Registration-Based Encryption: Removing Private-Key Generator from IBE

Sanjam Garg[1][⋆] Mohammad Hajiabadi[1,2][⋆⋆]
Mohammad Mahmoody[2][⋆⋆⋆] Ahmadreza Rahimi[2][†]

[1] University of California, Berkeley
[2] University of Virginia

Abstract. In this work, we introduce the notion of *registration-based encryption* (RBE for short) with the goal of removing the trust parties need to place in the private-key generator in an IBE scheme. In an RBE scheme, users sample their own public and secret keys. There will also be a "key curator" whose job is only to aggregate the public keys of all the registered users and update the "short" public parameter whenever a new user joins the system. Encryption can still be performed to a particular recipient using the recipient's identity and any public parameters released subsequent to the recipient's registration. Decryption requires some auxiliary information connecting users' public (and secret) keys to the public parameters. Because of this, as the public parameters get updated, a decryptor may need to obtain "a few" additional auxiliary information for decryption. More formally, if n is the total number of identities and κ is the security parameter, we require the following.
Efficiency requirements: (1) A decryptor only needs to obtain updated auxiliary information for decryption at most $O(\log n)$ times in its lifetime, (2) each of these updates are computed by the key curator in time $\mathrm{poly}(\kappa, \log n)$, and (3) the key curator updates the public parameter upon the registration of a new party in time $\mathrm{poly}(\kappa, \log n)$. Properties (2) and (3) require the key curator to have *random* access to its data.
Compactness requirements: (1) Public parameters are always at most $\mathrm{poly}(\kappa, \log n)$ bit, and (2) the total size of updates a user ever needs for decryption is also at most $\mathrm{poly}(\kappa, \log n)$ bits.
We present feasibility results for constructions of RBE based on indistinguishably obfuscation. We further provide constructions of *weakly efficient* RBE, in which the registration step is done in $\mathrm{poly}(\kappa, n)$, based on CDH, Factoring or LWE assumptions. Note that registration is done

[⋆] Research supported in part from DARPA/ARL SAFEWARE Award W911NF15C0210, AFOSR Award FA9550-15-1-0274, AFOSR YIP Award, DARPA and SPAWAR under contract N66001-15-C-4065, a Hellman Award and research grants by the Okawa Foundation, Visa Inc., and Center for Long-Term Cybersecurity (CLTC, UC Berkeley). The views expressed are those of the author and do not reflect the official policy or position of the funding agencies.
[⋆⋆] Supported by NSF award CCF-1350939 and AFOSR Award FA9550-15-1-0274.
[⋆⋆⋆] Supported by NSF CAREER award CCF-1350939, and two University of Virginia's SEAS Research Innovation Awards.
[†] Supported by NSF award CCF-1350939.

only once per identity, and the more frequent operation of generating updates for a user, which can happen more times, still runs in time poly($\kappa, \log n$). We leave open the problem of obtaining standard RBE (with poly($\kappa, \log n$) registration time) from standard assumptions.

1 Introduction

Public-key encryption [10,21,15] allows Alice to send Bob private messages without any a-priori shared secrets. However, before Alice can send any messages to Bob, she must obtain Bob's public key. Enabling Alice to obtain Bob's public key often requires additional public-key infrastructure and in some cases complex certification authorities; consequently, making implementation of public-key encryption rather cumbersome.

With the goal of simplifying key-management in public-key encryption, Shamir [23] introduced the notion of identity based encryption (IBE). An IBE scheme allows Alice to encrypt her messages to Bob knowing just the identity of Bob and some additional system public parameters. In this setup, Bob can then decrypt Alice's ciphertexts using an identity-specific secret key that he obtains from the private key generator (PKG). In their celebrated work, Boneh and Franklin [3] provided the first construction of IBE using bilinear maps. A long line of subsequent research has provided many other constructions of IBE based on a variety of assumptions [9,11]. IBE serves as the basis of several real-world systems (e.g., in systems by Voltage security) to simplify key-management.

Despite its significant advantages, one important limitation of IBE schemes is the so-called *key-escrow* problem. Namely, in an IBE scheme a PKG can generate the identity-specific secret key for any identity. This allows the PKG to arbitrarily decrypt messages that are intended for specific recipients. While in certain applications it is reasonable to place trust in a PKG, doing so is not *always* acceptable. This limitation of IBE often attracts significant criticism and restricts applicability in certain scenarios. In words of Rogaway [22],

> *"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 keyseven ones that haven't yet been issued. [...] Descriptions of IBE don't usually emphasize the change in trust model. And the key-issuing authority seems never to be named anything like that: it's just the PKG, for Private Key Generator. This sounds more innocuous than it is, and more like an algorithm than an entity."*

With the goal of enhancing the applicability of IBE, prior works suggested ways for reducing the level of trust that parties need to place in the PKG. Boneh and Franklin [3] suggested the use of multiple PKGs, instead of just one, with the goal of making the trust de-centralized. This idea was further explored in subsequent work (e.g., see [5,20,19]). In a different approach, Goyal [16], later followed by Goyal et al. [17], studied the notion of accountable IBE, which allows users to get their decryption keys from the PKG using a secure key generation protocol. Such schemes provide safeguard against a malicious PKG who might distribute the identity-specific secret key for a particular user to unauthorized parties, as by doing so it risks the possibility of being caught in the future. Another approach to the key escrow problem, studied in [6,8,24], involves settings in which the number of identities is huge, limiting the server's ability of finding out the receiver identity when it is chosen at random; hence, guaranteeing a form of anonymity. Finally, Al-Riyami and Paterson [1] put forward the notion of "Certificateless" Public Key Cryptography which is a hybrid of IBE and public-key directories, but which, on the down side, does not let the sender use the system as a true IBE, because more information about the user needs to be read from the public-key infrastructure before a message can be encrypted to them.

None of the above approaches, however, resolve the key-escrow problem entirely, as the PKG (or a collection of several of them) can still decrypt all ciphertexts in the system. Indeed even a trusted PKG may not be able to protect ciphertexts against a subpoena requesting decryption keys. This state of affairs leads us to the main question of this work:

Can we entirely remove PKG from IBE schemes?

