Verifiable Registration-Based Encryption

Rishab Goyal[1] and Satyanarayana Vusirikala[2]

[1] MIT
goyal@utexas.edu
[2] UT Austin
satya@cs.utexas.edu

Abstract. In recent work, Garg, Hajiabadi, Mahmoody, and Rahimi [18] 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 [33]. 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 [18] based on standard assumptions only provided weak efficiency properties. In a follow-up work by Garg, Hajiabadi, Mahmoody, Rahimi, and Sekar [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 that 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 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 prior work of Garg et al. [19]. And lastly, we initiate a study on the extension of VRBE to a wider range of access and trust structures.

1 Introduction

Public-key encryption (PKE) [13,30,21] 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) [33,12,5], Attribute-Based Encryption (ABE) [32,23], and most notably Functional Encryption (FE) [6] which is meant to encapsulate both IBE and ABE functionalities.

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 ([5,8,1,9,22,29,11,25] 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 [18], 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, [18] 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 [18,19], efficient construction of RBE systems were provided from a wide variety of assumptions (such as CDH, Factoring, LWE, iO). Specifically, [18] gave an efficient construction from indistinguishability obfuscation (iO) [2,17], and a weakly efficient construction from hash garbling scheme [18]. In a follow-up work by Garg et al. [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 [31] which was one of the prompts behind the initial work on RBE in [18]:

> *"[...] 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 [18,19], 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 [18] 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. [19], wherein the size of ciphertexts in our construction is significantly smaller.[3] 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 [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[4], 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

[3] Looking ahead, our efficiency gain is due to the fact that our construction takes a one-shot (single-step) approach whereas [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.

[4] 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.

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 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 [18,19]. Before outlining our approach, we quickly recall the high-level structure used in prior works [18,19] since our construction uses similar building blocks. Let us first look at the weakly efficient RBE construction provided in [18] 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 [18] construction uses the ideas developed in a long line of works [10,14,16,7,15] 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 [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 [18]. 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 [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 generic methods for making existing RBE schemes verifiable.

Although the basic principles behind our simplified construction and the one provided in [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 [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 an 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 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 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 (id, pk) as a leaf in the 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 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 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 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.[5] 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 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 EncTree corresponding to $\mathsf{snapshot}_{j_1}$ for decryption/obtaining update information;

[5] The snapshotting trick was implicitly used in [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/generic 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.[6] 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.

[6] 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.

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.[7] 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 generic 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 [27,24,20,26,4] 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 generic approach seems possible, but would require maintaining additional data structures for consistency checks. More importantly, this approach necessitates making additional assumptions as for most succinct non-interactive proof systems we either need to make certain non-falsifiable assumptions [28,20], or work in the Random Oracle model [27,3]. Our construction and the proofs of verifiability do not rely on any extra assumptions other than what is already required in existing RBE systems [18,19] themselves, thus our results show that verifiability comes for free.

[7] 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.

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-generic 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.

Related Work and Future Directions. Due to space constraints, we postpone this to full version of the paper.

2 Hash Garbling

We now review the notion of hash garbling scheme introduced in [18].

$\mathsf{Setup}(1^\lambda, 1^\ell) \to \mathsf{hk}$. 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.

Definition 1 (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 $\mathsf{Eval}(\tilde{C}, \tilde{y}, x) = C(x)$.*

Definition 2 (Security). *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 [18]. 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.

$\mathsf{PostProve}^{\mathsf{aux}}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk}) \rightarrow \pi$. 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.

$\mathsf{PostVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk}, \pi) \rightarrow 0/1$. 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.

3.1 Correctness

The definition of completeness, compactness, and efficiency for RBE system is studied in [18,19]. For completeness, we review the definition in full version of the paper. 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. Due to space constraints, we postpone the formal definitions to full version of the paper.

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 (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* $\mathsf{id} \notin S_{\mathsf{ID}} \setminus \{\mathsf{id}^*\}$*. 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* $\lambda \in \mathbb{N}$*,* $\Pr[\mathcal{A} \text{ wins in } \mathsf{Sec}^{\mathsf{RBE}}_{\mathcal{A}}(\lambda)] \leq \frac{1}{2} + negl(\lambda)$*.*

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 4 (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*

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

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 5 (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*

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

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 PKE = (PKE.Setup, PKE.Enc, PKE.Dec), and a hash garbling scheme HG = (HG.Setup, HG.Hash, HG.GarbleCkt, HG.GarbleInp, 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.[8]
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

[8] Note that the tree re-balancing operation has to be carefully performed as in our abstraction (as well as the [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.

