2 (There exists j s.t. $\mathsf{id}^* \leq \mathsf{id}_{i,j,\mathsf{upr}}$ and $\mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\mathsf{upr}}) \neq a_{i,j,\mathsf{upr}}$): As π_i satisfies the well-formedness condition, we know that $\mathsf{node}_{i,j+1,\mathsf{upr}}$ is right child of $\mathsf{node}_{i,j,\mathsf{upr}}$ (i.e., $\mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\mathsf{upr}}) = b_{i,j,\mathsf{upr}}$ and $\mathsf{id}_{i,j+1,\mathsf{upr}} > \mathsf{id}_{i,j,\mathsf{upr}}$). Let k be the largest index such that for all $\ell \leq k$, $\mathsf{node}_{i,\ell,\mathsf{upr}} = \mathsf{node}_{i,\ell,\mathsf{lwr}}$. We now consider 3 subcases.
   - Case 2a ($j < k$): We know that $\mathsf{node}_{i,j+1,\mathsf{lwr}}$ is the right child of $\mathsf{node}_{i,j,\mathsf{lwr}}$. Consequently, by the well-formedness condition, $\mathsf{id}_{i,d_i,\mathsf{lwr}} > \mathsf{id}_{i,j,\mathsf{lwr}} \geq \mathsf{id}^*$ and the sub-proof π_i violates condition 2 in Fig. 3.
   - Case 2b ($j > k$): As $\mathsf{node}_{i,j+1,\mathsf{upr}}$ is right child of $\mathsf{node}_{i,j,\mathsf{upr}}$, the sub-proof π_i violates the condition (3b) in Fig. 3.
   - Case 2c ($j = k$): As π_i satisfies conditions 2 and 3 of Fig. 3, we know that $\mathsf{id}_{i,k,\mathsf{lwr}} = \mathsf{id}_{i,d_i,\mathsf{lwr}} < \mathsf{id}^*$. This violates our assumption that $\mathsf{id}^* \leq \mathsf{id}_{i,j,\mathsf{upr}} = \mathsf{id}_{i,j,\mathsf{lwr}}$.

Let us now consider the case where $\mathsf{path}_{i,\mathsf{upr}} = \epsilon$ and $\widetilde{\mathsf{tag}} = \mathsf{lwr}$. In this case, we know that for each $j < d_i$, $\mathsf{node}_{i,j+1,\mathsf{lwr}}$ is right child of $\mathsf{node}_{i,j,\mathsf{lwr}}$ and $\mathsf{id}^* > \mathsf{id}_{i,d_i,\mathsf{lwr}}$. This implies, for each $j < d_i$, $\mathsf{id}^* > \mathsf{id}_{i,j,\mathsf{lwr}}$ and $b_{i,j,\mathsf{lwr}} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\mathsf{lwr}})$. ■

We now state a similar lemma that is useful for proving efficiently extractable completeness for post-registration property and soundness of post-registration property. Consider any $\mathsf{crs} = \mathsf{hk} \leftarrow \mathsf{HG.Setup}(1^\lambda, 1^{3\lambda+1})$, public parameters $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_i$, identity id^*, public key pk^* and a sub-proof $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{mid}}, \mathsf{path}_{i,\mathsf{upr}})$. Suppose, $(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*, \pi_i)$ satisfies the verification conditions described in Fig. 4. We prove that $\mathsf{path}_{i,\mathsf{mid}}$ resembles a sequence of nodes obtained by performing binary search on id^* in some tree $\mathsf{EncTree}$ with root value rt_i. Concretely, we prove the following lemma.

Lemma 4.4. *Consider any hash key* hk*, public parameters* $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_i$*, identity* id^**, public key* pk^* *and a sub-proof* $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{mid}}, \mathsf{path}_{i,\mathsf{upr}})$*. For any* $\mathsf{tag} \in \{\mathsf{lwr}, \mathsf{mid}, \mathsf{upr}\}$*, let* $\mathsf{path}_{i,\mathsf{tag}} = (\mathsf{node}_{i,1,\mathsf{tag}}, \cdots, \mathsf{node}_{i,d_i,\mathsf{tag}})$*, where* $\mathsf{node}_{i,j,\mathsf{tag}} = \mathsf{flag}_{i,j,\mathsf{tag}} || a_{i,j,\mathsf{tag}} || \mathsf{id}_{i,j,\mathsf{tag}} || b_{i,j,\mathsf{tag}}$*. If* $(\mathsf{hk}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*, \pi_i)$ *satisfies the post-registration sub-proof verification procedure of Fig. 4, then* π_i *satisfies the following conditions.*

- $\mathsf{node}_{i,1,\mathsf{mid}} = \mathsf{rt}_i$*,* $\mathsf{id}_{i,d_i,\mathsf{mid}} = \mathsf{id}^*$*,* $b_{i,d_i,\mathsf{mid}} = \mathsf{pk}^*$*,* $\mathsf{flag}_{i,j,\mathsf{mid}} = 1$ *iff* $j = d_i$ *and*
- *For each* $j < d_i$*, we have* $\mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\mathsf{mid}}) = \begin{cases} a_{i,j,\mathsf{mid}} & \text{if } \mathsf{id}^* \leq \mathsf{id}_{i,j,\mathsf{mid}} \\ b_{i,j,\mathsf{mid}} & \text{if } \mathsf{id}^* > \mathsf{id}_{i,j,\mathsf{mid}} \end{cases}$.

Proof. This proof is similar to the proof of Lemma 4.3. ■

Lemma 4.5. *Assuming* HG *and* PKE *are perfectly correct, the above construction satisfies efficiently extractable completeness for post-registration property (Definition 3.3).*

Proof. We first present an UpdExt algorithm which takes a post-registration proof π and an identity id as input and outputs an update u. We then prove the construction satisfies efficiently extractable completeness for post-registration property w.r.t the UpdExt algorithm.

$\mathsf{UpdExt}(\pi, \mathsf{id}) \to \{u/\perp\}$: Let the proof π be $(\pi_1, \cdots, \pi_{\ell_n}, \alpha)$, where $\alpha \in [\ell_n]$. For each index $i < \alpha$, let $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{upr}})$. For each $i \geq \alpha$, let $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{mid}}, \mathsf{path}_{i,\mathsf{upr}})$. Output $\perp$ if the proof is not in this format. Otherwise, output update $u = \mathsf{path}_{\alpha,\mathsf{mid}}$.

Consider any computationally unbounded *admissible* adversary $\mathcal{A}$. The challenger first samples $\mathsf{crs} = \mathsf{hk} \leftarrow \mathsf{CRSGen}(1^\lambda)$, a PKE key pair $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{Gen}(1^\lambda)$ and sends $(\mathsf{crs}, \mathsf{pk}^*)$ to the adversary. The adversary then outputs public parameters pp, identity id^*, proof π and a message m. Let $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_{i\in[\ell_n]}$ and $\pi = (\pi_1, \cdots, \pi_{\ell_n}, \alpha)$. As $\mathcal{A}$ is admissible, we know that π is a valid post-registration proof i.e., $\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*, \pi) = 1$. The challenger first computes $\mathsf{ct} = (\mathsf{pp}, \{\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}\}_{i,j}, \{\tilde{y}_{i,1}\}_{i,1}) \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, m)$, extracts $u = \mathsf{UpdExt}(\pi, \mathsf{id})$ and then runs $x = \mathsf{Dec}(\mathsf{sk}, u, \mathsf{ct})$. Let $u = (\mathsf{node}_1, \cdots, \mathsf{node}_d)$, where $\mathsf{node}_i = \mathsf{flag}_i||a_i||\mathsf{id}_i||b_i$ for each i. As $\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*, \pi) = 1$, by Lemma 4.4, we know (1) $\mathsf{flag}_i = 1$ iff $i = d$, (2) $\mathsf{id}_d = \mathsf{id}^*$, $b_d = \mathsf{pk}^*$, (3) $\mathsf{node}_1 = \mathsf{rt}_\alpha$, and (4) for each $i < d$, $\mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i+1}) = \begin{cases} a_i & \text{if } \mathsf{id}^* \leq \mathsf{id}_i \\ b_i & \text{Otherwise} \end{cases}$. Based on the above observations, we now show that $x = m$. As $\mathsf{node}_1 = \mathsf{rt}_\alpha$, the decryption algorithm runs $\mathsf{HEval}(\widetilde{\mathsf{Enc\text{-}Step}}_{\alpha,1}, \tilde{y}_{\alpha,1}, \mathsf{node}_1)$. By the perfect correctness of hash garbling scheme, this results in $0||\tilde{y}_{\alpha,2} = \begin{cases} 0||\mathsf{HG.GarbleInp}(\mathsf{hk}, a_2, \mathsf{state}_{i,2}; r_{i,2}) & \text{if } \mathsf{id}^* \leq \mathsf{id}_1 \\ 0||\mathsf{HG.GarbleInp}(\mathsf{hk}, b_2, \mathsf{state}_{i,2}; r_{i,2}) & \text{Otherwise} \end{cases}$. By our observation, $0||\tilde{y}_{\alpha,2} = 0||\mathsf{HG.GarbleInp}(\mathsf{hk}, \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,2}), \mathsf{state}_{i,2}; r_{i,2})$. The decryption algorithm then runs $\mathsf{HEval}(\widetilde{\mathsf{Enc\text{-}Step}}_{i,2}, \tilde{y}_{\alpha,2}, \mathsf{node}_2)$ and obtains $0||\tilde{y}_{\alpha,3} = 0||\mathsf{HG.GarbleInp}(\mathsf{hk}, \mathsf{node}_{i,3}, \mathsf{state}_{i,3}; r_{i,3})$. This continues until the final the final evaluation $\mathsf{HEval}(\widetilde{\mathsf{Enc\text{-}Step}}_{i,d}, \tilde{y}_{\alpha,d}, \mathsf{node}_d)$. As $\mathsf{id}_d = \mathsf{id}^*$ and $b_d = \mathsf{pk}^*$, the final circuit outputs $1||\mathsf{Enc}(\mathsf{pk}^*, m)$. The decryption algorithm then decrypts $\mathsf{Enc}(\mathsf{pk}^*, m)$ using PKE secret key sk^*. By the perfect correctness of PKE scheme, the decryption algorithm outputs m. ■

Remark 4.3. *We now describe the need of* flag *in every node of* EncTree*s present in auxilliary information. This* flag *indicates whether a node is a leaf or an internal node. We could consider an alternate construction where the node values are of the form* $a||\mathsf{id}||b \in \{0,1\}^{3\lambda}$ *(without the* flag*), and the step circuits detect any node* $a||\mathsf{id}||b$ *as a leaf by checking whether* $a = 0^\lambda$*. We notice that the construction of [GHM+19] uses this approach and does not satisfy completeness property against computationally unbounded adversaries. A computationally unbounded adversary* $\mathcal{A}$ *can break the completeness property of this alternate approach as follows.* $\mathcal{A}$ *picks an arbitrary target identity* id^* *and makes* $(\mathsf{regtgt}, \mathsf{id}^*)$ *query. The challenger samples PKE pair* $(\mathsf{pk}^*, \mathsf{sk}^*)$*, registers* $(\mathsf{id}^*, \mathsf{pk}^*)$ *and sends* pk^* *back to* $\mathcal{A}$*. At the point, the tree storing (identity, public key) pairs in* aux *consists of only a root node* rt *and a leaf node* $0^\lambda||\mathsf{id}^*||\mathsf{pk}^*$*.* $\mathcal{A}$ *then picks an arbitrary identity* $\mathsf{id} < \mathsf{id}^*$ *and computes* pk *such that* $\mathsf{Hash}(\mathsf{hk}, 0^\lambda||\mathsf{id}||\mathsf{pk}) = 0^\lambda$*.* $\mathcal{A}$ *now makes* $(\mathsf{regnew}, \mathsf{id}, \mathsf{pk})$ *query. The challenger registers* $(\mathsf{id}, \mathsf{pk})$ *pair. Now, the tree consists of root node* $0^\lambda||\mathsf{id}||h$*, a left child* $0^\lambda||\mathsf{id}||\mathsf{pk}$ *and right child* $0^\lambda||\mathsf{id}^*||\mathsf{pk}^*$*. If a message is now encrypted to* id^* *and the ciphertext is decypted using update of* id^**, then the ouput is* $\perp$*.*

4.4 Security

In this section, we prove that the above scheme satisfies soundness of pre/post-Registration Verifiability and Message Hiding properties as defined in Definitions 3.4 to 3.6. We now provide a brief overview of the proofs.

Recall that soundness of pre-registration verifiability property ensures that if a PPT adversary $\mathcal{A}$ can create valid public parameters pp along with a pre-registration proof π that an identity id is not registered, then he will not be able to decrypt any ciphertext ct encrypted for id with non-negligible probability. To provide the proof's intuition, consider the scenario where a cheating accumulator/adversary creates public parameters by inserting $(\mathsf{id}, \mathsf{pk})$ at a wrong leaf location by violating property that the EncTree is to be sorted as per identities. Such an adversary could provide a valid pre-registration proof that the identity is not registered. However, it cannot decrypt the ciphertexts encrypted for the identity. For example, the EncTree generated by adversary has 3 registered identities $\mathsf{id}_1 < \mathsf{id}_2 < \mathsf{id}_3$, has root value $\mathsf{rt} = h_1||\mathsf{id}_3||h_2$ with left subtree containing $\mathsf{id}_1, \mathsf{id}_3$ and right subtree containing id_2. Clearly, the paths to the leaves containing

$\mathsf{id}_1, \mathsf{id}_3$ form a valid pre-registration proof. A ciphertext contains 3 garbled circuits $\{\widetilde{\mathsf{Enc\text{-}Step}}_i\}_i$ and garbling of $\mathsf{Hash}(\mathsf{rt})$. When the garbled circuit $\widetilde{\mathsf{Enc\text{-}Step}}_1$ is run with input as the root value rt, it identifies that id_2 is in left subtree (as $\mathsf{id}_2 < \mathsf{id}_3$) and outputs garbling of h_1. Now, $\widetilde{\mathsf{Enc\text{-}Step}}_2$ can only be run on the left child value of the root node. The garbing values output the garbled circuits would follow the path that is present as part of pre-registration proof, and as a result the final garbled circuit outputs $\perp$ and the adversary cannot decrypt the ciphertext. We formally prove that the scheme satisfies the property, by arguing that when the adversary is forced to generate public parameters along with a pre-registration proof, it cannot distinguish between a real ciphertext and a simulated ciphertext that is generated without using the message.