A new primitive: registration-based encryption (RBE). In this work, we pursue a new approach to constructing IBE schemes by introducing a new notion which we call *registration-based encryption*, and which does not suffer from the key-escrow problem. Recall that in traditional IBE schemes, the PKG plays an active role in maintaining the cryptographic secrets corresponding to the public parameters of the system, leading to the key-escrow problem. Deviating from this approach, in our RBE we replace the PKG with a much weaker entity that we call a *key curator*. A key curator does not possess any cryptographic secrets and just plays the role of *aggregating* the public keys of the users.

In more detail, in an RBE scheme each user samples its own public key and secret key and provides its identity and the chosen public key to the key curator.[3] The key curator is now tasked with the goal of curating this new user's public key in the public parameters. Towards this, the key curator updates the public parameters and publicizes the new public parameters. Thus, unlike traditional IBE schemes, the public parameters in an RBE scheme evolve as new users register in the system. For example, let $\mathsf{pp}_0, \mathsf{pp}_1, \ldots, \mathsf{pp}_n$ be the different instances of the public parameters in the system, where pp_i is the public parameter after i users have registered in the system. Just like an IBE scheme, we require that the size of the public parameter is always small: $|\mathsf{pp}_i| \leq \mathrm{poly}(\kappa, \log n)$ for $i \leq n$, where κ is the security parameter and n is the number of users in the system.

In an RBE scheme, decryption by a user is performed using its secret key and some *auxiliary information* that connects its public key with system's public

[3] The key curator will need to verify the identity of the user requesting the registration as it is done by certification authorities in public-key infrastructure.

parameters. Note that as new users join the system and public parameters are updated, an update to the auxiliary information connecting a user's public key to the new public parameters is necessary.[4] However, it would be prohibitive to update each user's auxiliary information (needed for decryption) after each single registration. Thus, we require that the effect of registration by new users on the previously registered users is minimal. In particular, we require that a registered user needs to query the key curator for auxiliary information connecting its public key to the public parameters at most $O(\log n)$ times in its lifetime where n is the total number of registered users. Additionally, we require that the total size of the auxiliary information provided by the key curator needed for any decryption is at most $\mathrm{poly}(\kappa, \log n)$ for security parameter κ.

Our results. We consider two variants of RBE schemes based on the efficiency of the registration and give constructions for both of them. In particular, we construct (standard) RBE using indistinguishabiltiy obfuscation, and we construct a "weakly efficient" variant of this primitive based on more standard assumptions.

– *RBE based on IO*: First, we construct (standard) RBE schemes in which the running time of key curator for every new user registration is $\mathrm{poly}(\kappa, \log n)$ for security parameter κ assuming the key curator has *random* access to its auxiliary information. Other than the desired efficiency itself, one motivation for such minimization in curator's complexity is that since the work done in each user registration is small, it is then more reasonable to distribute the key curator's job between the users themselves, removing the need of a dedicated key curator entirely. In such a system, a new user will only need to do a "small" amount of *public* computation to update the public parameters at the time of joining the system. Moreover, any previously registered user could obtain its updated auxiliary information needed for decryption from the public ledger as well. We obtain a feasibility result for this notion based on somewhere statistically binding hash functions [18] and indistinguishably obfuscation [2,13].

– *RBE with weakly-efficient registration*: Second, we consider a setting where the key curator is allowed to be "weakly efficient"; i.e., the running time of key curator for updating the public parameters as a single new user registers can $\mathrm{poly}(\kappa, n)$. We call such RBE schemes weakly efficient and obtain a construction of weakly-efficient RBE based on any *hash garbling* scheme. The notion of hash garbling and its construction has been implicit in prior works [7,11,11,12,4], and it was shown there that hash garbling can be realized based on CDH, Factoring or LWE assumptions. In this work, we give a formal definition of this primitive (Definition 19) and use it to construct RBE.

Our two constructions above leave open the problem of constructing (standard) RBE with $\mathrm{poly}(\kappa, \log n)$ registration time based on standard assumptions.

Communication cost of RBE compared with PKE and IBE. We view RBE as a hybrid between PKE and traditional IBE. PKE schemes are com-

[4] Note that since the public parameters are small, they cannot contain the public keys of all the registered parties.

munication heavy for encryptors. In other words, each encryptor must obtain the public keys of each recipient that it sends encrypted messages to. In contrast, IBE schemes remove the need for the communication by the encryptors — specifically, encryptors no longer need to recover the public key of each user separately. However, the decryptor must still obtain its identity-specific secret key via communication with the PKG. Note that since this communication with PKG is only done once, the communication cost of an IBE is much smaller than the communication cost of a PKE. However, this efficiency comes at the cost of the key-escrow problem. Our RBE achieves, in large parts, the communication benefits of IBE without the key-escrow problem. More specifically, in an RBE, the encryptors do not need to recover the public key of each recipient individually. Additionally, a decryptor only needs to interact with the key curator to obtain the relevant updates at most $\log n$ times in total.

IBE was originally proposed with the goal of simplifying key management in IBE, yet the problem of key-escrow has prevented it from serving as a substitute for PKE — specifically, its applicability remains limited to specialized settings where trust is not a problem. We believe that efficient variants of our RBE constructions could indeed provide an alternative for PKE while also simplifying key management as IBE does.

1.1 Technical Overview

Here we describe the high level ideas behind our two constructions. The main challenge in realizing our RBE is to have the key curator gather together public keys of registering users in such a way that no individual's relation to the public parameter is affected too many times. Doing that is the key for having few necessary updates for decryption. We start by describing how we resolve this challenge using indistinguishability obfuscation (IO). Next, we give our ideas for realizing a (registration) weakly efficient version of this primitive based on standard assumptions such as CDH and Factoring. The IO-based construction, however, remains conceptually simpler and achieves all the desirable efficiency properties asymptotically.

Our IO based solution is inspired by prior works on using witness encryption [14], if we interpret the decryption key (i.e., the secret key together with the required auxiliary updates) as a witness that enables decryption. Additionally, both our IO-based and the hash obfuscation based solutions (and in particular their tree-based hashing of the public keys) use ideas developed recently in the context of laconic OT [7] and IBE from the CDH assumption [11]. In both of these settings, our contribution is in formalizing the subtle aspects of RBE and then realizing RBE schemes (as mentioned above) using these ideas.