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 $b_{i,j}$.[9] (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.[10] And, the public parameters pp consists 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.

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

[10] 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.

$\mathsf{CRSGen}(1^\lambda) \to \mathsf{crs}$. 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 $\mathsf{aux} = \Big(\mathsf{IDTree}, \{(\mathsf{EncTree}_i, n_i)\}_{i\in[\ell_n]}\Big)$, 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 modifies the auxilliary information as $\mathsf{aux} = \Big(\mathsf{IDTree}, \left\{(\mathsf{EncTree}'_i, n'_i)\right\}_{i\in[\ell_n+1-\delta]}\Big)$, 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:

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 $\mathsf{aux} = \{\mathsf{IDTree}, (\mathsf{EncTree}'_1, 8) : [\mathsf{id}_1, \ldots, \mathsf{id}_8]\}$.

Fig. 1. An example demonstrating aux being updated during registration

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 Figure 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 $\widetilde{y}_{i,1} = \mathsf{HG.GarbleInp}(\mathsf{hk}, h_i, \mathsf{state}_{i,1}; r_{i,1})$.
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

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 $0||\mathsf{HG.GarbleInp}(\mathsf{hk}, a, \mathsf{state}_{i,j+1}; r_{i,j+1})$.

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

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: $\mathsf{ct} = \left(\{(\mathsf{rt}_i, d_i)\}_i, \left\{\widetilde{\mathsf{Enc\text{-}Step}}_{i,j}\right\}_{i,j}, \{\widetilde{y}_{i,1}\}_i\right)$ 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 $\perp$.

$\mathsf{PreProve}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{id}) \to \pi$. Let $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_{i\in[\ell_n]}$ and $\mathsf{aux} = \Big(\mathsf{IDTree}, \{(\mathsf{EncTree}_i, n_i)\}_{i\in[\ell_n]}\Big)$. 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[11] 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,

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

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 $\pi = (\pi_1, \ldots, \pi_{\ell_n})$.

$\mathsf{PreVerify}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \pi) \to 0/1$. Let $\mathsf{crs} = \mathsf{hk}$, $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_{i \in [\ell_n]}$, $\pi = (\pi_i)_{i \in [\ell_n]}$.[12] 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 Figure 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.

$\mathsf{PostProve}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{id}, \mathsf{pk}) \to \pi$. Let $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_{i \in [\ell_n]}$ and $\mathsf{aux} = \Big(\mathsf{IDTree}, \{(\mathsf{EncTree}_i, n_i)\}_{i \in [\ell_n]}\Big)$. 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 $\mathsf{EncTree}_i$.[13] (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.

— 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,

[12] 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.

[13] 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. At the end, we explain how to handle 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.

Fig. 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 $\mathsf{EncTree}_i$

$\mathsf{id}_{i,\mathsf{upr}}$ (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 $\mathsf{crs} = \mathsf{hk}$, $\mathsf{pp} = \{(\mathsf{rt}_i, d_i)\}_{i \in [\ell_n]}$, $\pi = (\pi_1, \ldots, \pi_{\ell_n}, \ell)$.[14] 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 Figure 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 Figure 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 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 and Completeness

The above VRBE construction is efficient in the sense that if n is the number of registered users, then (1) The time complexity of Reg algorithm is $O(\log^2 n)$, (2) The size of the public parameters is $O(\log n)$, (3) The time complexity of Upd algorithm is $O(\log n)$, (4) The size of an update is $O(\log n)$, (5) The number of updates to any user is $O(\log n)$, (6) The time complexity of $\mathsf{PreProve}$, $\mathsf{PostProve}$ algorithms is $O(\log^2 n)$, and (7) The size of pre/post-registration proofs is $O(\log^2 n)$. Due to space constraints, we provide the full efficiency analysis of the above construction in full version of the paper.

The above scheme satisfies the correctness property as the decryptor internally performs a binary search on id in the $\mathsf{EncTree}$ and always obtains a

[14] 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.

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. At the end, we explain how to handle 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.

The case when $\mathsf{path}_{i,\mathsf{upr}} = \epsilon$ is handled analogously.