Soundness of post-registration verifiability property guarantees that if an adversary can create valid public parameters pp along with a post-registration proof π that an identity-key pair $(\mathsf{id}, \mathsf{pk})$ is registered (for an honestly generated pk such that corresponding secret key sk is not revealed to the adversary), then he will not be able decrypt any ciphertext ct encrypted for id. The proof is similar to the proof of pre-registration verifiability, except that the simulated ciphertext is now generated using only PKE encryptions of the message with the identity's public key (the corresponding secret key is unknown to the adversary).

Message Hiding properties guarantees that if the public parameters pp are honestly generated, then a PPT adversary cannot decrypt ciphertexts of unregistered identities, and cannot decrypt ciphertexts of registered identities without the knowledge of their secret keys. We argue that if any RBE scheme satisfies soundness of pre/post-registration verifiability properties along with completeness property, it also satisfies message hiding property. If an $(\mathsf{id}, \mathsf{pk})$ pair is registered as part of pp, then one could also create a valid post-registration proof as per the completeness property. Therefore, as per soundness of post-registration verifiability the ciphertexts meant for id cannot be decrypted with non-negligible probability when secret key corresponding to pk is unknown. If an id is not registered as part of pp, then one could create a valid pre-registration proof as per the completeness property. Therefore, as per soundness of pre-registration verifiability, the ciphertexts meant for id cannot be decrypted with non-negligible probability.

4.4.1 Soundness of Pre-Registration Verifiability

Theorem 4.1. *Assuming* HG *is a secure hash garbling scheme, the above construction satisfies soundness of pre-registration verifiability property (Definition 3.5).*

Proof. We prove the above theorem via a hybrid argument. The first hybrid H_{real} is same as the soundness game for pre-registration verifiability. In this hybrid, the challenger samples a $\mathsf{crs} = \mathsf{hk} \leftarrow \mathsf{HG.Setup}(1^\lambda, 1^{3\lambda+1})$ and sends it to the adversary. The adversary then sends public parameters $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_{i\in[\ell_n]}$, identity id^*, pre-registration proof π, and challenge messages m_0, m_1 to the challenger. The challenger aborts if π is invalid i.e., $\mathsf{PreVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \pi) = 0$. Otherwise, the challenger samples a bit $b \leftarrow \{0,1\}$, encrypts message m_b and sends the ciphertext $\mathsf{ct} = (\{(\mathsf{rt}_i, d_i)\}_i, \{\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}\}_{i,j}, \{\tilde{y}_{i,1}\}_i)$ to the adversary. Concretely, given crs, public parameters pp, identity id^* and message m_b, the challenger first samples $\mathsf{state}_{i,j}, r_{i,j} \leftarrow \{0,1\}^\lambda$ for $i \in [\ell_n], j \in [d_i + 1]$. The challenger then constructs circuits $\mathsf{Enc\text{-}Step}_{i,j}$ for each $i \in [\ell_n], j \in [d_i]$ as in Fig. 2. It then garbles the circuits $\{\mathsf{Enc\text{-}Step}_{i,j}\}_{i,j}$ along with their inputs $\{\mathsf{rt}_i\}_i$ using a hash garbling scheme to obtain $\{\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}\}_{i,j}$ and $\{\tilde{y}_{i,1}\}_i$ respectively. The adversary outputs a bit b', and wins if $b' = b$. We prove that the advantage of any PPT adversary in winning against H_{real} challenger is negligible via a sequence of hybrids.

In each subsequent hybrid, the challenger switches a garbled circuit in the ciphertext with a simulated one. Concretely, in Hybrid $H_{\kappa.\tau}$, the challenger obtains $\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}$ for each $(i,j) \preceq (\kappa, \tau)$ and $\tilde{y}_{i,1}$ for each $(i,1) \preceq (\kappa, \tau)$ using a simulator $\mathsf{HG.Sim}$, whereas the challenger computes $\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}$ for each $(i,j) \succ (\kappa, \tau)$ and $\tilde{y}_{i,1}$ for each $(i,1) \succ (\kappa, \tau)$ using hash garbling scheme as earlier. Here, we say that $(i,j) \prec (k,\ell)$ if $(i < k) \vee (i = k \wedge j < \ell)$. Similarly, we say that $(i,j) \preceq (k,\ell)$ iff $(i,j) = (k,\ell) \vee (i,j) \prec (k,\ell)$. Also, $(i,j) \succ (k,\ell)$ iff $(k,\ell) \prec (i,j)$. We now describe how the challenger obtains $\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}$ for each $(i,j) \preceq (\kappa, \tau)$ using a simulator.

First, we introduce some notations and make some observations. Let the proof π sent by the adversary

be $\pi = (\pi_1, \cdots, \pi_{\ell_n})$, where $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{upr}})$ for each $i \in [\ell_n]$. Both $\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{upr}}$ consists of either an empty string ϵ or a sequence of values in $\{0,1\}^{3\lambda+1}$. For any $i \in [\ell_n], \mathsf{tag} \in \{\mathsf{lwr}, \mathsf{upr}\}$, let $\mathsf{path}_{i,\mathsf{tag}} = (\mathsf{node}_{i,1,\mathsf{tag}}, \mathsf{node}_{i,2,\mathsf{tag}}, \cdots, \mathsf{node}_{i,d_i,\mathsf{tag}})$[17], where each $\mathsf{node}_{i,j,\mathsf{tag}} = \mathsf{flag}_{i,j,\mathsf{tag}} || a_{i,j,\mathsf{tag}} || \mathsf{id}_{i,j,\mathsf{tag}} || b_{i,j,\mathsf{tag}} \in \{0,1\}^{3\lambda+1}$. Let use define $\mathsf{tag}_i = \mathsf{upr}$ if $\mathsf{path}_{i,\mathsf{upr}} \neq \epsilon$. Otherwise, let $\mathsf{tag}_i = \mathsf{lwr}$. If π is a valid pre-registration proof for id^*, then intuitively $\mathsf{path}_{i,\mathsf{tag}_i}$ resembles like a sequence of nodes obtained by performing binary search on id^*. Concretely, by Lemma 4.3, we know that for any index i the following holds. (1) $\mathsf{node}_{i,1,\mathsf{tag}_i} = \mathsf{rt}_i$, (2) $\mathsf{flag}_{i,j,\mathsf{tag}_i} = 1$ iff $j = d_i$, (3) $\mathsf{id}_{i,d_i,\mathsf{tag}_i} \neq \mathsf{id}^*$ and (4) for all $j < d_i$, we have

$$\mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\mathsf{tag}_i}) = \begin{cases} a_{i,j,\mathsf{tag}_i} & \text{if } \mathsf{id}^* \leq \mathsf{id}_{i,j,\mathsf{tag}_i} \\ b_{i,j,\mathsf{tag}_i} & \text{if } \mathsf{id}^* > \mathsf{id}_{i,j+1,\mathsf{tag}_i} \end{cases}.$$

Consider any index $i \in [\ell_n]$. When the circuit $\mathsf{Enc\text{-}Step}_{i,1}$ (with identity id^* embedded in it) is fed with the root value $\mathsf{rt}_i = \mathsf{node}_{i,1,\mathsf{tag}_i}$ as input, the circuit outputs $\mathsf{flag}_{i,1,\mathsf{tag}_i} || \tilde{y}_{i,2} = \mathsf{flag}_{i,1,\mathsf{tag}_i} || \mathsf{HG.GarbleInp}(\mathsf{hk}, a_{i,1,\mathsf{tag}_i}, \mathsf{state}_{i,2}; r_{i,2})$ if $\mathsf{id}^* \leq \mathsf{id}_{i,1,\mathsf{tag}_i}$. Otherwise, the circuit outputs $\mathsf{flag}_{i,1,\mathsf{tag}_i} || \tilde{y}_{i,2} = \mathsf{flag}_{i,1,\mathsf{tag}_i} || \mathsf{HG.GarbleInp}(\mathsf{hk}, b_{i,1,\mathsf{tag}_i}, \mathsf{state}_{i,2}; r_{i,2})$. By the above equation, we know that $\tilde{y}_{i,2} = \mathsf{HG.GarbleInp}(\mathsf{hk}, h_{i,2}, \mathsf{state}_{i,2}; r_{i,2})$, where $h_{i,2} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,2,\mathsf{tag}_i})$. Similarly, when the circuit $\mathsf{Enc\text{-}Step}_{i,2}$ is fed with input $\mathsf{node}_{i,2,\mathsf{tag}_i}$, it outputs $\mathsf{flag}_{i,2,\mathsf{tag}_i} || \tilde{y}_{i,3} = \mathsf{flag}_{i,2,\mathsf{tag}_i} || \mathsf{HG.GarbleInp}(\mathsf{hk}, h_{i,3}, \mathsf{state}_{i,3}; r_{i,3})$, where $h_{i,3} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,3,\mathsf{tag}_i})$. On repeating this process, for each $j < d_i$, the circuit $\mathsf{Enc\text{-}Step}_{i,j}$ outputs $\mathsf{flag}_{i,j,\mathsf{tag}_i} || \tilde{y}_{i,j+1} = \mathsf{flag}_{i,j,\mathsf{tag}_i} || \mathsf{HG.GarbleInp}(\mathsf{hk}, h_{i,j+1}, \mathsf{state}_{i,j+1}; r_{i,j+1})$, where $h_{i,j+1} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\mathsf{tag}_i})$. The output of the final circuit $\mathsf{Enc\text{-}Step}_{i,d_i}$ is $1 || \perp$ as $\mathsf{id}_{i,d_i \mathsf{tag}_i} \neq \mathsf{id}^*$.

Based on the above observations, we now describe how the challenger obtains $\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}$ for each $(i,j) \preceq (\kappa,\tau)$ using a simulator. In order to simulate circuit $\widetilde{\mathsf{Enc\text{-}Step}}_{\kappa,\tau}$, the challenger first samples $\mathsf{state}_{\kappa,\tau+1}, r_{\kappa,\tau+1} \leftarrow \{0,1\}^\lambda$ and computes $\mathsf{flag}_{\kappa,\tau} || \tilde{y}_{\kappa,\tau+1}$ as follows.

$$\mathsf{flag}_{\kappa,\tau} || \tilde{y}_{\kappa,\tau+1} = \begin{cases} 0 || \mathsf{HG.GarbleInp}(\mathsf{hk}, h_{\kappa,\tau+1}, \mathsf{state}_{\kappa,\tau+1}; r_{\kappa,\tau+1}) & \text{if } \tau \neq d_\kappa \\ 1 || \perp & \text{if } \tau = d_\kappa \end{cases}$$

where $h_{\kappa,\tau+1} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{\kappa,\tau+1,\mathsf{tag}_\kappa})$. The challenger then runs the simulator using $\tilde{y}_{\kappa,\tau+1}$ as output and obtains $(\widetilde{\mathsf{Enc\text{-}Step}}_{\kappa,\tau}, \tilde{y}_{\kappa,\tau}) \leftarrow \mathsf{HG.Sim}(\mathsf{hk}, \mathsf{node}_{\kappa,\tau,\mathsf{tag}_\kappa}, 1^L, \mathsf{flag}_{\kappa,\tau} || \tilde{y}_{\kappa,\tau+1})$. Here L is the length of the step circuit described in Fig. 2. The challenger then runs the simulator again using $\tilde{y}_{\kappa,\tau}$ as output and obtains $(\widetilde{\mathsf{Enc\text{-}Step}}_{\kappa,\tau-1}, \tilde{y}_{\kappa,\tau-1}) \leftarrow \mathsf{HG.Sim}(\mathsf{hk}, \mathsf{node}_{\kappa,\tau-1,\mathsf{tag}_\kappa}, 1^L, 0 || \tilde{y}_{\kappa,\tau})$. The challenger continues the process and obtains $\widetilde{\mathsf{Enc\text{-}Step}}_{\kappa,j}, \tilde{y}_{\kappa,j}$ for each $j \leq \tau$.

For any $i < \kappa$, the challenger first sets $\tilde{y}_{i,d_i+1} = \perp$. It then runs the simulator $(\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}, \tilde{y}_{i,j}) \leftarrow \mathsf{HG.Sim}(\mathsf{hk}, \mathsf{node}_{i,j,\mathsf{tag}_i}, 1^L, \mathsf{flag}_{i,j+1} || \tilde{y}_{i,j+1})$ for each $j \leq d_i$. Here, L is the length of the step circuit described in Fig. 2 and $\mathsf{flag}_{i,j+1} = 1$ iff $j = d_i$. The challenger obtains $\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}$ for each $(i,j) \preceq (\kappa,\tau)$ and $\tilde{y}_{i,1}$ for each $(i,1) \preceq (\kappa,\tau)$ by running simulator in this way. Using the security of hash garbling scheme, we prove that every 2 adjacent hybrids are computationally indistinguishable.

In the final hybrid $H_{\ell_n.d_{\ell_n}}$, the challenger produces the entire ciphertext using a simulator without even sampling b. Clearly, the advantage of any adversary in this hybrid is 0. By the above indistinguishability argument, the advantage of any PPT adversary in the original game H_{real} is negligible.

We now provide a formal description of the hybrids. Note that the numbering of hybrids depends on the size of public parameters sent by the adversary. Let us define the set S to be $\{(i,j) : i \in [\ell_n], j \in [d_i]\}$, when pp sent by the adversary is $\{(\mathsf{rt}_i, d_i)\}_{i \in [\ell_n]}$.

Hybrid H_{real}: This is same as original soundness game for pre-registration verifiability property.

1. The challenger first samples a hash key $\mathsf{hk} \leftarrow \mathsf{HG.Setup}(1^\lambda, 1^{3\lambda+1})$, sets $\mathsf{crs} = \mathsf{hk}$ and sends it to the adversary.