High level description of our IO-based construction of RBE. A natural first try for the solution would be for the curator to just Merkle hash together the public keys of all the users in the system (along with their corresponding identities). Here encryption could be performed by an obfuscation of the following program $\mathrm{P}[\mathsf{h}, \mathsf{m}]$, with the Merkle hash root h and the encrypted message

m hardwired. Given input $(\mathsf{pk}, \mathsf{id}, \mathsf{pth})$, the program $\mathrm{P}[\mathsf{h}, \mathsf{m}]$ outputs an encryption of m under the public key pk *only if* pth is a "Merkle opening" (i.e., the right leaf to root path with siblings) for $(\mathsf{pk}, \mathsf{id})$ as a pair of sibling leaves in the Merkle hash tree with root h, and it outputs $\perp$ otherwise. Decryption can proceed naturally with the right Merkle opening as auxiliary information that the key curator needs to provide for decryption. The main issue with this solution is that the Merkle hash root h changes with every new user registering in the system. Our idea for solving this problem is to maintain *multiple* Merkle hash trees such that any individual user is affected only a bounded number of times. Below, we explain this idea in more detail.

– Public parameters and auxiliary information. At a high level, in our construction, after n parties have registered, the key curator holds an auxiliary information aux_n of the following form: it consists of η *full* binary Merkle trees, $\mathsf{Tree}_1, \ldots, \mathsf{Tree}_\eta$ with corresponding depths $\mathsf{d}_1 > \cdots > \mathsf{d}_\eta$ and number of leaves $2^{\mathsf{d}_1}, \ldots, 2^{\mathsf{d}_\eta}$. The public parameter would be the set of the labels of the *roots* of these trees. Every leaf in either of these trees is either an identity id or its public key pk as the sibling of the leaf id, and every registered identity id appears exactly once as a leaf. Thus, half of the leaves of these trees contain the strings encoding the registered identities, and for each leaf id, the sibling leaf contains the public key pk of id. So, if there are n people registered so far in the system, then the total number of leaves in the trees is equal to $2n$. Since we stated that $\mathsf{d}_1 > \cdots > \mathsf{d}_\eta$, it means that the number of these trees η is at most $\log(n)$, simply because $(d_1, \ldots, d_\eta)$ would be the binary representation of number $2n$. This point implies that the public parameter is indeed short.

– What is needed for decryption. Even though in general it is more natural to describe encryption first, in our case it is easier to describe the information that is needed for decryption. Each identity id will hold is own secret key sk which will be necessary for decryption, but it would need more information for doing so. Indeed, if Tree is the tree hold by the curator that contains (sibling leaves) $(\mathsf{id}, \mathsf{pk})$ in its leaves, then the identity id needs to know the "Merkle opening" of $(\mathsf{id}, \mathsf{pk})$ to the root of Tree in order to do any decryption. Since the length of this path is at most the depth of Tree, which is at most $\log(n)$, the total size of the decryption key dk (which includes sk and the knowledge of such opening to the root of Tree) is at most $\kappa \cdot \log(n)$. This makes dk also short enough.

– How to encrypt. For simplicity, suppose there is only one tree Tree held by the key curator and that all the identities are leaves of this tree. The encryptor, knows the public parameter, which is the root rt of Tree. For any message m, the encryptor then sends the *obfuscation* of the following program P. The program P takes as input any Merkle opening that contains the path from leaves $(\mathsf{id}, \mathsf{pk})$ to the root rt of Tree, and if such opening is given, then P outputs an encryption of m under the corresponding registered public key pk. Since id is the only identity who knows the corresponding sk to the registered pk, nobody other than id can decrypt the message m encrypted that way. When there are *multiple* trees $\mathsf{Tree}_1, \ldots, \mathsf{Tree}_\eta$ held by the key curator, the ciphertext includes η obfuscations, one for every Tree_i.

– How to register. When a new party id joins to register, we first create a single tree Tree for that party, with id, pk as its only leaves. But creating too many trees naively increases the length of the public parameter. So, to handle this issue we "merge" the trees every now and then. In particular, upon any registration, so long as there are any two trees $\mathsf{Tree}_1, \mathsf{Tree}_2$ of the *same size* held by the key curator, it "merges" them by simply hashing their roots $\mathsf{rt}_1, \mathsf{rt}_2$ into a new root rt. This way, the key curator keeps the invariance property (stated above) that the trees are always full binary trees of different sizes. After doing any such merge, the key curator sends the the generated update of the form $(\mathsf{rt}_1, \mathsf{rt}, \mathsf{rt}_2)$ to all of the identities that are in *either* of the trees $\mathsf{Tree}_1, \mathsf{Tree}_2$. That is because, the identities in Tree_1 would now need to know rt_2 and the identities in Tree_2 now need the label rt_1 in order to decrypt what is encrypted for them. Alternatively, if the key curator is passive and does not send updates, the users who are in the merged tree Tree would need to pull their updates whenever they have a ciphertext that they cannot decrypt, realizing that their auxiliary information is outdated.

To prove security of the above construction, collision-resistance of the used hash function is not enough, and we rely on *somewhere statistically binding* hash functions [18] (see Definition 3).

Weakly-efficient construction based on standard assumptions. In order to replace the use of obfuscation in the above construction, we build on the techniques by Cho, Döttling, Garg, Gupta, Miao, and Polychroniadou [7] and Döttling and Garg [11]. We abstract their idea of using hash encryption and garbled circuits as a new primitive that we call *hash garbling*. Use of this abstraction simplifies exposition. A hash garbling scheme consists of algorithms $(\mathrm{Hash}, \mathrm{HG}, \mathrm{HInp})$.[5] Hash function is a function from $\{0,1\}^\ell$ to $\{0,1\}^\kappa$. HG takes as input a secret state stt and an arbitrary program P and outputs $\widetilde{\mathrm{P}}$. HInp takes as input a secret state stt and a value $y \in \{0,1\}^\kappa$ and outputs $\widetilde{y}$. Correctness and security require that $\widetilde{\mathrm{C}}, \widetilde{y}, x$ can be used to compute $\mathrm{C}(x)$, but also that they reveal nothing else about C.