Fig. 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 $\mathsf{EncTree}_i$

PKE encryption of the message m using his public key $\mathsf{pk}_{\mathsf{id}}$. The above scheme satisfies the completeness of pre/post-registration as any proof obtained by $\mathsf{PreProve}/\mathsf{PostProve}$ algorithms satisfy the conditions in Figures 3 and 4. Due to space constraints, we postpone full proofs to the full version of the paper.

4.3 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 to 5. 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 $\mathsf{EncTree}$ 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 $\mathsf{EncTree}$ 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.

Due to space constraints, we postpone the full proofs to full version of the paper.

Acknowledgements. We thank anonymous reviewers for useful feedback, especially for pointing out a possible black-box approach for achieving verifiability. The work is done in part while the first author was at UT Austin (supported by IBM PhD Fellowship), and at the Simons Institute for the Theory of Computing (supported by Simons-Berkeley research fellowship). The second author is supported by Packard Fellowship, NSF CNS-1908611, CNS-1414082, DARPA SafeWare and Packard Foundation Fellowship.

References

1. Al-Riyami, S.S., Paterson, K.G.: Certificateless public key cryptography. In: International conference on the theory and application of cryptology and information security (2003)
2. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. In: CRYPTO (2001)
3. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: CCS (1993)
4. Bitansky, N., Chiesa, A.: Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In: CRYPTO (2012)
5. Boneh, D., Franklin, M.K.: Identity-based encryption from the Weil Pairing. In: CRYPTO (2001)
6. Boneh, D., Sahai, A., Waters, B.: Functional encryption: definitions and challenges. In: TCC (2011)
7. Brakerski, Z., Lombardi, A., Segev, G., Vaikuntanathan, V.: Anonymous ibe, leakage resilience and circular security from new assumptions. In: EUROCRYPT (2018)
8. Chen, L., Harrison, K., Soldera, D., Smart, N.P.: Applications of multiple trust authorities in pairing based cryptosystems. In: International Conference on Infrastructure Security (2002)
9. Cheng, Z., Comley, R., Vasiu, L.: Remove key escrow from the identity-based encryption system. In: Exploring New Frontiers of Theoretical Informatics (2004)
10. Cho, C., Döttling, N., Garg, S., Gupta, D., Miao, P., Polychroniadou, A.: Laconic oblivious transfer and its applications. In: CRYPTO 2017 (2017)
11. Chow, S.S.: Removing escrow from identity-based encryption. In: PKC (2009)
12. Cocks, C.: An identity based encryption scheme based on Quadratic Residues. In: Cryptography and Coding, IMA International Conference (2001)

13. Diffie, W., Hellman, M.E.: New directions in cryptography (1976)
14. Döttling, N., Garg, S.: Identity-based encryption from the diffie-hellman assumption. In: CRYPTO (2017)
15. Döttling, N., Garg, S., Hajiabadi, M., Masny, D.: New constructions of identity-based and key-dependent message secure encryption schemes. In: PKC 2018 (2018)
16. Döttling, N., Garg, S.: From selective ibe to full ibe and selective hibe. TCC (2017)
17. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: FOCS (2013)
18. Garg, S., Hajiabadi, M., Mahmoody, M., Rahimi, A.: Registration-based encryption: Removing private-key generator from IBE. In: TCC 2018 (2018)
19. Garg, S., Hajiabadi, M., Mahmoody, M., Rahimi, A., Sekar, S.: Registration-based encryption from standard assumptions. In: PKC (2019)
20. Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: STOC 2011 (2011)
21. Goldwasser, S., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. (1984)
22. Goyal, V.: Reducing trust in the PKG in identity based cryptosystems. In: CRYPTO (2007)
23. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: CCS '06 (2006)
24. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: ASIACRYPT (2010)
25. Kate, A., Goldberg, I.: Distributed private-key generators for identity-based cryptography. In: ICSCN (2010)
26. Lipmaa, H.: Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In: TCC (2012)
27. Micali, S.: CS proofs (extended abstracts). In: FOCS (1994)
28. Naor, M.: On cryptographic assumptions and challenges. In: CRYPTO (2003)
29. Paterson, K.G., Srinivasan, S.: Security and anonymity of identity-based encryption with multiple trusted authorities. In: Pairing-Based Cryptography (2008)
30. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM (2) (1978)
31. Rogaway, P.: The moral character of cryptographic work. Cryptology ePrint Archive, Report 2015/1162, https://eprint.iacr.org/2015/1162
32. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: EUROCRYPT (2005)
33. Shamir, A.: Identity-based cryptosystems and signature schemes. In: CRYPTO (1985)