[17] For the sake of simplicity, we assume the paths output by the adversary are of length d_i. The proof still works even if the adversary outputs paths of different lengths.

2. The adversary sends an identity id^*, public parameters pp, a proof π and a pair of messages m_0, m_1 to the challenger. The challenger aborts and the adversary looses if $\mathsf{PreVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \pi) = 0$.
3. Let public parameters pp be $\{(\mathsf{rt}_i, d_i)\}_{i\in[\ell_n]}$ and proof π be $(\pi_1, \cdots, \pi_{\ell_n})$. For any $i \in [\ell_n]$, let $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{upr}})$. For any $i \in [\ell_n]$ and $\mathsf{tag} \in \{\mathsf{lwr}, \mathsf{upr}\}$, let $\mathsf{path}_{i,\mathsf{tag}}$ be sequence of values $(\mathsf{node}_{i,1,\mathsf{tag}}, \mathsf{node}_{i,2,\mathsf{tag}}, \cdots, \mathsf{node}_{i,d_i,\mathsf{tag}})$.[18]
4. Let $S = \{(i,j) : i \in [\ell_n], j \in [d_i]\}$ and $S' = \{(i,j) : i \in [\ell_n], j \in [d_i + 1]\}$. The challenger samples a bit $b \leftarrow \{0,1\}$.
5. It then samples $\mathsf{state}_{i,j} \leftarrow \{0,1\}^\lambda, r_{i,j} \leftarrow \{0,1\}^\lambda$ for each $(i,j) \in S'$.
6. For each $(i,j) \in S$, the challenger defines the circuit $\mathsf{Enc\text{-}Step}_{i,j}$ as in Fig. 2 with hash key hk, state $\mathsf{state}_{i,j+1}$, randomness $r_{i,j+1}$, identity id^*, message m_b hardwired in it. It then garbles the circuit $\mathsf{Enc\text{-}Step}_{i,j}$ i.e., $\widetilde{\mathsf{Enc\text{-}Step}}_{i,j} \leftarrow \mathsf{HG.GarbleCkt}(\mathsf{hk}, \mathsf{Enc\text{-}Step}_{i,j}, \mathsf{state}_{i,j})$.
   For each $i \in [\ell_n]$, the challenger computes $\tilde{y}_{i,1} = \mathsf{HG.GarbleInp}(\mathsf{hk}, h_{i,1}, \mathsf{state}_{i,1}; r_{i,1})$, where $h_{i,1} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{rt}_i)$.
7. Finally, the challenger sends ciphertext $\mathsf{ct} = (\{(\mathsf{rt}_i, d_i)\}_{i\in[\ell_n]}, \{\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}\}_{(i,j)\in S}, \{\tilde{y}_{i,1}\}_{i\in[\ell_n]})$ to the adversary.
8. The adversary outputs a bit b'. The adversary wins if $b' = b$.

Hybrid $H_{\kappa.\tau}$ ($(\kappa,\tau) \in S$): In this hybrid, for every $(i,j) \preceq (\kappa,\tau)$, the challenger computes $\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}$ using a simulator. The changes from the previous hybrid are highlighted in red.

5. It then samples $\mathsf{state}_{i,j} \leftarrow \{0,1\}^\lambda, r_{i,j} \leftarrow \{0,1\}^\lambda$ for each $(i,j) \in S'$ s.t. $(i,j) \succ (\kappa,\tau)$.
6. For each $(i,j) \in S$ s.t. $(i,j) \succ (\kappa,\tau)$, the challenger defines the circuit $\mathsf{Enc\text{-}Step}_{i,j}$ as in Fig. 2 with hash key hk, state $\mathsf{state}_{i,j+1}$, randomness $r_{i,j+1}$, identity id^*, message m_b hardwired in it. It then garbles the circuit $\mathsf{Enc\text{-}Step}_{i,j}$ i.e., $\widetilde{\mathsf{Enc\text{-}Step}}_{i,j} \leftarrow \mathsf{HG.GarbleCkt}(\mathsf{hk}, \mathsf{Enc\text{-}Step}_{i,j}, \mathsf{state}_{i,j})$.
   For each i s.t $(i,1) \succ (\kappa,\tau)$, the challenger computes $\tilde{y}_{i,1} = \mathsf{HG.GarbleInp}(\mathsf{hk}, h_{i,1}, \mathsf{state}_{i,1}; r_{i,1})$, where $h_{i,1} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{rt}_i)$.
   For any i, let use define $\mathsf{tag}_i = \mathsf{upr}$ if $\mathsf{path}_{i,\mathsf{upr}} \neq \epsilon$. Otherwise, let $\mathsf{tag}_i = \mathsf{lwr}$. For each i s.t. $(i,d_i) \preceq (\kappa,\tau)$, the challenger sets $\tilde{y}_{i,d_i+1} = \perp$. If $\tau \neq d_\kappa$, the challenger computes $x = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{\kappa,\tau+1,\mathsf{tag}_i})$ and sets $\tilde{y}_{\kappa,\tau+1} = \mathsf{HG.GarbleInp}(\mathsf{hk}, x, \mathsf{state}_{\kappa,\tau+1}; r_{\kappa,\tau+1})$.
   For each $(i,j) \in S$ s.t $(i,j) \preceq (\kappa,\tau)$, the challenger simulates the garbling of $\mathsf{Enc\text{-}Step}_{i,j}$ as $(\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}, \tilde{y}_{i,j}) \leftarrow \mathsf{HG.Sim}(\mathsf{hk}, \mathsf{node}_{i,j,\mathsf{tag}_i}, 1^L, \mathsf{flag}_{i,j+1} || \tilde{y}_{i,j+1})$. Here L is the length of circuit $\mathsf{Enc\text{-}Step}_{i,j}$ defined in Fig. 2 and $\mathsf{flag}_{i,j+1} = 1$ iff $j = d_i$. [19]

For the sake of simplicity, we define Hybrid $H_{1.0}$ same as Hybrid H_{real}. Note that hybrid H_{real} ($H_{1.0}$) is same as the soundness game for pre-registration verifiability. In this hybrid, the challenger generates the ciphertext by garbing circuits $\mathsf{Enc\text{-}Step}_{i,j}$ and their inputs rt_i using hash garbling scheme. In the final hybrid $H_{\ell_n.d_{\ell_n}}$, the challenger generates the ciphertext using simulator given only public parameters and length of the step circuits. The ciphertext in this final hybrid is independent of b. Consequently, the advantage of any adversary in Hybrid $H_{\ell_n.d_{\ell_n}}$ is 0. We prove that hybrids H_{real} and $H_{\ell_n.d_{\ell_n}}$ are computationally indstinguishable via a sequence of intermediate hybrids $H_{\kappa.\tau}$. For any PPT adversary $\mathcal{A}$, let the advantage of $\mathcal{A}$ in Hybrid H_s be $\mathsf{Adv}_s^{\mathcal{A}} = \Pr[b' = b] - 1/2$. For any $\kappa \in [\ell_n], \tau \in [d_\kappa]$, let $(\tilde{\kappa}, \tilde{\tau})$ be its preceding tuple

$$\text{i.e., } (\tilde{\kappa}, \tilde{\tau}) = \begin{cases} (\kappa, \tau - 1) & \text{if } \tau > 1 \\ (\kappa - 1, d_{\kappa-1}) & \text{if } \kappa > 1 \wedge \tau = 1 \\ (1, 0) & \text{if } (\kappa,\tau) = (1,1) \end{cases}.$$

[18] To simplify the notations, we assume that the paths $\mathsf{path}_{i,\mathsf{lwr}}$ and $\mathsf{path}_{i,\mathsf{upr}}$ output by the adversary have length d_i. With appropriate changes, the proof works even if length of the paths is smaller than d_i.

[19] Note that simulating $\tilde{y}_{i,j}$ requires $\tilde{y}_{i,j+1}$. As a result, for each i, the challenger runs the simulator in the decreasing order of j.

Lemma 4.6. *Assuming* HG *is a secure hash garbling scheme, for every admissible*[20] *PPT adversary* $\mathcal{A}$*, there exists a negligible function* $negl(\cdot)$ *such that for any* $\lambda \in \mathbb{N}$ *and any* $(\kappa, \tau) \in S$*, we have* $|\mathsf{Adv}^{\mathcal{A}}_{\kappa.\tau} - \mathsf{Adv}^{\mathcal{A}}_{\tilde{\kappa}.\tilde{\tau}}| \leq negl(\lambda)$.

Proof. Suppose there exists an admissible PPT adversary $\mathcal{A}$, indices $(\kappa, \tau) \in S$ such that $|\mathsf{Adv}^{\mathcal{A}}_{\kappa.\tau} - \mathsf{Adv}^{\mathcal{A}}_{\tilde{\kappa}.\tilde{\tau}}|$ is a non-negligible value ϵ. We construct a non-uniform reduction algorithm $\mathcal{B}$ that breaks the security of hash garbling scheme.

The hash garbling game challenger $\mathcal{C}$ first sends a hash key hk to the reduction algorithm $\mathcal{B}$. $\mathcal{B}$ sets $\mathsf{crs} = \mathsf{hk}$ and sends it to the adversary $\mathcal{A}$. The adversary sends an identity id^*, public parameters pp, a proof π and a pair of messages m_0, m_1 to $\mathcal{B}$. If $\mathsf{PreVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \pi) = 0$, $\mathcal{B}$ aborts and outputs a random bit in hash garbling game. Otherwise let $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_{i \in [\ell_n]}$, $\pi = (\pi_1, \cdots, \pi_{\ell_n})$. For any $i \in [\ell_n]$, let $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{upr}})$. For any $i \in [\ell_n]$ and $\mathsf{tag} \in \{\mathsf{lwr}, \mathsf{upr}\}$, let $\mathsf{path}_{i,\mathsf{tag}}$ be sequence of values $(\mathsf{node}_{i,1,\mathsf{tag}}, \mathsf{node}_{i,2,\mathsf{tag}}, \cdots, \mathsf{node}_{i,d_i,\mathsf{tag}})$[18]. Let $S = \{(i,j) : i \in [\ell_n], j \in [d_i]\}$ and $S' = \{(i,j) : i \in [\ell_n], j \in [d_i + 1]\}$. For any i, let use define $\mathsf{tag}_i = \mathsf{upr}$ if $\mathsf{path}_{i,\mathsf{upr}} \neq \epsilon$. Otherwise, let $\mathsf{tag}_i = \mathsf{lwr}$. $\mathcal{B}$ then samples a bit $b \leftarrow \{0,1\}$. For each $(i,j) \in S'$ s.t. $(i,j) \succ (\kappa, \tau)$, $\mathcal{B}$ samples $\mathsf{state}_{i,j} \leftarrow \{0,1\}^\lambda, r_{i,j} \leftarrow \{0,1\}^\lambda$. For each $(i,j) \in S$ s.t. $(i,j) \succeq (\kappa, \tau)$, $\mathcal{B}$ creates the circuit $\mathsf{Enc\text{-}Step}_{i,j}$ as in Fig. 2 with identity id^*, message m_b hardwired in it. $\mathcal{B}$ then sends $\mathsf{Enc\text{-}Step}_{\kappa,\tau}, \mathsf{node}_{\kappa,\tau,\mathsf{tag}_\kappa}$ to the challenger $\mathcal{C}$. $\mathcal{C}$ samples a bit $\gamma \leftarrow \{0,1\}$. If $\gamma = 0$, $\mathcal{C}$ samples $\mathsf{state} \leftarrow \{0,1\}^\lambda$ and computes $h = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{\kappa,\tau,\mathsf{tag}_\kappa})$, $\widetilde{\mathsf{Enc\text{-}Step}}_{\kappa,\tau} \leftarrow \mathsf{HG.GarbleCkt}(\mathsf{hk}, \mathsf{Enc\text{-}Step}_{\kappa,\tau}, \mathsf{state})$, $\tilde{y}_{\kappa,\tau} \leftarrow \mathsf{HG.GarbleInp}(\mathsf{hk}, h, \mathsf{state})$. Otherwise $\mathcal{C}$ computes $x = \mathsf{Enc\text{-}Step}_{\kappa,\tau}(\mathsf{node}_{\kappa,\tau,\mathsf{tag}_\kappa})$ and runs simulator $(\widetilde{\mathsf{Enc\text{-}Step}}_{\kappa,\tau}, \tilde{y}_{\kappa,\tau}) \leftarrow \mathsf{HG.Sim}(\mathsf{hk}, \mathsf{node}_{\kappa,\tau,\mathsf{tag}_\kappa}, 1^{|\mathsf{Enc\text{-}Step}_{\kappa,\tau}|}, x)$. $\mathcal{C}$ sends $\widetilde{\mathsf{Enc\text{-}Step}}_{\kappa,\tau}, \tilde{y}_{\kappa,\tau}$ to $\mathcal{B}$. For each $(i,j) \in S$ s.t. $(i,j) \succ (\kappa, \tau)$, $\mathcal{B}$ garbles the circuit $\mathsf{Enc\text{-}Step}_{i,j}$ to obtain $\widetilde{\mathsf{Enc\text{-}Step}}_{i,j} \leftarrow \mathsf{HG.GarbleCkt}(\mathsf{hk}, \mathsf{Enc\text{-}Step}_{i,j}, \mathsf{state}_{i,j})$. For each i s.t $(i,1) \succ (\kappa, \tau)$, $\mathcal{B}$ garbles the input rt_i and obtains $\tilde{y}_{i,1} = \mathsf{HG.GarbleInp}(\mathsf{hk}, \mathsf{rt}_i, \mathsf{state}_{i,1}; r_{i,1})$. For each $(i,j) \in S$ s.t. $(i,j) \prec (\kappa, \tau)$, $\mathcal{B}$ runs the simulator and computes $\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}, \tilde{y}_{i,j}$ as in Hybrid $H_{\kappa.\tau}$. Finally, $\mathcal{B}$ sends the ciphertext $\mathsf{ct} = (\{(\mathsf{rt}_i, d_i)\}_i, \{\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}\}_{i,j}, \{\tilde{y}_{i,1}\}_i)$ to $\mathcal{A}$. $\mathcal{A}$ outputs a bit b'. If $b = b'$, $\mathcal{B}$ outputs 1 in the hash garbling game. Otherwise $\mathcal{B}$ outputs 0 in the hash garbling game.