Our construction of RBE from standard assumption is very similar to the IO-based construction except that we replace the use of IO with the less powerful primitive hash garbling. The key challenge in making this switch comes from the fact that hash garbling, unlike IO, cannot process *the entire* root to leaf Merkle opening in one shot. Thus, our construction needs to provide a sequence of hash garblings that traverse the root to leaf path step by step. Therefore, as the tree is being traversed, the hash garblings need to identify whether to go left or to go right. Note that this decision must be taken without any knowledge of what identities are included in the leaves of the left sub-tree and what identities are included in the leaves of the right sub-tree. We resolve this challenge by modifying the Merkle tree in two ways:

1. We ensure that the identities in the leave of any tree are always sorted.

[5] The hash function also has a key setup function which we ignore here for the sake of simplicity.

2. In addition to the hashes of its two children, in the computation of the Merkle hash, we also hash the information about the largest identity that is present any leaf of the left subtree at any node. (The latter information allows us to traverse down a Merkle tree using it as a binary search tree.)

Using these enhancements over the simple Merkle trees, we can indeed substitute IO with the less powerful primitive of hash garbling, which in turn can be obtained from more standard assumptions. On the down side, this new construction needs to sort the identities for every registration, and in particular the registration cannot run in sublinear time $\text{poly}(\kappa, \log n)$. We refer the reader Section 5 for more details on this construction.

2 Preliminaries

Notation. For a probabilistic algorithm A, by $A(x) \to y$, we denote the randomized process of running A on input x and obtaining the output y. We use PPT to denote a probabilistic polynomial-time algorithms, where running time is polynomial over the length of their main input (not the random seed). For randomized algorithms $A_1, A_2, \ldots$, by $\Pr_{A_1,A_2,\ldots}[E]$ we denote the probability of event E when the randomness is over the algorithms $A_1, A_2, \ldots$ as well. For deterministic algorithms A_1, A_2, by $A_1 \equiv A_2$, we denote that they have the same input-output functionality; namely, for all x (of the right length, if A_1, A_2 are circuits), $A_1(x) = A_2(x)$. For distribution ensembles X_n, Y_n, by $X_n \overset{c}{\approx} Y_n$ we mean that they are indistinguishable against $\text{poly}(n)$-time algorithms. By $x||y$ we denote the concatenation of the strings x, y. By $\text{negl}(\kappa)$ we denote some function that is negligible in input κ; namely for all k, $\text{negl}(\kappa) \le O(1/\kappa^k)$. U_n denotes the uniform distribution over $\{0,1\}^n$. For algorithm A, by A^B we denote an oracle access by A to oracle B. By $A^{[B]}$ we denote A accessing oracle B with read and *and write* operations. So, if A writes y at location x, reading a query x next time will return y.

Definition 1 (Public key encryption). *A public key encryption scheme consists of three PPT algorithms* $(\mathrm{G}, \mathrm{E}, \mathrm{D})$ *as follows.*

– $\mathrm{G}(1^\kappa) \to (\mathsf{pk}, \mathsf{sk})$*: This algorithm takes a security parameter* 1^κ *as input and outputs a pair of public key* pk *secret key* sk*. Without loss of generality we assume that* $|\mathsf{pk}| = |\mathsf{sk}| = \kappa$*.*
– $\mathrm{E}(\mathsf{pk}, \mathsf{m}) \to \mathsf{ct}$*: takes a message* m *and a public key* pk *as input and outputs a ciphertext* ct*.*
– $\mathrm{D}(\mathsf{sk}, \mathsf{ct}) \to \mathsf{m}$*: takes a ciphertext* ct *and a secret key* sk *as inputs and outputs a message* m*.*

The completeness and security properties are defined as follows.

– **Completeness.** *The PKE scheme is complete if for every message* m*:*

$$\Pr_{\mathrm{G,E,D}}[\mathrm{D}(\mathsf{sk}, \mathrm{E}(\mathsf{pk}, \mathsf{m})) = \mathsf{m} : (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathrm{G}] = 1.$$

- **Semantic Security.** *Any PPT adversary* Adv *wins the following game with probability* $\frac{1}{2} + \mathrm{negl}(\kappa)$*:*
  - *The challenger generates* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathrm{G}(1^\kappa)$ *and sends* pk *to* Adv*.*
  - *The challenger chooses a random bit* b *and sends* $c \leftarrow \mathrm{E}(\mathsf{pk}, b)$ *to* Adv*.*
  - Adv *outputs* b' *and wins if* $b = b'$*.*

Definition 2 (Indistinguishability obfuscation). *A uniform PPT algorithm* Obf *is called an* indistinguishability obfuscator *for a circuit class* $\{\mathcal{C}_\kappa\}_{\kappa \in \mathbb{N}}$ *(where each* $\mathcal{C}_\kappa$ *is a set indexed by a security parameter* κ*) if the following holds:*

- **Completeness.** *For all security parameters* $\kappa \in \mathbb{N}$ *and all circuits* $\mathrm{C} \in \mathcal{C}_\kappa$*, we obtain an obfuscation with the same function:*

$$\Pr_{\mathrm{Obf}}[\mathrm{C}' \equiv \mathrm{C} : \mathrm{C}' = \mathrm{Obf}(1^\kappa, \mathrm{C})] = 1.$$

- **Security.** *For any PPT distinguisher* D*, there exists a negligible function* $\mathrm{negl}(\cdot)$ *such that for all* $\kappa \in \mathbb{N}$*, for all pairs of functionally equivalent circuits* $\mathrm{C}_1 \equiv \mathrm{C}_2$ *from the same family* $\mathrm{C}_1, \mathrm{C}_2 \in \mathcal{C}_\kappa$*,*

$$\left| \Pr_{\mathrm{Obf}}[D(1^\kappa, \mathrm{Obf}(1^\kappa, \mathrm{C}_1)) = 1)] - \Pr_{\mathrm{Obf}}[D(1^\kappa, \mathrm{Obf}(1^\kappa, \mathrm{C}_2)) = 1)] \right| \leq \mathrm{negl}(\kappa).$$

The next definition is a special case of the definition of somewhere statistically binding (SSB) hash functions introduced by Hubacek and Wichs [18] for the blockwise setting. Here we only use two-input blocks.