We now analyze the advantage of $\mathcal{B}$ in hash garbling game. As the adversary $\mathcal{A}$ is admissible, $\mathsf{PreVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \pi) = 1$ and $\mathcal{B}$ does not abort. By Lemma 4.3, when $\mathsf{PreVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \pi) = 1$, the path $\mathsf{path}_{\kappa,\mathsf{tag}_\kappa}$ resembles a sequence of nodes obtain by performing binary search on id^* in a tree with root rt_κ. Consequently, when the challenger runs step circuit $\mathsf{Enc\text{-}Step}_{\kappa,\tau}$ on input $\mathsf{node}_{\kappa,\tau,\mathsf{tag}_\kappa}$, the result is $0||\mathsf{node}_{\kappa,\tau+1,\mathsf{tag}_\kappa}$ if $\tau < d_\kappa$, and $1||\perp$ if $\tau = d_\kappa$. As a result, if $\gamma = 0$, then $\mathcal{B}$ emulates Hybrid $H_{\tilde{\kappa}.\tilde{\tau}}$ challenger to $\mathcal{A}$, and if $\gamma = 1$, then $\mathcal{B}$ emulates Hybrid $H_{\kappa.\tau}$ challenger to $\mathcal{A}$. By our assumption, $|\mathsf{Adv}^{\mathcal{A}}_{\kappa.\tau} - \mathsf{Adv}^{\mathcal{A}}_{\tilde{\kappa}.\tilde{\tau}}|$ is non-negligible. Therefore, $\mathcal{B}$ has a non-negligible advantage in predicting γ.

■

By the above lemma and triangle inequality, the advantage of any PPT adversary in Hybrid H_{real} is negligible. Therefore, our construction satisfies soundness for pre-registration verifiability property. ■

4.4.2 Soundness of Post-Registration Verifiability

Theorem 4.2. *Assuming* HG *is a secure hash garbling scheme and* PKE *is a secure public key encryption scheme, the above construction satisfies soundness of post-registration verifiability property (Definition 3.5).*

We prove the above theorem via a hybrid argument. In the first hybrid H_b ($b \in \{0,1\}$), the challenger samples a crs, a PKE pair $(\mathsf{pk}, \mathsf{sk})$ and sends a $(\mathsf{crs}, \mathsf{pk})$ to the adversary. The adversary then sends public parameters $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_{i \in [\ell_n]}$, identity id^*, post-registration proof π, and challenge messages m_0, m_1 to the challenger. The challenger aborts if π is invalid i.e., $\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}, \pi) = 0$. Otherwise, the challenger encrypts message m_b and sends the ciphertext $\mathsf{ct} = (\{(\mathsf{rt}_i, d_i)\}_i, \{\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}\}_{i,j}, \{\tilde{y}_{i,1}\}_i)$ to the

[20] A PPT adversary in the soundness game is said to be admissible if it always produces valid proof π.

adversary. Concretely, given crs, public parameters pp, identity id^* and message m_b, the challenger first samples $\mathsf{state}_{i,j}, r_{i,j} \leftarrow \{0,1\}^\lambda$ for each $i \in [\ell_n], j \in [d_i+1]$. It then constructs circuits $\mathsf{Enc\text{-}Step}_{i,j}$ for each $i \in [\ell_n], j \in [d_i]$ as in Fig. 2. The challenger then garbles the circuits $\{\mathsf{Enc\text{-}Step}_{i,j}\}_{i,j}$ along with their inputs $\{\mathsf{rt}_i\}_i$ using a hash garbling scheme to obtain $\{\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}\}_{i,j}$ and $\{\tilde{y}_{i,1}\}_i$ respectively. Finally, the adversary outputs bit b'. We prove that for any PPT adversary, $\Pr[b'=1]$ in Hybrids H_0 and H_1 are negligibly close via a sequence of hybrids.

In each subsequent hybrid, the challenger switches a garbled circuit in the ciphertext with a simulated one. Concretely, in Hybrid $H_{\kappa.\tau}$, the challenger obtains $\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}$ for each $(i,j) \preceq (\kappa,\tau)$ and $\tilde{y}_{i,1}$ for each $(i,1) \preceq (\kappa,\tau)$ using a simulator. The challenger computes $\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}$ for each $(i,j) \succ (\kappa,\tau)$ and $\tilde{y}_{i,1}$ for each $(i,1) \succ (\kappa,\tau)$ using hash garbling scheme as earlier. Here, we say that $(i,j) \prec (k,\ell)$ if $(i<k) \vee (i=k \wedge j<\ell)$. Similarly, we say that $(i,j) \preceq (k,\ell)$ iff $(i,j) = (k,\ell) \vee (i,j) \prec (k,\ell)$. Also, $(i,j) \succ (k,\ell)$ iff $(k,\ell) \prec (i,j)$. We now describe how the challenger obtains $\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}$ for each $(i,j) \preceq (\kappa,\tau)$ using a simulator.

First, we introduce some notations and make some observations. Let the proof π output by the adversary be $\pi = (\pi_1, \cdots, \pi_{\ell_n}, \alpha)$. For any $i < \alpha$, let $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{upr}})$. For any $i \geq \alpha$, let $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{mid}}, \mathsf{path}_{i,\mathsf{upr}})$. For any $i \in [\ell_n], \mathsf{tag} \in \{\mathsf{lwr}, \mathsf{mid}, \mathsf{upr}\}$, let $\mathsf{path}_{i,\mathsf{tag}} = (\mathsf{node}_{i,1,\mathsf{tag}}, \mathsf{node}_{i,2,\mathsf{tag}}, \cdots, \mathsf{node}_{i,d_i,\mathsf{tag}})$[21], where each $\mathsf{node}_{i,j,\mathsf{tag}} = \mathsf{flag}||a_{i,j,\mathsf{tag}}||\mathsf{id}_{i,j,\mathsf{tag}}||b_{i,j,\mathsf{tag}} \in \{0,1\}^{3\lambda+1}$. For any $i \in [\ell_n]$, let us define

$$\mathsf{tag}_i = \begin{cases} \mathsf{upr} & \text{if } i < \alpha \wedge \mathsf{path}_{i,\mathsf{upr}} \neq \epsilon \\ \mathsf{lwr} & \text{if } i < \alpha \wedge \mathsf{path}_{i,\mathsf{upr}} = \epsilon \\ \mathsf{mid} & \text{Otherwise} \end{cases}.$$

Suppose π is a valid post-registration proof for id^*. We know that for any i, $\mathsf{path}_{i,\mathsf{tag}_i}$ looks like a sequence of nodes obtained by performing binary search on id^*. Concretely, by Lemmas 4.3 and 4.4, we know that for any index i, the following holds. (1) $\mathsf{node}_{i,1,\mathsf{tag}_i} = \mathsf{rt}_i$, (2) $\mathsf{flag}_{i,j,\mathsf{tag}_i} = 1$ iff $j = d_i$, (3) $\mathsf{id}_{i,d_i,\mathsf{tag}_i} \neq \mathsf{id}^*$ if $i < \alpha$, (4) $\mathsf{id}_{i,d_i,\mathsf{tag}_i} = \mathsf{id}^*$, $b_{i,d_i,\mathsf{tag}_i} = \mathsf{pk}^*$ if $i \geq \alpha$, and (5) for all $j < d_i$,

$$\mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\mathsf{tag}_i}) = \begin{cases} a_{i,j,\mathsf{tag}_i} & \text{if } \mathsf{id}^* \leq \mathsf{id}_{i,j,\mathsf{tag}_i} \\ b_{i,j,\mathsf{tag}_i} & \text{if } \mathsf{id}^* > \mathsf{id}_{i,j,\mathsf{tag}_i} \end{cases}.$$

Consider any index $i \in [\ell_n]$. When the circuit $\mathsf{Enc\text{-}Step}_{i,1}$ (with identity id^* embedded in it) is fed with the root value $\mathsf{rt}_i = \mathsf{node}_{i,1,\mathsf{tag}_i}$ as input, the circuit outputs

$$\begin{aligned} \mathsf{flag}_{i,1,\mathsf{tag}_i}||\tilde{y}_{i,2} &= \begin{cases} \mathsf{flag}_{i,1,\mathsf{tag}_i}||\mathsf{HG.GarbleInp}(\mathsf{hk}, a_{i,1,\mathsf{tag}_i}, \mathsf{state}_{i,2}; r_{i,2}) & \text{if } \mathsf{id}^* \leq \mathsf{id}_{i,1,\mathsf{tag}_i} \\ \mathsf{flag}_{i,1,\mathsf{tag}_i}||\mathsf{HG.GarbleInp}(\mathsf{hk}, b_{i,1,\mathsf{tag}_i}, \mathsf{state}_{i,2}; r_{i,2}) & \text{Otherwise} \end{cases} \\ &= \mathsf{flag}_{i,1,\mathsf{tag}_i}||\mathsf{HG.GarbleInp}(\mathsf{hk}, h_{i,2}, \mathsf{state}_{i,2}; r_{i,2}) \end{aligned}$$

where $h_{i,2} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,2,\mathsf{tag}_i})$. Similarly, when the circuit $\mathsf{Enc\text{-}Step}_{i,2}$ is fed with input $\mathsf{node}_{i,2,\mathsf{tag}_i}$, it outputs $\mathsf{flag}_{i,2,\mathsf{tag}_i}||\tilde{y}_{i,3} = \mathsf{flag}_{i,2,\mathsf{tag}_i}||\mathsf{HG.GarbleInp}(\mathsf{hk}, h_{i,3}, \mathsf{state}_{i,3}; r_{i,3})$, where $h_{i,3} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,3,\mathsf{tag}_i})$. On repeating this process, for each $j < d_i$, the circuit $\mathsf{Enc\text{-}Step}_{i,j}$ outputs $\mathsf{flag}_{i,j,\mathsf{tag}_i}||\tilde{y}_{i,j+1} = \mathsf{flag}_{i,j,\mathsf{tag}_i}||\mathsf{HG.GarbleInp}(\mathsf{hk}, h_{i,j+1}, \mathsf{state}_{i,j+1}; r_{i,j+1})$, where $h_{i,j+1} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{i,j+1,\mathsf{tag}_i})$. The output of the final circuit $\mathsf{Enc\text{-}Step}_{i,d_i}$ is $1||\perp$ if $i < \alpha$, and $1||\mathsf{Enc}(\mathsf{pk}, m)$ if $i \geq \alpha$.

Based on the above observations, we now describe how the challenger obtains $\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}$ for each $(i,j) \preceq (\kappa,\tau)$ using a simulator. In order to simulate circuit $\widetilde{\mathsf{Enc\text{-}Step}}_{\kappa,\tau}$, the challenger first samples $\mathsf{state}_{\kappa,\tau+1}, r_{\kappa,\tau+1} \leftarrow \{0,1\}^\lambda$ and computes $\mathsf{flag}_{\kappa,\tau}||\tilde{y}_{\kappa,\tau+1}$ as follows.

$$\mathsf{flag}_{\kappa,\tau}||\tilde{y}_{\kappa,\tau+1} = \begin{cases} 0||\mathsf{HG.GarbleInp}(\mathsf{hk}, h_{\kappa,\tau+1}, \mathsf{state}_{\kappa,\tau+1}; r_{\kappa,\tau+1}) & \text{if } \tau \neq d_\kappa \\ 1||\perp & \text{if } \tau = d_\kappa \wedge \kappa < \alpha \\ 1||\mathsf{Enc}(\mathsf{pk}, m_b) & \text{if } \tau = d_\kappa \wedge \kappa \geq \alpha \end{cases}$$

[21] To simplify the notations, we assume that the paths $\mathsf{path}_{i,\mathsf{lwr}}$, $\mathsf{path}_{i,\mathsf{mid}}$ and $\mathsf{path}_{i,\mathsf{upr}}$ output by the adversary have length d_i. With appropriate changes, the proof works even if length of the paths is smaller than d_i.

where $h_{\kappa,\tau+1} = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{\kappa,\tau+1,\mathsf{tag}_i})$. The challenger then runs the simulator using $\tilde{y}_{\kappa,\tau+1}$ as output and obtains $(\widetilde{\mathsf{Enc\text{-}Step}}_{\kappa,\tau}, \tilde{y}_{\kappa,\tau}) \leftarrow \mathsf{HG.Sim}(\mathsf{hk}, \mathsf{node}_{\kappa,\tau,\mathsf{tag}_i}, 1^L, \mathsf{flag}_{\kappa,\tau}||\tilde{y}_{\kappa,\tau+1})$. Here L is the length of the step circuit described in Fig. 2. The challenger then runs the simulator again using $\tilde{y}_{\kappa,\tau}$ as output and obtains $(\widetilde{\mathsf{Enc\text{-}Step}}_{\kappa,\tau-1}, \tilde{y}_{\kappa,\tau-1}) \leftarrow \mathsf{HG.Sim}(\mathsf{hk}, \mathsf{node}_{\kappa,\tau-1,\mathsf{tag}_i}, 1^L, 0||\tilde{y}_{\kappa,\tau})$. Similarly, for each $j < \tau$, the challenger samples $(\widetilde{\mathsf{Enc\text{-}Step}}_{\kappa,j}, \tilde{y}_{\kappa,j}) \leftarrow \mathsf{HG.Sim}(\mathsf{hk}, \mathsf{node}_{\kappa,j,\mathsf{tag}_i}, 1^L, 0||\tilde{y}_{\kappa,j+1})$.

For any $i < \kappa$, the challenger first sets $\tilde{y}_{i,d_i+1} = \begin{cases} \bot & \text{if } i < \alpha \\ \mathsf{Enc}(\mathsf{pk}, m_b) & \text{Otherwise} \end{cases}$. It then runs the simulator $(\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}, \tilde{y}_{i,j}) \leftarrow \mathsf{HG.Sim}(\mathsf{hk}, \mathsf{node}_{i,j,\mathsf{tag}_i}, 1^L, \mathsf{flag}_{i,j}||\tilde{y}_{i,j+1})$ for each $j \leq d_\kappa$. Here, L is the length of the step circuit described in Fig. 2 and $\mathsf{flag}_{i,j+1} = 1$ iff $j = d_i$. The challenger obtains $\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}$ for each $(i,j) \preceq (\kappa,\tau)$ and $\tilde{y}_{i,1}$ for each $(i,1) \preceq (\kappa,\tau)$ by running simulator in this way. Using the security of hash garbling scheme, we prove that every 2 adjacent hybrids are computationally indistinguishable.

In the final hybrid $H_{b.\ell_n.d_{\ell_n}}$, the challenger produces the entire ciphertext using a simulator given PKE encryptions of message m_b. By semantic security of the underlying PKE system, we know that PKE encryptions of m_0 are computationally indistinguishable from PKE encryptions of m_1. Using this, we prove that hybrids $H_{0.\ell_n.d_{\ell_n}}$ and $H_{1.\ell_n.d_{\ell_n}}$ are computationally indistinguishable. By combining the above indistinguishability arguments, we prove that Hybrids H_0 and H_1 are computationally indistinguishable.

We now provide a formal description of the hybrids. Note that the numbering of hybrids depends on the size of public parameters sent by the adversary. Let us define the set S to be $\{(i,j) : i \in [\ell_n], j \in [d_i]\}$, when pp sent by the adversary is $\{(\mathsf{rt}_i, d_i)\}_{i\in[\ell_n]}$.

Proof. We prove the above scheme using a sequence of following hybrids.

Hybrid H_b ($b \in \{0,1\}$): This hybrid is same as the original soundness game in which the challenger always encrypts message m_b.

1. The challenger first samples a hash key $\mathsf{hk} \leftarrow \mathsf{HG.Setup}(1^\lambda, 1^{3\lambda+1})$, a PKE key pair $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$, sets $\mathsf{crs} = \mathsf{hk}$ and sends $(\mathsf{crs}, \mathsf{pk}^*)$ to the adversary.
2. The adversary sends an identity id^*, public parameters pp, a proof π and a pair of messages m_0, m_1 to the challenger. The challenger aborts and the adversary looses if $\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*, \pi) = 0$.
3. Let public parameters $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_{i\in[\ell_n]}$ and proof $\pi = (\pi_1, \cdots, \pi_{\ell_n}, \alpha)$. Let $S = \{(i,j) : i \in [\ell_n], j \in [d_i]\}$ and $S' = \{(i,j) : i \in [\ell_n], j \in [d_i + 1]\}$.
4. For each $i < \alpha$, let $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{upr}})$. Similarly, for each $i \geq \alpha$, let $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{mid}}, \mathsf{path}_{i,\mathsf{upr}})$. For any $i \in [\ell_n]$ and $\mathsf{tag} \in \{\mathsf{lwr}, \mathsf{mid}, \mathsf{upr}\}$, let $\mathsf{path}_{i,\mathsf{tag}}$ be a sequence of values $(\mathsf{node}_{i,1,\mathsf{tag}}, \mathsf{node}_{i,2,\mathsf{tag}}, \cdots, \mathsf{node}_{i,d_i,\mathsf{tag}})$.[21]. For any $i \in [\ell_n]$, let $\mathsf{tag}_i = \begin{cases} \mathsf{upr} & \text{if } i < \alpha \\ \mathsf{mid} & \text{Otherwise} \end{cases}$.
5. The challenger samples $\mathsf{state}_{i,j} \leftarrow \{0,1\}^\lambda, r_{i,j} \leftarrow \{0,1\}^\lambda$ for each $(i,j) \in S'$.
6. For each $(i,j) \in S$, the challenger defines the circuit $\mathsf{Enc\text{-}Step}_{i,j}$ as in Fig. 2 with hash key hk, state $\mathsf{state}_{i,j+1}$, randomness $r_{i,j+1}$, identity id^*, message m_b hardwired in it. It then garbles the circuit $\mathsf{Enc\text{-}Step}_{i,j}$ i.e., $\widetilde{\mathsf{Enc\text{-}Step}}_{i,j} \leftarrow \mathsf{HG.GarbleCkt}(\mathsf{hk}, \mathsf{Enc\text{-}Step}_{i,j}, \mathsf{state}_{i,j})$.
   For each $i \in [\ell_n]$, the challenger computes $\tilde{y}_{i,1} = \mathsf{HG.GarbleInp}(\mathsf{hk}, \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{rt}_i), \mathsf{state}_{i,1}; r_{i,1})$.
7. Finally, the challenger sends ciphertext $\mathsf{ct} = (\{(\mathsf{rt}_i, d_i)\}_{i\in[\ell_n]}, \{\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}\}_{(i,j)\in S}, \{\tilde{y}_{i,1}\}_{i\in[\ell_n]})$ to $\mathcal{A}$. The adversary outputs a bit b'.

Hybrid $H_{b.\kappa.\tau}$ ($b \in \{0,1\}, (\kappa,\tau) \in S$): In this hybrid, for every $(i,j) \preceq (\kappa,\tau)$, the challenger computes $\mathsf{Enc\text{-}Step}i, j$ using a simulator. The differences from the previous hybrid are highlighted in red color.

5. The challenger samples $\mathsf{state}_{i,j} \leftarrow \{0,1\}^\lambda, r_{i,j} \leftarrow \{0,1\}^\lambda$ for each $(i,j) \in S'$ s.t. $(i,j) \succ (\kappa,\tau)$.

6. For each $(i,j) \in S$ s.t. $(i,j) \succ (\kappa,\tau)$, the challenger defines the circuit $\mathsf{Enc\text{-}Step}_{i,j}$ as in Fig. 2 with hash key hk, state $\mathsf{state}_{i,j+1}$, randomness $r_{i,j+1}$, identity id^*, message m_b hardwired in it. It then garbles the circuit $\mathsf{Enc\text{-}Step}_{i,j}$ i.e., $\widetilde{\mathsf{Enc\text{-}Step}}_{i,j} \leftarrow \mathsf{HG.GarbleCkt}(\mathsf{hk}, \mathsf{Enc\text{-}Step}_{i,j}, \mathsf{state}_{i,j})$.
For each $i \in [\ell_n]$ s.t $(i,1) \succ (\kappa,\tau)$, the challenger computes $\tilde{y}_{i,1} = \mathsf{HG.GarbleInp}(\mathsf{hk}, \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{rt}_i), \mathsf{state}_{i,1}; r_{i,1})$.
For each i s.t. $(i,d_i) \preceq (\kappa,\tau)$,
   - The challenger sets $\tilde{y}_{i,d_i+1} = 1 || \perp$ if $i < \alpha$.
   - The challenger samples ciphertext $\mathsf{ct}_i^* \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}^*, m_b)$ and sets $\tilde{y}_{i,d_i+1} = 1 || \mathsf{ct}_i^*$ if $i \geq \alpha$.

   For any i, Let us define $\mathsf{tag}_i = \mathsf{upr}$ if $i < \alpha \wedge \mathsf{path}_{i,\mathsf{upr}} \neq \epsilon$. $\mathsf{tag}_i = \mathsf{lwr}$ if $i < \alpha \wedge \mathsf{path}_{i,\mathsf{upr}} = \epsilon$, and $\mathsf{tag}_i = \mathsf{mid}$ if $i \geq \alpha$.
   If $\tau \neq d_\kappa$, the challenger computes $x = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{\kappa,\tau+1,\mathsf{tag}_\kappa})$ and sets $\tilde{y}_{\kappa,\tau+1} = \mathsf{HG.GarbleInp}(\mathsf{hk}, x, \mathsf{state}_{\kappa,\tau+1}; r_{\kappa,\tau+1})$.
   For each $(i,j) \in S$ s.t $(i,j) \preceq (\kappa,\tau)$, the challenger simulates the garbling of $\mathsf{Enc\text{-}Step}_{i,j}$ as $(\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}, \tilde{y}_{i,j}) \leftarrow \mathsf{HG.Sim}(\mathsf{hk}, \mathsf{node}_{i,j,\mathsf{tag}_i}, 1^L, \mathsf{flag}_{i,j+1} || \tilde{y}_{i,j+1})$[22], where L is the length of circuit $\mathsf{Enc\text{-}Step}_{i,j}$ defined in Fig. 2 and $\mathsf{flag}_{i,j+1} = 1$ iff $j = d_i$.

For the sake of simplicity, we hereby define Hybrid $H_{b.1.0}$ same as Hybrid H_b for any bit b. We now prove that Hybrid H_0 is computationally indistinguishable from Hybrid H_1 via a sequence of intermediate hybrids. For any PPT adversary $\mathcal{A}$ and any Hybrid H_s, let the probability that $\mathcal{A}$ outputs 1 in Hybrid H_s be $p_s^{\mathcal{A}}$.

For any $\kappa \in [\ell_n], \tau \in [d_\kappa]$, let $(\tilde{\kappa}, \tilde{\tau})$ be its preceding tuple i.e., $(\tilde{\kappa}, \tilde{\tau}) = \begin{cases} (\kappa, \tau-1) & \text{if } \tau > 1 \\ (\kappa-1, d_{\kappa-1}) & \text{if } \kappa > 1 \wedge \tau = 1 \\ (1,0) & \text{if } (\kappa,\tau) = (1,1) \end{cases}$.

Lemma 4.7. *Assuming* HG *is a secure hash garbling scheme, for every admissible*[20] *PPT adversary* $\mathcal{A}$*, there exists a negligible function* $negl(\cdot)$ *such that for any* $\lambda \in \mathbb{N}$*, bit* $b \in \{0,1\}$*, any* $(\kappa,\tau) \in S$*, we have* $|p_{b.\kappa.\tau}^{\mathcal{A}} - p_{b.\tilde{\kappa}.\tilde{\tau}}^{\mathcal{A}}| \leq negl(\lambda)$.

Proof. Suppose there exists an admissible PPT adversary $\mathcal{A}$ and indices $(\kappa,\tau) \in S$ s.t. $|p_{b.\kappa.\tau}^{\mathcal{A}} - p_{b.\tilde{\kappa}.\tilde{\tau}}^{\mathcal{A}}|$ is a non-negligible value ϵ. We construct a non-uniform reduction algorithm $\mathcal{B}$ which knows the indices (κ,τ) and breaks the security of hash garbling scheme.

The hash garbling game challenger $\mathcal{C}$ first sends a hash key hk to the reduction algorithm $\mathcal{B}$. $\mathcal{B}$ samples a PKE key pair $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$, sets $\mathsf{crs} = \mathsf{hk}$ and sends $(\mathsf{crs}, \mathsf{pk}^*)$ to the adversary $\mathcal{A}$. The adversary sends an identity id^*, public parameters pp, a proof π and a pair of messages m_0, m_1 to the challenger. If $\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*, \pi) = 0$, $\mathcal{B}$ aborts and outputs a random bit in hash garbling game. Let public parameters pp be $\{(\mathsf{rt}_i, d_i)\}_{i \in [\ell_n]}$ and proof π be $(\pi_1, \cdots, \pi_{\ell_n}, \alpha)$. For each $i < \alpha$, let $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{mid}})$. Similarly, for each $i \geq \alpha$, let $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{mid}}, \mathsf{path}_{i,\mathsf{upr}})$. For any $i \in [\ell_n]$ and $\mathsf{tag} \in \{\mathsf{lwr}, \mathsf{mid}, \mathsf{upr}\}$, let $\mathsf{path}_{i,\mathsf{tag}}$ be a sequence of values $(\mathsf{node}_{i,1,\mathsf{tag}}, \mathsf{node}_{i,2,\mathsf{tag}}, \cdots, \mathsf{node}_{i,d_i,\mathsf{tag}})$[21]. Let us define set $S = \{(i,j) : i \in [\ell_n], j \in [d_i]\}$ and set $S' = \{(i,j) : i \in [\ell_n], j \in [d_i+1]\}$. For any i, Let us define $\mathsf{tag}_i = \mathsf{upr}$ if $i < \alpha \wedge \mathsf{path}_{i,\mathsf{upr}} \neq \epsilon$. $\mathsf{tag}_i = \mathsf{lwr}$ if $i < \alpha \wedge \mathsf{path}_{i,\mathsf{upr}} = \epsilon$, and $\mathsf{tag}_i = \mathsf{mid}$ if $i \geq \alpha$. For each $(i,j) \in S'$ s.t. $(i,j) \succ (\kappa,\tau)$, $\mathcal{B}$ samples $\mathsf{state}_{i,j} \leftarrow \{0,1\}^\lambda, r_{i,j} \leftarrow \{0,1\}^\lambda$. For each $(i,j) \in S$ s.t. $(i,j) \succeq (\kappa,\tau)$, $\mathcal{B}$ creates the circuit $\mathsf{Enc\text{-}Step}_{i,j}$ as in Fig. 2 with message m_b and identity id^* hardwired in it. $\mathcal{B}$ then sends circuit $\mathsf{Enc\text{-}Step}_{\kappa,\tau}$ and input $\mathsf{node}_{\kappa,\tau,\mathsf{tag}_\kappa}$ to the hash garbling game challenger $\mathcal{C}$. $\mathcal{C}$ samples a bit $\gamma \leftarrow \{0,1\}$. If $\gamma = 0$, $\mathcal{C}$ samples $\mathsf{state} \leftarrow \{0,1\}^\lambda$ and computes $h = \mathsf{HG.Hash}(\mathsf{hk}, \mathsf{node}_{\kappa,\tau,\mathsf{tag}_\kappa})$, $\widetilde{\mathsf{Enc\text{-}Step}}_{\kappa,\tau} \leftarrow \mathsf{HG.GarbleCkt}(\mathsf{hk}, \mathsf{Enc\text{-}Step}_{\kappa,\tau}, \mathsf{state})$ and $\tilde{y}_{\kappa,\tau} \leftarrow \mathsf{HG.GarbleInp}(\mathsf{hk}, h, \mathsf{state})$. Otherwise $\mathcal{C}$ computes $x = \mathsf{Enc\text{-}Step}_{\kappa,\tau}(\mathsf{node}_{\kappa,\tau,\mathsf{tag}_\kappa})$ and runs simulator $(\widetilde{\mathsf{Enc\text{-}Step}}_{\kappa,\tau}, \tilde{y}_{\kappa,\tau}) \leftarrow \mathsf{HG.Sim}(\mathsf{hk}, \mathsf{node}_{\kappa,\tau,\mathsf{tag}_\kappa}, 1^{|\mathsf{Enc\text{-}Step}_{\kappa,\tau}|}, x)$. $\mathcal{C}$ sends $\widetilde{\mathsf{Enc\text{-}Step}}_{\kappa,\tau}, \tilde{y}_{\kappa,\tau}$ to $\mathcal{B}$. For each $i < \alpha$ s.t. $(i,d_i) \prec (\kappa,\tau)$, the challenger sets $\tilde{y}_{i,d_i+1} = 1 || \perp$. For each $i \geq \alpha$ s.t. $(i,d_i) \prec (\kappa,\tau)$, $\mathcal{B}$ samples a