Definition 3 (SSB hash functions [18]). *A* somewhere statistically binding hash system *consists of two polynomial time algorithms* HGen, Hash*.*

- $\mathrm{HGen}(1^\kappa, b) \rightarrow \mathsf{hk}$*. This algorithm takes the security parameter* κ *and an index bit* $b \in \{0, 1\}$*, and outputs a hash key* hk*.*
- $\mathrm{Hash}(\mathsf{hk}, x) \rightarrow y$*. This is a deterministic algorithm that takes as input* $x = (x_0, x_1) \in \{0,1\}^\kappa \times \{0,1\}^\kappa$ *and outputs* $y \in \{0,1\}^\kappa$*.*

We require the following properties for an SSB hashing scheme:

- **Index hiding.** *No* $\mathrm{poly}(\kappa)$*-time adversary can distinguish between* hk_0 *and* hk_1 *by more than* $\mathrm{negl}(\kappa)$*, where* $\mathsf{hk}_b \leftarrow \mathrm{HGen}(1^\kappa, b)$ *for* $b \in \{0, 1\}$*.*
- **Somewhere statistically binding.** *We say that* hk *is statistically binding for index* $i \in \{0,1\}$*, if there do* not *exist two values* $(x_0, x_1), (x'_0, x'_1) \in \{0,1\}^\ell \times \{0,1\}^\ell$ *such that* $x_i \neq x'_i$ *and* $\mathrm{Hash}(\mathsf{hk}, x) = \mathrm{Hash}(\mathsf{hk}, x')$*. We require that for both* $i \in \{0,1\}$*,*

$$\Pr_{\mathrm{HGen}}[\mathsf{hk} \textit{ is statistically binding for } i : \mathsf{hk} \leftarrow \mathrm{HGen}(1^\kappa, i)] \geq 1 - \mathrm{negl}(\kappa).$$

3 Formal Definition of Registration-Based Encryption

In this section, we formalize the new notion of RBE. After defining the "default" version of RBE, we define weakened forms of this primitive with a specific relaxation in the efficiency requirements. The goal of this relaxation is to base the (relaxed) RBE on more standard assumptions.

We start by defining the syntax of the default notion of RBE. We will then discuss the required compactness, completeness, and security properties.

Definition 4 (Syntax of RBE). *A* registration-based encryption *(RBE for short) scheme consists of PPT algorithms* (Gen, Reg, Enc, Upd, Dec) *working as follows. The* Reg *and* Upd *algorithms are performed by the key curator, which we call KC for short.*

– **Generating common random string.** *Some of the subroutines below will need a common random string* crs*, which could be sampled publicly using some public randomness beacon.* crs *of length* $\mathrm{poly}(\kappa)$ *is sampled at the beginning, for the security parameter* κ*.*
– **Key generation.** $\mathrm{Gen}(1^\kappa) \to (\mathsf{pk}, \mathsf{sk})$*: The randomized algorithm* Gen *takes as input the security parameter* 1^κ *and outputs a pair of public/secret keys* $(\mathsf{pk}, \mathsf{sk})$*. Note that these are only* public *and* secret *keys, not the* encryption *or* decryption *keys. The key generation algorithm is run by any honest party locally who wants to register itself into the system.*
– **Registration.** $\mathrm{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk}) \to \mathsf{pp}'$*: The deterministic*[6] *algorithm* Reg *takes as input the common random sting* crs*, current public parameter* pp*, a registering identity* id *and a public key* pk *(supposedly for the identity* id*), and it outputs* pp' *as the updated public parameters. The* Reg *algorithm uses* read and write *oracle access to* aux *which will be updated into* aux' *during the process of registration.*[7] *(The system is initialized with public parameters* pp *and auxiliary information* aux *set to* $\perp$*.)*
– **Encryption.** $\mathrm{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{m}) \to \mathsf{ct}$*: The randomized algorithm* Enc *takes as input the common random sting* crs*, a public parameter* pp*, a recipient identity* id *and a plaintext message* m *and outputs a ciphertext* ct*.*
– **Update.** $\mathrm{Upd}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{id}) \to \mathsf{u}$*: The deterministic algorithm* Upd *takes as input the current information* pp *stored at the KC and an identity* id*, has* read only *oracle access to* aux *and generates an* update *information* u *that can help* id *to decrypt its messages.*[8]
– **Decryption.** $\mathrm{Dec}(\mathsf{sk}, \mathsf{u}, \mathsf{ct})$*: The deterministic decryption algorithm* Dec *takes as input a secret key* sk*, an update information* u*, and a ciphertext* ct*, and it outputs a message* $\mathsf{m} \in \{0,1\}^*$ *or in* $\{\perp, \texttt{GetUpd}\}$*. The special symbol* $\perp$ *indicates a syntax error, while* $\texttt{GetUpd}$ *indicates that more recent update information (than* u*) might be needed for decryption.*

Remark 5 (Key curator is transparent). We emphasize that in the definition above the KC has no secret state. In fact, the registration and update operations are both *deterministic*. This makes KC's job fully auditable. Even the generation of the crs (that is done before KC takes control of the server's information) only

[6] In our constructions, the algorithms Reg, Upd and Reg are deterministic, and this feature makes our KC transparent (see Remark 5), so we keep the default definition based on deterministic version of these subroutines.

[7] This is the step that needs the identity of the registering id to be verified. This verification step is similar to IBE and its details are outside scope of this work.

[8] Looking ahead, we will aim for schemes that require the identity id to launch this request as rarely as possible. However, we note that this information u does not need to be kept secret for the security of the scheme, and any user can request this update without its identity being checked.

needs common *random* strings (as opposed to a common *reference* string), so that can be generated using public randomness beacon as well.

We will now first describe the *completeness*, *compactness*, *efficiency* properties (under the completeness definition) and then we will describe the *security* properties. Both definitions are based on a security game that involves an "adversary" that tries to break the security, completeness, compactness, or efficiency properties by controlling how the identities (including the target/challenge identity) are registered and when the encryptions and decryptions happen.

Definition 6 (Completeness, compactness, and efficiency of RBE). *For any interactive* computationally unbounded *adversary* Adv *that still has a limited* $\mathrm{poly}(\kappa)$ *round complexity, consider the following game* $\mathtt{Comp}_{\mathsf{Adv}}(\kappa)$ *between* Adv *and a challenger* Chal.