[22] Note that simulating $\tilde{y}_{i,j}$ requires $\tilde{y}_{i,j+1}$. As a result, for each i, the challenger runs the simulator in the decreasing order of j.

ciphertext $\mathsf{ct}_i^* \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}^*, m_b)$ and sets $\tilde{y}_{i,j+1} = 1||\mathsf{ct}_i^*$. For each $(i,j) \in S$ s.t $(i,j) \prec (\kappa,\tau)$, $\mathcal{B}$ simulates the garbling of $\mathsf{Enc\text{-}Step}_{i,j}$ as $(\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}, \tilde{y}_{i,j}) \leftarrow \mathsf{HG.Sim}(\mathsf{hk}, \mathsf{node}_{i,j}, 1^L, \mathsf{flag}_{i,j+1}||\tilde{y}_{i,j+1})$[19], where L is the length of circuit $\mathsf{Enc\text{-}Step}_{i,j}$ defined in Fig. 2 and $\mathsf{flag}_{i,j+1} = 1$ iff $j = d_i$. Finally, $\mathcal{B}$ sends ciphertext $\mathsf{ct} = (\{(\mathsf{rt}_i, d_i)\}_{i\in[\ell_n]}, \{\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}\}_{(i,j)\in S}, \{\tilde{y}_{i,1}\}_{i\in[\ell_n]})$ to $\mathcal{A}$. The adversary outputs a bit b'. $\mathcal{B}$ outputs b' as its guess in hash garbling game.

We now analyze the advantage of $\mathcal{B}$ in hash garbling game. As the adversary $\mathcal{A}$ is admissible, $\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*, \pi) = 1$ and $\mathcal{B}$ does not abort. By Lemmas 4.3 and 4.4, when $\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \pi) = 1$, the path $\mathsf{path}_{\kappa,\mathsf{tag}_\kappa}$ resembles a sequence of nodes obtain by performing binary search on id^* in a tree with root rt_κ. Consequently, when the challenger runs step circuit $\mathsf{Enc\text{-}Step}_{\kappa,\tau}$ on input $\mathsf{node}_{\kappa,\tau,\mathsf{tag}_\kappa}$, the result is $0||\mathsf{node}_{\kappa,\tau+1,\mathsf{tag}_\kappa}$ if $\tau < d_\kappa$, and $1||\perp$ if $\tau = d_\kappa \wedge \kappa < \alpha$, and $1||\mathsf{Enc}(\mathsf{pk}^*, m_b)$ if $\tau = d_\kappa \wedge \kappa \geq \alpha$. As a resut, if $\gamma = 0$, then $\mathcal{B}$ emulates Hybrid $H_{b.\tilde{\kappa}.\tilde{\tau}}$ challenger to $\mathcal{A}$ and if $\gamma = 1$, then $\mathcal{B}$ emulates Hybrid $H_{b.\kappa.\tau}$ challenger to $\mathcal{A}$. By our assumption, $|p^{\mathcal{A}}_{\kappa.\tau} - p^{\mathcal{A}}_{\tilde{\kappa}.\tilde{\tau}}|$ is non-negligible. Therefore, the advantage of $\mathcal{B}$ in hash garbling game given by $|\Pr[b' = 1|\gamma = 0] - \Pr[b' = 1|\gamma = 1]|$ is non-negligible. ∎

Lemma 4.8. *Assuming* PKE *is a secure public key encryption scheme, for every admissible*[20] *PPT adversary* $\mathcal{A}$*, there exists a negligible function* $negl(\cdot)$ *such that for any* $\lambda \in \mathbb{N}$*, we have* $|p^{\mathcal{A}}_{0.\ell_n.d_{\ell_n}} - p^{\mathcal{A}}_{1.\ell_n.d_{\ell_n}}| \leq negl(\lambda)$.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $|p^{\mathcal{A}}_{0.\ell_n.d_{\ell_n}} - p^{\mathcal{A}}_{1.\ell_n.d_{\ell_n}}|$ is non-negligible value ϵ. We construct a reduction algorithm $\mathcal{B}$ that breaks semantic security of the PKE scheme.[23]

The challenger first sends a public key pk^* to the reduction algorithm $\mathcal{B}$. $\mathcal{B}$ samples a hash key $\mathsf{hk} \leftarrow \mathsf{HG.Setup}(1^\lambda, 1^{3\lambda+1})$, sets $\mathsf{crs} = \mathsf{hk}$ and sends $(\mathsf{crs}, \mathsf{pk}^*)$ to the adversary. The adversary then sends an identity id^*, public parameters pp, a proof π and a pair of messages m_0, m_1 to $\mathcal{B}$. If $\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*, \pi) = 0$, $\mathcal{B}$ aborts and outputs a random bit in PKE game. Let public parameters pp be $\{(\mathsf{rt}_i, d_i)\}_{i\in[\ell_n]}$ and proof π be $(\pi_1, \cdots, \pi_{\ell_n}, \alpha)$. For each $i < \alpha$, let $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{mid}})$. Similarly, for each $i \geq \alpha$, let $\pi_i = (\mathsf{path}_{i,\mathsf{lwr}}, \mathsf{path}_{i,\mathsf{mid}}, \mathsf{path}_{i,\mathsf{upr}})$. For any $i \in [\ell_n]$ and $\mathsf{tag} \in \{\mathsf{lwr}, \mathsf{mid}, \mathsf{upr}\}$, let $\mathsf{path}_{i,\mathsf{tag}}$ be a sequence of values $(\mathsf{node}_{i,1,\mathsf{tag}}, \mathsf{node}_{i,2,\mathsf{tag}}, \cdots, \mathsf{node}_{i,d_i,\mathsf{tag}})$[21]. $\mathcal{B}$ makes $\ell_n - \alpha + 1$ challenge encryption queries on (m_0, m_1) to the PKE challenger and obtains ciphertexts $\mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\alpha+1}, \cdots, \mathsf{ct}^*_{\ell_n}$. For each $i < \alpha$, $\mathcal{B}$ sets $\tilde{y}_{i,d_i+1} = 1||\perp$. For each $i \geq \alpha$, $\mathcal{B}$ sets $\tilde{y}_{i,d_i+1} = 1||\mathsf{ct}^*_i$. For each $(i,j) \in S$, $\mathcal{B}$ simulates the garbling of $\mathsf{Enc\text{-}Step}_{i,j}$ as $(\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}, \tilde{y}_{i,j}) \leftarrow \mathsf{HG.Sim}(\mathsf{hk}, \mathsf{node}_{i,j}, 1^L, \mathsf{flag}_{i,j+1}||\tilde{y}_{i,j+1})$[19], where L is the length of circuit $\mathsf{Enc\text{-}Step}_{i,j}$ defined in Fig. 2 and $\mathsf{flag}_{i,j+1} = 1$ iff $j = d_i$. Finally, $\mathcal{B}$ sends ciphertext $\mathsf{ct} = (\{(\mathsf{rt}_i, d_i)\}_{i\in[\ell_n]}, \{\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}\}_{i\in[\ell_n], j\in[d_i]}, \{\tilde{y}_{i,1}\}_{i\in[\ell_n]})$ to $\mathcal{A}$. The adversary outputs a bit b'. $\mathcal{B}$ outputs b' as its guess in PKE game.

We now analyze the advantage of $\mathcal{B}$ in breaking semantic security of PKE. We note that if PKE challenger encrypts message m_0, then $\mathcal{B}$ emulates Hybrid $H_{0.\ell_n.d_{\ell_n}}$ challenger to $\mathcal{A}$. Whereas if the PKE challenger encrypts message m_1, then $\mathcal{B}$ emulates Hybrid $H_{1.\ell_n.d_{\ell_n}}$ challenger to $\mathcal{A}$. As $\mathcal{A}$ can distinguish between the hybrids with non-negligible probability, $\mathcal{B}$ can break PKE semantic security with non-negligible probability. ∎

By the above sequence of lemmas and triangle inequality, Hybrid H_0 is computationally indistinguishable from Hybrid H_1. Therefore, our construction satisfies soundness for pre-registration verifiability property. ∎

4.4.3 Message Hiding Security

We now prove that any VRBE construction that satisfies completeness and soundeness requirements of Definitions 3.2, 3.5 and 3.6 also satisfies message hiding property.

[23] For the ease of exposition, we consider the semantic security game where the adversary is allowed to make polynomially many challenge queries of the form (m_0, m_1). The challenger samples a bit b and responds to each challenge query (m_0, m_1) with $\mathsf{Enc}(\mathsf{pk}, m_b)$. By a standard hybrid argument, this definition of semantic security is equivalent to the Definition 2.1.

Theorem 4.3. *Assuming a VRBE satisfies completeness property (Definition 3.2) soundness of pre-registration and post-registration properties (Definitions 3.5 and 3.6), then the scheme also satisfies message hiding property (Definition 3.4).*

Proof. Consider the message hiding game described in Definition 3.4. We first divide the adversary into 2 types.

- Type 1 Adversary: Restricted to make challenge encryption query $(\mathsf{chal}, \mathsf{id}, m_0, m_1)$ s.t. $\mathsf{id} = \mathsf{id}^*$, where id^* is the registered target identity.
- Type 2 Adversary: Restricted to make challenge encryption query $(\mathsf{chal}, \mathsf{id}, m_0, m_1)$ s.t. $\mathsf{id} \notin S_{\mathsf{ID}}$, where S_{ID} is the set of all registered identities.

For any adversary $\mathcal{A}$, let the advantage of $\mathcal{A}$ in message hiding game be $\mathsf{Adv}^{\mathcal{A}}$. We now prove Lemmas 4.9 and 4.10 which together imply Theorem 4.3.

Lemma 4.9. *Assuming any VRBE satisfies completeness property as per Definition 3.2 and soundness of post-registration property as per Definition 3.6, then for every Type 1 PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for every $\lambda \in \mathbb{N}$, $\mathsf{Adv}^{\mathcal{A}} \leq negl(\lambda)$*

Proof. Suppose there exists a VRBE scheme VRBE satisfying completeness property as per Definition 3.2, a type 1 PPT adversary $\mathcal{A}$ such that the advantage $\mathsf{Adv}^{\mathcal{A}}$ is non-negligible. We now construct a PPT reduction algorithm $\mathcal{B}$ which breaks soundness of post-registration property of VRBE scheme with non-negligible probability.

The soundness game challenger $\mathcal{C}$ sends a $(\mathsf{crs}, \mathsf{pk}^*)$ to the reduction algorithm $\mathcal{B}$, which initializes parameters $(\mathsf{pp}, \mathsf{aux}, u, S_{\mathsf{ID}}, \mathsf{id}^*) = (\epsilon, \epsilon, \epsilon, \emptyset, \perp)$ and forwards crs to the adversary $\mathcal{A}$. The adversary then adaptively makes registration queries of the form $(\mathsf{regnew}, \mathsf{id}, \mathsf{pk})$. For each such query, $\mathcal{B}$ aborts if $\mathsf{id} \in S_{\mathsf{ID}}$. Otherwise, $\mathcal{B}$ registers $(\mathsf{id}, \mathsf{pk})$ as $\mathsf{pp} \leftarrow \mathsf{Reg}^{\mathsf{aux}}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk})$ and sets $S_{\mathsf{ID}} = S_{\mathsf{ID}} \cup \{\mathsf{id}\}$. If the adversary makes a target registeration query $(\mathsf{regtgt}, \mathsf{id}^*)$, $\mathcal{B}$ aborts if $\mathsf{id}^* \in S_{\mathsf{ID}}$. Otherwise, $\mathcal{B}$ registers $(\mathsf{id}^*, \mathsf{pk}^*)$ as $\mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*)$, sets $S_{\mathsf{ID}} = S_{\mathsf{ID}} \cup \{\mathsf{id}^*\}$ and sends pk^* to $\mathcal{A}$. Finally, the adversary makes a challenge query $(\mathsf{chal}, \mathsf{id}, m_0, m_1)$. If $\mathsf{id} \neq \mathsf{id}^*$, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ computes proof $\pi^* \leftarrow \mathsf{PostProve}^{\mathsf{aux}}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*)$ and sends $(\mathsf{pp}, \mathsf{id}^*, \pi^*, m_0, m_1)$ to the challenger. If $\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*, \pi^*) = 0$, the challenger aborts. Otherwise, it samples a bit $b \leftarrow \{0, 1\}$ and sends $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, m_b)$ to $\mathcal{B}$, which forwards ct^* to $\mathcal{A}$. The adversary $\mathcal{A}$ outputs a bit b'. $\mathcal{B}$ outputs b' as its guess in soundness game.

We now analyze the advantage of $\mathcal{B}$ in the soundness game. As $\mathcal{A}$ is a valid type 1 adversary, it only makes challenge queries $(\mathsf{chal}, \mathsf{id}, m_0, m_1)$ s.t. $\mathsf{id} = \mathsf{id}^*$. As VRBE satisfies completeness property and $(\mathsf{id}^*, \mathsf{pk}^*)$ is registered by $\mathcal{B}$, we know that $\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*, \pi^*) = 1$ with overwhelming probability. Therefore, $\mathcal{B}$ does not abort and acts as a valid soundness game adversary and a valid message hiding game challenger with all but negligible probability. As $\mathcal{A}$ has a non-negligible advantage in winning the message hiding game, $\mathcal{B}$ has a non-negligible advantage in winning the soundness game. ∎

Lemma 4.10. *Assuming any VRBE satisfies completeness property as per Definition 3.2 and soundness of pre-registration property as per Definition 3.5, then for every Type 2 PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for every $\lambda \in \mathbb{N}$, $\mathsf{Adv}^{\mathcal{A}} \leq negl(\lambda)$*

Proof. This proof is similar to the proof Lemma 4.9. Suppose there exists a VRBE scheme VRBE satisfying completeness property as per Definition 3.2, a type 2 PPT adversary $\mathcal{A}$ such that the advantage $\mathsf{Adv}^{\mathcal{A}}$ is non-negligible. We now construct a PPT reduction algorithm $\mathcal{B}$ which breaks soundness of pre-registration property of VRBE scheme with non-negligible probability.

The soundness game challenger $\mathcal{C}$ sends a crs to the reduction algorithm $\mathcal{B}$, which initializes parameters $(\mathsf{pp}, \mathsf{aux}, u, S_{\mathsf{ID}}, \mathsf{id}^*) = (\epsilon, \epsilon, \epsilon, \emptyset, \perp)$ and forwards crs to the adversary $\mathcal{A}$. The adversary then adaptively makes registration queries of the form $(\mathsf{regnew}, \mathsf{id}, \mathsf{pk})$. For each such query, $\mathcal{B}$ aborts if $\mathsf{id} \in S_{\mathsf{ID}}$. Otherwise, $\mathcal{B}$ registers $(\mathsf{id}, \mathsf{pk})$ as $\mathsf{pp} \leftarrow \mathsf{Reg}^{\mathsf{aux}}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk})$ and sets $S_{\mathsf{ID}} = S_{\mathsf{ID}} \cup \{\mathsf{id}\}$. If the adversary makes a target registeration query $(\mathsf{regtgt}, \mathsf{id}^*)$, $\mathcal{B}$ aborts if $\mathsf{id}^* \in S_{\mathsf{ID}}$. Otherwise, $\mathcal{B}$ samples PKE pair $(\mathsf{pk}^*, \mathsf{sk}^*)$,

registers $(\mathsf{id}^*, \mathsf{pk}^*)$ as $\mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*)$, sets $S_{\mathsf{ID}} = S_{\mathsf{ID}} \cup \{\mathsf{id}^*\}$ and sends pk^* to $\mathcal{A}$. Finally, the adversary makes a challenge query $(\mathsf{chal}, \mathsf{id}, m_0, m_1)$. If $\mathsf{id} \in S_{\mathsf{ID}}$, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ computes proof $\pi \leftarrow \mathsf{PreProve}^{\mathsf{aux}}(\mathsf{crs}, \mathsf{pp}, \mathsf{id})$ and sends $(\mathsf{pp}, \mathsf{id}, \pi, m_0, m_1)$ to the challenger. If $\mathsf{PreVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \pi) = 0$, the challenger aborts. Otherwise, it samples a bit $b \leftarrow \{0, 1\}$ and sends $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, m_b)$ to $\mathcal{B}$, which forwards ct^* to $\mathcal{A}$. The adversary $\mathcal{A}$ outputs a bit b'. $\mathcal{B}$ outputs b' as its guess in soundness game.

We now analyze the advantage of $\mathcal{B}$ in the soundness game. As $\mathcal{A}$ is a valid type 2 adversary, it only makes challenge queries $(\mathsf{chal}, \mathsf{id}, m_0, m_1)$ s.t. $\mathsf{id} \notin S_{\mathsf{ID}}$. As VRBE satisfies completeness property and id is unregistered by $\mathcal{B}$, we know that $\mathsf{PreVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \pi) = 1$ with overwhelming probability. Therefore, $\mathcal{B}$ does not abort and acts as a valid soundness game adversary and a valid message hiding game challenger with all but negligible probability. As $\mathcal{A}$ has a non-negligible advantage in winning the message hiding game, $\mathcal{B}$ has a non-negligible advantage in winning the soundness game. ■

■

Acknowledgements

We thank anonymous reviewers for useful feedback, especially for pointing out a possible black-box approach for achieving verifiability.

References

[AKTZ17] Nikolaos Alexopoulos, Aggelos Kiayias, Riivo Talviste, and Thomas Zacharias. Mcmix: Anonymous messaging via secure multiparty computation. In *26th USENIX Security Symposium, USENIX Security 2017*, pages 1217–1234, 2017.

[ARP03] Sattam S Al-Riyami and Kenneth G Paterson. Certificateless public key cryptography. In *International conference on the theory and application of cryptology and information security*, pages 452–473. Springer, 2003.

[BC12] Nir Bitansky and Alessandro Chiesa. Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In *Advances in Cryptology - CRYPTO 2012*, pages 255–272, 2012.

[BCOP04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *EUROCRYPT '04*, volume 3027 of LNCS, pages 506–522, 2004.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil Pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, 2001.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology - CRYPTO 2001*, pages 1–18, 2001.

[BLSV17] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous ibe, leakage resilience and circular security from new assumptions. Cryptology ePrint Archive, Report 2017/967, 2017. http://eprint.iacr.org/2017/967.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BSJ+17] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. Ratcheted encryption and key exchange: The security of messaging. In *Advances in Cryptology - CRYPTO 2017*, pages 619–650, 2017.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: definitions and challenges. In *Proceedings of the 8th conference on Theory of cryptography*, TCC'11, pages 253–273, Berlin, Heidelberg, 2011. Springer-Verlag.

[CB95] David A. Cooper and Kenneth P. Birman. Preserving privacy in a network of mobile computers. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, pages 26–38, 1995.

[CBM15] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy, SP 2015*, pages 321–338, 2015.

[CCD+17] Katriel Cohn-Gordon, Cas J. F. Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017*, pages 451–466, 2017.

[CCV04] Zhaohui Cheng, Richard Comley, and Luminita Vasiu. Remove key escrow from the identity-based encryption system. In *Exploring New Frontiers of Theoretical Informatics*, pages 37–50. Springer, 2004.

[CDG+17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In *Annual International Cryptology Conference*, 2017.

[CF10] Henry Corrigan-Gibbs and Bryan Ford. Dissent: accountable anonymous group messaging. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010*, pages 340–350, 2010.

[Cho09] Sherman SM Chow. Removing escrow from identity-based encryption. In *International Workshop on Public Key Cryptography*, pages 256–276. Springer, 2009.

[CHSS02] Liqun Chen, Keith Harrison, David Soldera, and Nigel P Smart. Applications of multiple trust authorities in pairing based cryptosystems. In *International Conference on Infrastructure Security*, pages 260–275. Springer, 2002.

[Coc01] Clifford Cocks. An identity based encryption scheme based on Quadratic Residues. In *Cryptography and Coding, IMA International Conference*, volume 2260 of LNCS, pages 360–363, 2001.

[DG17a] Nico Döttling and Sanjam Garg. Identity-based encryption from the diffie-hellman assumption. In *Advances in Cryptology - CRYPTO 2017*, pages 537–569, 2017.

[DG17b] Nico Döttling and Sanjam Garg. From selective ibe to full ibe and selective hibe. TCC, 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.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography, 1976.

[Gen06] Craig Gentry. Practical identity-based encryption without random oracles. In *EUROCRYPT '06*, volume 4004 of LNCS, pages 445–464, 2006.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[GHM+19] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In *Public-Key Cryptography - PKC 2019*, pages 63–93, 2019.

[GHMR18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. In *TCC 2018*, pages 689–718, 2018.

[GLSW08] Vipul Goyal, Steve Lu, Amit Sahai, and Brent Waters. Black-box accountable authority identity-based encryption. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 427–436. ACM, 2008.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[Goy07] Vipul Goyal. Reducing trust in the PKG in identity based cryptosystems. In *CRYPTO '07*, volume 4622 of LNCS, pages 430–447, 2007.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, 2006.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, 2010.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011*, pages 99–108, 2011.

[KG10] Aniket Kate and Ian Goldberg. Distributed private-key generators for identity-based cryptography. In *International Conference on Security and Cryptography for Networks*, pages 436–453. Springer, 2010.

[LBD+04] Byoungcheon Lee, Colin Boyd, Ed Dawson, Kwangjo Kim, Jeongmo Yang, and Seungjae Yoo. Secure key issuing in id-based cryptography. In *ACSW Frontiers 2004, 2004 ACSW Workshops*, pages 69–74, 2004.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012*, pages 169–189, 2012.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th Annual Symposium on Foundations of Computer Science*, pages 436–453, 1994.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In *Advances in Cryptology - CRYPTO 2003*, pages 96–109, 2003.

[PS08] Kenneth G Paterson and Sriramkrishnan Srinivasan. Security and anonymity of identity-based encryption with multiple trusted authorities. In *International Conference on Pairing-Based Cryptography*, pages 354–375. Springer, 2008.

[RMS18] Paul Rösler, Christian Mainka, and Jörg Schwenk. More is less: On the end-to-end security of group chats in signal, whatsapp, and threema. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018*, pages 415–429, 2018.

[Rog15] Phillip Rogaway. The moral character of cryptographic work. Cryptology ePrint Archive, Report 2015/1162, 2015. https://eprint.iacr.org/2015/1162.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[SW08] Amit Sahai and Brent Waters. Slides on functional encryption. PowerPoint presentation, 2008. http://www.cs.utexas.edu/~bwaters/presentations/files/functional.ppt.

[UDB+15] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. Sok: Secure messaging. In *2015 IEEE Symposium on Security and Privacy, SP 2015*, pages 232–249, 2015.

[WQT18] Quanyun Wei, Fang Qi, and Zhe Tang. Remove key escrow from the BF and gentry identity-based encryption with non-interactive key generation. *Telecommunication Systems*, 69(2):253–262, 2018.

A Universal Registration Based Encryption

In Registration Based Encryption, various identities locally sample (public key, secret key) pair of an apriori fixed public key encryption scheme, and a key accumulator accumulates identity-public key pairs of various users and publishes a short commitment. This commitment can be later used to encrypt a message to any registered identity. In this section, we extend the definition of Registration Based Encryption to more general access structures. Concretely, we define the notion of Universal Registration Based Encryption where the users have the flexibility to choose a functional encryption scheme they would like to use from an apriori fixed class of schemes. Any user can choose a functional encryption scheme and locally sample (master public key, master secret key) pair for the scheme. The users then encode the public key in a universal format, which includes both the public key and name of the scheme selected by the user. We call a public key encoded in such a format as a universal public key. The key accumulator then registers (identity, universal public key) pairs of all the users and publishes a short commitment as earlier. For the ease of exposition, we define the notion of universal RBE for FE schemes. However, one could also support regular PKE, IBE, HIBE and ABE systems in a similar way. To simplify our notations, we define a universal FE system as follows.

- $\mathsf{UnivSetup}(1^\lambda, \mathsf{Sch}) \to (\mathsf{upk}, \mathsf{usk})$: The universal setup algorithm takes a security parameter λ and name of an FE scheme Sch as input. It samples master public key and master secret key $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Sch.Setup}(1^\lambda)$ of the scheme. It then outputs a universal master public key $\mathsf{upk} = (\mathsf{Sch}, \mathsf{mpk})$ and a universal master secret key $\mathsf{usk} = (\mathsf{Sch}, \mathsf{msk})$.

- $\mathsf{UnivKeygen}(\mathsf{usk}, f) \to \mathsf{usk}_f$: The universal keygen algorithm takes a universal secret key $\mathsf{usk} = (\mathsf{Sch}, \mathsf{msk})$ and a description of a function f as input. It computes secret key $\mathsf{sk}_f \leftarrow \mathsf{Sch.Keygen}(\mathsf{msk}, f)$ and outputs a universal function-specific secret key $\mathsf{usk}_f = (\mathsf{Sch}, \mathsf{sk}_f)$.

- $\mathsf{UnivEnc}(\mathsf{upk}, m) \to \mathsf{ct}$: The universal encryption algorithm takes as input a universal public key $\mathsf{upk} = (\mathsf{Sch}, \mathsf{mpk})$ and a message m. It computes and outputs a ciphertext $\mathsf{ct} \leftarrow \mathsf{Sch.Enc}(\mathsf{mpk}, m)$.

- $\mathsf{UnivDec}(\mathsf{usk}_f, \mathsf{ct}) \to y$: The universal decryption algorithm takes as input a universal secret key $\mathsf{usk}_f = (\mathsf{Sch}, \mathsf{sk}_f)$ and a ciphertext ct. It then decrypts the ciphertext as $y = \mathsf{Sch.Dec}(\mathsf{sk}_f, \mathsf{ct})$ and outputs y.

We now formally define the notion of universal RBE for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_\lambda$, identity space $\mathcal{ID} = \{\mathcal{ID}_\lambda\}_\lambda$, and a class of FE schemes $\mathcal{FE} = \{\mathcal{FE}_\lambda\}_\lambda$. The system consists of $(\mathsf{CRSGen}, \mathsf{Gen}, \mathsf{Reg}, \mathsf{Enc},$

$\mathsf{Upd}, \mathsf{Dec}, \mathsf{PreProve}, \mathsf{PreVerify}, \mathsf{PostProve}, \mathsf{PostVerify}$) algorithms. The syntax of all the algorithms remains same as VRBE except for few changes. The Gen algorithm now additionally takes name of an FE scheme as input and behaves similar to the universal setup algorithm described above. The $\mathsf{Reg}, \mathsf{PostProve}$ and $\mathsf{PostVerify}$ algorithms now deal with universal public keys, instead of PKE public keys. The Dec algorithm uses a universal function-specific secret key as input. We highlight the changes from the definition of VRBE in red.

$\mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{upk}) \to \mathsf{pp}'$. The registration algorithm is a deterministic algorithm, that takes as input the common reference string crs, current public parameter pp, an identity id to be registered, and a corresponding universal public key upk. It maintains auxiliary information aux, and outputs the updated parameters pp'. The registration algorithm is modelled as a RAM program where it can read/write to arbitrary locations of the auxiliary information aux. (The system is initialized with pp and aux set to ϵ.)