1. ***Initialization.*** Chal *sets* $\mathsf{pp} = \bot$, $\mathsf{aux} = \bot$, $\mathsf{u} = \bot$, $\mathcal{D} = \varnothing$, $\mathsf{id}^* = \bot$, $t = 0$, $\mathsf{crs} \leftarrow U_{\mathrm{poly}(\kappa)}$ *and sends the sampled* crs *to* Adv.
2. *Till* Adv *continues (which is at most* $\mathrm{poly}(\kappa)$ *steps), proceed as follows. At every iteration,* Adv *chooses exactly one of the actions below to be performed.*
   (a) ***Registering new (non-target) identity.*** Adv *sends some* $\mathsf{id} \notin \mathcal{D}$ *and* pk *to* Chal. Chal *registers* $(\mathsf{id}, \mathsf{pk})$ *by letting* $\mathsf{pp} := \mathrm{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk})$ *and* $\mathcal{D} := \mathcal{D} \cup \{\mathsf{id}\}$.
   (b) ***Registering the target identity.*** *If* id^* *was chosen by* Adv *already (i.e.,* $\mathsf{id}^* \neq \bot$*), skip this step. Otherwise,* Adv *sends some* $\mathsf{id}^* \notin \mathcal{D}$ *to* Chal. Chal *then samples* $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathrm{Gen}(1^\kappa)$, *makes the updates* $\mathsf{pp} := \mathrm{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*), \mathcal{D} := \mathcal{D} \cup \{\mathsf{id}^*\}$, *and sends* pk^* *to* Adv.
   (c) ***Encrypting for the target identity.*** *If* $\mathsf{id}^* = \bot$ *then skip this step. Otherwise,* Chal *sets* $t = t + 1$, *then* Adv *sends some* $\mathsf{m}_t \in \{0,1\}^*$ *to* Chal *who then sets* $\mathsf{m}'_t := \mathsf{m}_t$ *and sends back a corresponding ciphertext* $\mathsf{ct}_t \leftarrow \mathrm{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{m}_t)$ *to* Adv.
   (d) ***Decryption by target identity.*** Adv *sends a* $j \in [t]$ *to* Chal. Chal *then lets* $\mathsf{m}'_j = \mathrm{Dec}(\mathsf{sk}^*, \mathsf{u}, \mathsf{ct}_j)$. *If* $\mathsf{m}'_j = \mathtt{GetUpd}$, *then* Chal *obtains the update* $\mathsf{u} = \mathrm{Upd}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{id}^*)$ *and then lets* $\mathsf{m}'_j = \mathrm{Dec}(\mathsf{sk}^*, \mathsf{u}, \mathsf{ct}_j)$.
3. *The adversary* Adv *wins the game if there is some* $j \in [t]$ *for which* $\mathsf{m}'_j \neq \mathsf{m}_j$.

Let $n = |\mathcal{D}|$ *be the number of identities registered till a specific moment. We require the following properties to hold for any* Adv *(as specified above) and for* all *the moments (and so for all the values of* $\mathcal{D}$ *and* $n = |\mathcal{D}|$ *as well) during the game* $\mathtt{Comp}_{\mathsf{Adv}}(\kappa)$.

- **Completeness.** $\Pr[\mathsf{Adv}$ *wins in* $\mathtt{Comp}_{\mathsf{Adv}}(\kappa)] = \mathrm{negl}(\kappa)$.
- **Compactness of public parameters and updates.** $|\mathsf{pp}|, |\mathsf{u}|$ *are both* $\leq \mathrm{poly}(\kappa, \log n)$.
- **Efficiency of runtime of registration and update.** *The running time of each invocation of* Reg *and* Upd *algorithms is at most* $\mathrm{poly}(\kappa, \log n)$. *(This implies the compactness property.)*

– **Efficiency of the number of updates.** *The* total *number of invocations of* Upd *for identity* id^* *in Step 2d of the game* $\mathit{Comp}_{\mathsf{Adv}}(\kappa)$ *is at most* $O(\log n)$ *for every* n *during* $\mathit{Comp}_{\mathsf{Adv}}(\kappa)$*.*

Remark 7 (Other definitions based on quantifying compactness and efficiency parameters). Even though Definition 6 requires compactness and efficiency requirements using function $c(\kappa, n) \leq \text{poly}(\kappa, \log n)$, one can consider a more general definition that uses different (e.g., sublinear) functions to obtain various versions of RBE. In general, one can consider $(c_1, \ldots, c_5)$-RBE schemes where c_i's are functions of (κ, n), and that functions c_1, c_2 describe the compactness requirements (of public-key and updates), and functions c_3, c_4, c_5 describe the efficiency requirements.

The following definition instantiates the general quantified definition of Remark 7 by relaxing the efficiency of the registration and keeping the other efficiency and compactness requirements to be as needed for Definition 6.

Definition 8 (WE-RBE). *A* registration weakly efficient RBE *(or WE-RBE for short) is defined similarly to Definition 6, where the specified* $\text{poly}(\kappa, \log n)$ *runtime efficiency of the registration algorithm is not required anymore, but instead we require the registration time to be* $\text{poly}(\kappa, n)$*.*

Remark 9 (Denial of service attacks using fake ciphertexts). A class of malicious adversaries that are *not* captured by Definition 6 can potentially launch a "denial of service" attack against the efficiency of the decryption procedure as follows. Specifically, such malicious completeness adversary (that can also be seen as a form of "environment") can cause an honest user to request too many updates by continually providing it with fake ciphertexts that seem to require an update for decryption. Here, we propose a generic approach for dealing with this issue. We can generalize the RBE primitive and allow the KC to have a secret state. This will take away the appealing transparency feature of the KC, but it will instead allow the KC to sign the public parameters, and those signed public parameters can then be included in the ciphertexts. Doing this will allow the decryption algorithm to detect fake ciphertexts that (maliciously) indicate that the population has grown beyond the last update, and that new update is needed for recent decryptions.

Security. For security, we require that no PPT adversary should be able to distinguish between encryptions of two messages (of equal lengths) made to a user who has registered honestly into the system, even if the adversary colludes and obtains the secret keys of all the other users. This is formalized by the adversary specifying a challenge identity and distinguishing between encryptions made to that identity. In order to prevent the adversary from winning trivially, we require that the adversary does not know any secret key for a public key registered for the challenge identity.

We present the formal definition only for the case of bit encryption, but any scheme achieving this level of security can be extended to arbitrary length messages using independent bit-by-bit encryption and a standard hybrid argument.

Definition 10 (Security of RBE). *For any interactive PPT adversary* Adv*, consider the following game* $\mathit{Sec}_{\mathsf{Adv}}(\kappa)$ *between* Adv *and a challenger* Chal*. (Steps that are different from the completeness definition are denoted with purple stars* ($\star\star$)*. Specifically, Steps 2c and 2d from Definition 6 are replaced by Step 3 below. Additionally, Step 3 from Definition 6 is replaced by Step 4 below.)*

1. ***Initialization.*** Chal *sets* $\mathsf{pp} = \bot$, $\mathsf{aux} = \bot$, $\mathcal{D} = \varnothing$, $\mathsf{id}^* = \bot$, $\mathsf{crs} \leftarrow U_{\mathrm{poly}(\kappa)}$ *and sends the sampled* crs *to* Adv*.*
2. *Till* Adv *continues (which is at most* $\mathrm{poly}(\kappa)$ *steps), proceed as follows. At every iteration,* Adv *chooses exactly one of the actions below to be performed.*
   (a) ***Registering new (non-target) identity.*** Adv *sends some* $\mathsf{id} \notin \mathcal{D}$ *and* pk *to* Chal*.* Chal *registers* $(\mathsf{id}, \mathsf{pk})$ *by letting* $\mathsf{pp} := \mathrm{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk})$ *and* $\mathcal{D} := \mathcal{D} \cup \{\mathsf{id}\}$*.*
   (b) ***Registering the target identity.*** *If* id^* *was chosen by* Adv *already (i.e.,* $\mathsf{id}^* \neq \bot$*), skip this step. Otherwise,* Adv *sends some* $\mathsf{id}^* \notin \mathcal{D}$ *to* Chal*.* Chal *then samples* $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathrm{Gen}(1^\kappa)$*, makes the updates* $\mathsf{pp} := \mathrm{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*), \mathcal{D} := \mathcal{D} \cup \{\mathsf{id}^*\}$*, and sends* pk^* *to* Adv*.*
3. ($\star\star$) **Encrypting for the target identity.** If no id^* was chosen by Adv before (i.e., $\mathsf{id}^* = \bot$) then Adv first sends some $\mathsf{id}^* \notin \mathcal{D}$ to Chal. Next, Chal generates $\mathsf{ct} \leftarrow \mathrm{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, b)$, where $b \leftarrow \{0, 1\}$ is a random bit, lets $\mathcal{D} = \mathcal{D} \cup \{\mathsf{id}^*\}$, and sends ct to Adv.
4. ($\star\star$) The adversary Adv outputs a bit b' and wins the game if $b = b'$.

We call an RBE scheme secure if $\Pr[\mathsf{Adv}$ *wins in* $\mathit{Sec}_{\mathsf{Adv}}(\kappa)] < \frac{1}{2} + \mathrm{negl}(\kappa)$ *for any PPT* Adv*.*

Equivalence to other definitions. One might consider a seemingly stronger security definition in which the adversary chooses its challenge identity from a *set* of previously chosen identities for which it does *not* know the keys. However, since the adversary can *guess* its own selection with probability $1/\mathrm{poly}(\kappa)$, that definition becomes equivalent to Definition 10 above. Another seemingly stronger definition would allow the adversary to register even more identities after receiving the challenge ciphertext (and before answering the challenge), however this is again an equivalent definition as the information distributed in this extra step is simulatable by the adversary and thus not helpful to her.

Choosing a registered or an unregistered identity. Here we note a subtle aspect of Definition 10. If the adversary chooses Step 2b, it means that it is attacking a target identity that is registered in the system. Otherwise, the adversary shall choose the target identify in Step 3, which means that the attacked target identity is not even registered in the system. In both cases, we require that the adversary has negligible advantage in guessing the encrypted bit.

Why not giving update oracle to adversary? In Definition 10, we did not provide explicit oracle access to Upd subroutine for the adversary. The reason is that the adversary receives the crs, chooses the identities and receives the public keys. Moreover, KC is deterministic, has no secret state, and all the inputs it receives in maintaining the auxiliary information is crs, identities, and the public-keys. Therefore, throughout the attack, the adversary knows the exact state of $(\mathsf{pp}, \mathsf{aux})$ hold by the key curator, and thus it can run the update operation itself. However, if one considers a KC with a *secret state* (perhaps for the goal of signing the public parameters as discussed in Remark 9) then the corresponding security definition shall give the adversary oracle access to the update subroutine.

Remark 11 (Unauthorized registration of an identity). A malicious KC K^*, not following the protocol as modeled in the security game of Definition 10 can generate a pair of keys $(\mathsf{pk}, \mathsf{sk})$ on its own and register pk on behalf of an identity id. By that, K^* can read messages that are subsequently encrypted to the identity id. Here we describe two approaches to tackle this problem.

1. **Bootstrapping public-key directories.** RBE schemes could be launched with respect to an external public-key directory D. Namely, only public-keys in D could be registered for matching identities. This way, a malicious key curator K^* can only register the *actual* public keys of the identities, and thus it is not able to decrypt the messages encrypted to them. Moreover, by also including (public) verification keys of the signatures by the identities in the public-key directory D, we can even prevent K^* from successfully registering any identities in the RBE scheme without having their permission (even by using their real public keys) as follows. Whenever the public parameter pp is updated, a signature of pp by the registering identity is added to the public auxiliary aux. This way, a public auditor can detect a fake registration.
2. **Proof of Knowledge.** An alternative method to prevent fake identity registrations is to use a similar approach to the one mentioned above, but replace the signature with a zero-knowledge proof of knowledge of an actual certificate from some trusted party (e.g., their driving licence information) that validates the ownership of an identity.

4 IO-Based Construction of RBE

In this section we present a formal construction of (efficient) RBE based on indistinguishability obfuscation and SSB hash functions (see Section 2 for formal definitions of the standard primitives used). We first describe the construction along the line of Definition 4 and then will prove its completeness, compactness, and security based on Definitions 6 and 10. We will then describe minor modifications that make the construction efficient according to Definition 8 (basically by not producing the updates in the registration).