$\mathsf{Dec}(\mathsf{usk}_f, u, \mathsf{ct}) \to m/\mathsf{GetUpd}/\perp$. The decryption algorithm takes as input a universal function-specific secret key usk_f, a key update u, *auxilliary information* f and a ciphertext ct, and it outputs either a message $m \in \mathcal{M}$, or a special symbol in $\{\perp, \mathsf{GetUpd}\}$. (Here GetUpd indicates that a key update might be needed for decryption.)

$\mathsf{PostProve}^{\mathsf{aux}}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{upk}) \to \pi$. The post-registration prover algorithm is a deterministic algorithm, that takes as input the common reference string crs, public parameters pp, an identity id, and a universal public key upk. Given the auxiliary information aux, it outputs a post-registration proof π. Similar to the registration algorithm, this is also modelled as a RAM program, but it is only given read access to arbitrary locations of the auxiliary information aux.

$\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{upk}, \pi) \to 0/1$. The post-registration verifier algorithm takes as input the common reference string crs, public parameter pp, an identity id, a universal public key upk, and a proof π. It outputs a single bit $0/1$ denoting whether the proof is accepted or not.

We need a unviersal RBE to satisfy all the completeness, efficiency, completeness of pre/post-registration, efficiently extractable completeness of post-registration, soundness of pre/post-registration verifiability and message hiding requirements as presented in Section 3 with few changes.

Definition A.1 (Completeness, compactness, and efficiency of Universal RBE)**.** *For any (stateful) interactive computationally unbounded adversary $\mathcal{A}$ that has a $\mathsf{poly}(\lambda)$ round complexity, consider the following game $\mathsf{Comp}^{\mathsf{RBE}}_{\mathcal{A}}(\lambda)$.*

1. *(Initialization) The challenger initializes parameters as $(\mathsf{pp}, \mathsf{aux}, u, S_{\mathsf{ID}}, \mathsf{id}^*, t) = (\epsilon, \epsilon, \epsilon, \emptyset, \perp, 0)$, samples $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$, and sends the crs to $\mathcal{A}$.*

2. *(Query Phase) $\mathcal{A}$ makes polynomially many queries of the following form, where each query is considered as a single round of interaction between the challenger and the adversary.*

   (a) ***Registering new (non-target) identity.*** *On a query of the form $(\mathsf{regnew}, \mathsf{id}, \mathsf{upk})$, the challenger checks that $\mathsf{id} \notin S_{\mathsf{ID}}$, and registers $(\mathsf{id}, \mathsf{upk})$ by running the registration procedure as $\mathsf{pp} := \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{upk})$. It adds id to the set as $S_{\mathsf{ID}} := S_{\mathsf{ID}} \cup \{\mathsf{id}\}$. (Note that the challenger updates the parameters $\mathsf{pp}, \mathsf{aux}, S_{\mathsf{ID}}$ after each query. Also, it aborts if the check fails.)*

   (b) ***Registering target identity.*** *On a query of the form $(\mathsf{regtgt}, \mathsf{id}, \mathsf{Sch})$, the challenger first checks that $\mathsf{id}^* = \perp$. If the check fails, it aborts. Else, it sets $\mathsf{id}^* := \mathsf{id}$, samples challenge key pair $(\mathsf{upk}^*, \mathsf{usk}^*) \leftarrow \mathsf{UnivSetup}(1^\lambda, \mathsf{Sch})$, updates public parameters as $\mathsf{pp} := \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{upk}^*)$, and sets $S_{\mathsf{ID}} := S_{\mathsf{ID}} \cup \{\mathsf{id}^*\}$. Finally, it sends the challenge universal public key upk^* to $\mathcal{A}$. (Here the challenger stores the universal secret key usk^* in addition to updating all other parameters. Also, note that the adversary here is restricted to make such a query at most once, since the challenger would abort otherwise.)*

(c) ***Keygen Query.*** *On a query of the form* $(\mathsf{keygentgt}, f)$*, the challenger first checks and aborts if* $\mathsf{id}^* = \perp$*. Otherwise, the challenger samples* $\mathsf{usk}_f^* \leftarrow \mathsf{UnivKeygen}(\mathsf{usk}^*, f)$ *and sends it to the adversary.*

(d) ***Target identity encryptions.*** *On a query of the form* (enctgt, m)*, the challenger checks if* $\mathsf{id}^* \neq \perp$*. It aborts if the check fails. Otherwise, it sets* $t := t + 1$*,* $\widetilde{m}_t := m$*, and computes ciphertext* $\mathsf{ct}_t \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \widetilde{m}_t)$*. It stores the tuple* $(t, \widetilde{m}_t, \mathsf{ct}_t)$*, and sends the ciphertext* ct_t *to* $\mathcal{A}$*.*[24]

(e) ***Target identity decryptions.*** *On a query of the form* (dectgt, f, j)*, the challenger checks if* $\mathsf{id}^* \neq \perp$ *and* $j \in [t]$*. It aborts if the check fails. Otherwise, it samples* $\mathsf{usk}_f^* \leftarrow \mathsf{UnivKeygen}(\mathsf{usk}, f)$*, computes* $y_j = \mathsf{Dec}(\mathsf{usk}_f^*, u, \mathsf{ct}_j)$*. If* $y_j = \mathsf{GetUpd}$*, then it generates the fresh update* $u := \mathsf{Upd}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{id}^*)$ *and then re-computes* $y_j = \mathsf{Dec}(\mathsf{usk}_f^*, u, \mathsf{ct}_j)$*. Finally, the challenger stores the tuple* (j, f, y_j)*.*

3. *(Output Phase) We say that the adversary* $\mathcal{A}$ *wins the game if there is some tuple of the form* (j, f, y_j)*, for which* $f(\widetilde{m}_j) \neq y_j$*.*

Let $n = |S_{\mathsf{ID}}|$ *denote the number of identities registered until any specific round in the above game. We say that an RBE scheme is complete, compact, and efficient if for every (stateful) interactive computationally unbounded adversary* $\mathcal{A}$ *that has a* $\mathsf{poly}(\lambda)$ *round complexity, there exists polynomials* p_1, p_2, p_3, p_4, p_5 *and a negligible function* $negl(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*, the following holds:*

Completeness. $\Pr[\mathcal{A}$ *wins in* $\mathsf{Comp}_{\mathcal{A}}^{\mathsf{RBE}}(\lambda)] \leq negl(\lambda)$*.*

Compactness of public parameters and updates. $|\mathsf{pp}| \leq p_1(\lambda, \log n)$ *and* $|u| \leq p_2(\lambda, \log n)$*.*

Efficiency of registration and update. *The running time of each invocation of* Reg *and* Upd *algorithms is at most* $p_3(\lambda, \log n)$ *and* $p_4(\lambda, \log n)$*, respectively. (Note that this implies the above compactness property.)*

Efficiency of the number of updates. *The total number of invocations of* Upd *for identity* id^* *during* target identity decryption phase *(i.e., Step* 2d *of game* $\mathsf{Comp}_{\mathcal{A}}^{\mathsf{RBE}}(\lambda)$*) is at most* $p_5(\lambda, \log n)$ *for every* n*.*

Definition A.2 (Completeness and Efficiency of Pre/Post-Registration)**.** *The Completeness and Efficiency of pre/post-Registration property of universal RBE is same as that of regular RBE presented in Definition 3.2.*

Definition A.3 (Efficiently Extractable Completeness of Post-Registration)**.** *A universal RBE scheme satisfies efficiently extractable completeness property for post-registration if there exists a deterministic polynomial-time algorithm* UpdExt *such that for any computationally unbounded admissible adversary* $\mathcal{A}$*, there exists a negligible function* $negl(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*, we have*

$$\Pr\left[\mathsf{Dec}(\mathsf{usk}_f, u, \mathsf{ct}) \neq f(m) \;:\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda), \mathsf{Sch} \leftarrow \mathcal{A}(\mathsf{crs}), (\mathsf{upk}, \mathsf{usk}) \leftarrow \mathsf{UnivSetup}(1^\lambda, \mathsf{Sch}) \\ (\mathsf{pp}, \mathsf{id}, \pi, f, m) \leftarrow \mathcal{A}^{\mathsf{UnivKeygen}(\mathsf{usk}, \cdot)}(\mathsf{crs}, \mathsf{upk}), u \leftarrow \mathsf{UpdExt}(\pi, \mathsf{id}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, m), \mathsf{usk}_f \leftarrow \mathsf{UnivKeygen}(\mathsf{usk}, f) \end{array}\right] \leq negl(\lambda)$$

where an adversary $\mathcal{A}$ *is said to be admissible if it always produces a valid post-registration proof* π *i.e.,* $\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{upk}, \pi) = 1$*.*

Definition A.4 (Soundness of Pre-Registration Verifiability)**.** *A universal RBE scheme satisfies soundness of pre-registration verifiability if for every stateful* admissible *PPT adversary* $\mathcal{A}$*, there exists a negligible function* $negl(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*, the following holds*

$$\Pr\left[\mathcal{A}(\mathsf{ct}) = b \;:\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda) \\ (\mathsf{pp}, \mathsf{id}, \pi, m_0, m_1) \leftarrow \mathcal{A}(\mathsf{crs}) \\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, m_b) \end{array}\right] \leq \frac{1}{2} + negl(\lambda),$$

[24] Here and throughout, whenever we write the challenger stores the tuple, we mean that it appends this to its local state such that these could be obtained by the challenger when referred to later in the game.

where $\mathcal{A}$ is admissible if and only if π is a valid pre-registration proof, i.e. $\mathsf{PreVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \pi) = 1$.

Definition A.5 (Soundness of Post-Registration Verifiability)**.** *A universal RBE scheme satisfies soundness of post-registration verifiability if for every stateful* admissible *PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds*

$$\Pr\left[\mathcal{A}^{\mathsf{UnivKeygen}(\mathsf{usk},\cdot)}(\mathsf{ct}) = b \ : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda);\ \mathsf{Sch} \leftarrow \mathcal{A}(\mathsf{crs});\ (\mathsf{upk}, \mathsf{usk}) \leftarrow \mathsf{UnivSetup}(1^\lambda, \mathsf{Sch}) \\ (\mathsf{pp}, \mathsf{id}, \pi, m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{UnivKeygen}(\mathsf{usk},\cdot)}(\mathsf{crs}, \mathsf{upk}) \\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, m_b) \end{array}\right] \leq \frac{1}{2} + negl(\lambda),$$

where $\mathcal{A}$ is admissible if and only if π is a valid post-registration proof, i.e. $\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{upk}, \pi) = 1$*, and for each* $\mathsf{UnivKeygen}(\mathsf{usk}, f)$ *query made by the $\mathcal{A}$, it should satisfy $f(m_0) = f(m_1)$.*

Definition A.6 (Message Hiding Security)**.** *For any (stateful) interactive PPT adversary $\mathcal{A}$, consider the following game* $\mathsf{Sec}^{\mathsf{RBE}}_{\mathcal{A}}(\lambda)$.

1. *(Initialization) The challenger initializes parameters as* $(\mathsf{pp}, \mathsf{aux}, u, S_{\mathsf{ID}}, \mathsf{id}^*, \mathcal{F}) = (\epsilon, \epsilon, \epsilon, \emptyset, \perp, \emptyset)$*, samples* $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$*, and sends the* crs *to $\mathcal{A}$.*
2. *(Query Phase) $\mathcal{A}$ makes polynomially many queries of the following form:*
   (a) ***Registering new (non-target) identity.*** *On a query of the form* $(\mathsf{regnew}, \mathsf{id}, \mathsf{upk})$*, the challenger checks that* $\mathsf{id} \notin S_{\mathsf{ID}}$*, and registers* $(\mathsf{id}, \mathsf{upk})$ *by running the registration procedure as* $\mathsf{pp} := \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{upk})$*. It adds* id *to the set as* $S_{\mathsf{ID}} := S_{\mathsf{ID}} \cup \{\mathsf{id}\}$.
   (b) ***Registering target identity.*** *On a query of the form* $(\mathsf{regtgt}, \mathsf{id}, \mathsf{Sch})$*, the challenger first checks that* $\mathsf{id}^* = \perp$*. If the check fails, it aborts. Else, it sets* $\mathsf{id}^* := \mathsf{id}$*, samples challenge universal key pair* $(\mathsf{upk}^*, \mathsf{usk}^*) \leftarrow \mathsf{UnivSetup}(1^\lambda, \mathsf{Sch})$*, updates public parameters as* $\mathsf{pp} := \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{upk}^*)$*, and sets* $S_{\mathsf{ID}} := S_{\mathsf{ID}} \cup \{\mathsf{id}^*\}$*. Finally, it sends the challenge universal public key* upk^* *to $\mathcal{A}$.*
   (c) ***Keygen Query.*** *On a query of the form* $(\mathsf{keygentgt}, f)$*, the challenger first checks and aborts if* $\mathsf{id}^* = \perp$*. Otherwise, the challenger sets* $\mathcal{F} : \mathcal{F} \cup \{f\}$*, samples* $\mathsf{usk}^*_f \leftarrow \mathsf{UnivKeygen}(\mathsf{usk}^*, f)$ *and sends it to the adversary.*
3. *(Challenge Phase) On a query of the form* $(\mathsf{chal}, \mathsf{id}, m_0, m_1)$*, then the challenger aborts if* $\mathsf{id} \in S_{\mathsf{ID}} \backslash \{\mathsf{id}^*\}$*. The challenger also aborts if $f(m_0) \neq f(m_1)$ for any function $f \in \mathcal{F}$. Otherwise, the challenger samples a bit* $b \leftarrow \{0,1\}$ *and computes challenge ciphertext* $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, m_b)$.
4. *(Output Phase) The adversary $\mathcal{A}$ outputs a bit b' and wins the game if $b' = b$.*

We say that a universal RBE scheme is message-hiding secure if for every (stateful) interactive PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for every $\lambda \in \mathbb{N}$, $\Pr[\mathcal{A}$ *wins in* $\mathsf{Sec}^{\mathsf{RBE}}_{\mathcal{A}}(\lambda)] \leq \frac{1}{2} + negl(\lambda)$.