Notation on binary trees. In our construction below, Tree is always a *full* binary tree (with 2^i leaves for some i), where the label of each node in Tree is calculated as the "hash" of its left and right children. We define the size of a tree Tree as the number of its *leaves*, denoted by $\mathrm{size}(\mathsf{Tree})$ (so if $\mathrm{size}(\mathsf{Tree}) = s$, the total number of nodes will be $2s - 1$), and we denote the root of Tree as $\mathsf{rt}(\mathsf{Tree})$, and we use $\mathsf{d}(\mathsf{Tree})$ to refer to the depth of Tree. Since we assume that Tree is always a full tree, we always have $2^{\mathsf{d}(\mathsf{Tree})} = \mathrm{size}(\mathsf{d}(\mathsf{Tree}))$. When it is clear from the context, we use rt and d to denote the root and the depth of Tree.

Simplifying assumption on lengths. We note that without loss of generality, we can assume that public keys, secret keys and identities are all of the length security parameter κ.

Construction 12 (RBE from IO and SSB Hashing) *We will use an IO scheme* $(\mathrm{Obf}, \mathrm{Eval})$ *and a SSB hash function system* $(\mathrm{Hash}, \mathrm{HGen})$ *and a PKE scheme* $(\mathrm{G}, \mathrm{E}, \mathrm{D})$*. Using them, we show how to implement the subroutines of RBE according to Definition 4.*

- $\mathrm{Stp}(1^\kappa) \to (\mathsf{pp}_0, \mathsf{aux}_0)$*. This algorithm outputs* $\mathsf{pp}_0 = (\mathsf{hk}_1, \dots, \mathsf{hk}_\kappa)$ *where each* hk_i *is sampled from* $\mathrm{HGen}(1^\kappa, 0)$ *and* $\mathsf{aux} = \varnothing$ *is empty.*
- $\mathrm{Reg}^{[\mathsf{aux}]}(\mathsf{pp}_n, \mathsf{id}, \mathsf{pk}) \to \mathsf{pp}_{n+1}$*. This algorithm works as follows:*
  1. *Parse* $\mathsf{aux} := ((\mathsf{Tree}_1, \dots, \mathsf{Tree}_\eta), (\mathsf{id}_1, \dots, \mathsf{id}_n))$ *where the trees have corresponding depths* $\mathsf{d}_1 > \mathsf{d}_2 \cdots > \mathsf{d}_\eta$*, and* $(\mathsf{id}_1, \dots, \mathsf{id}_n)$ *is the order by which the current identities have registered.*[9]
  2. *Parse* pp_n *as a sequence* $((\mathsf{hk}_1, \dots, \mathsf{hk}_\kappa), (\mathsf{rt}_1, \mathsf{d}_1), \dots, (\mathsf{rt}_\eta, \mathsf{d}_\eta))$ *where* $\mathsf{rt}_i \in \{0,1\}^\kappa$ *represents the root of* Tree_i*, and* d_i *represents the depth of* Tree_i*.*
  3. *Create new tree* $\mathsf{Tree}_{\eta+1}$ *with leaves* id, pk *and set its root as* $\mathsf{rt}_{\eta+1} := \mathrm{Hash}(\mathsf{hk}_1, \mathsf{id}||\mathsf{pk})$ *and thus its depth would be* $\mathsf{d}_{\eta+1} = 1$*.*
  4. *Let* $\mathcal{T} = \{\mathsf{Tree}_1, \dots, \mathsf{Tree}_{\eta+1}\}$*. (We will keep changing* $\mathcal{T}$ *in steps below.)*
  5. *While there are two different trees* $\mathsf{Tree}_L, \mathsf{Tree}_R \in \mathcal{T}$ *of the same depth* d *and same size* $s = 2^{\mathsf{d}}$ *(recall that our trees are always full binary trees), and roots* $\mathsf{rt}_L, \mathsf{rt}_R$*, do the following.*
     (a) *Let* Tree *be a new tree of depth* $d+1$ *that contains* Tree_L *as its left subtree,* Tree_R *as its right subtree, and* $\mathsf{rt} = \mathrm{Hash}(\mathsf{hk}_{d+1}, \mathsf{rt}_L||\mathsf{rt}_R)$ *as its root.*
     (b) *Remove both of* $\mathsf{Tree}_L, \mathsf{Tree}_R$ *from* $\mathcal{T}$ *and add* Tree *to* $\mathcal{T}$ *instead.*
  6. *Let* $\mathcal{T} := (\mathsf{Tree}_1, \dots, \mathsf{Tree}_\zeta)$ *be the final set of trees where* $\mathsf{d}'_1 > \cdots > \mathsf{d}'_\zeta$ *are their corresponding depths and* $\mathsf{rt}'_1, \dots, \mathsf{rt}'_\zeta$ *are their corresponding roots. Set* pp_{n+1} *and* aux *as follows:*

$$\mathsf{pp}_{n+1} := ((\mathsf{hk}_1, \dots, \mathsf{hk}_\kappa), (\mathsf{rt}'_1, \mathsf{d}'_1), \dots, (\mathsf{rt}'_\zeta, \mathsf{d}'_\zeta)) \text{ and}$$

$$\mathsf{aux} := (\mathcal{T}, (\mathsf{id}_1, \dots, \mathsf{id}_n, \mathsf{id}_{n+1} = \mathsf{id})).$$

- $\mathrm{Enc}(\mathsf{pp}, \mathsf{id}, \mathsf{m}) \to \mathsf{ct}$*: First parse* $\mathsf{pp} := ((\mathsf{hk}_1, \dots, \mathsf{hk}_\kappa), (\mathsf{rt}_1, \mathsf{d}_1), \dots, (\mathsf{rt}_\eta, \mathsf{d}_\eta))$*. Generate programs* $\mathrm{P}_1, \dots, \mathrm{P}_\eta$ *where each program* P_i *works as follows:*
  Hardwired values: $\mathsf{rt}_i, \mathsf{d}_i, (\mathsf{hk}_1, \dots, \mathsf{hk}_{\mathsf{d}_i}), \mathsf{m}, \mathsf{id}, \mathsf{r}$ *(the randomness)*
  Input: pth
  1. *Parse* $\mathsf{pth} := [(\mathsf{h}^0_0, \mathsf{h}^1_0), (\mathsf{h}^0_1, \mathsf{h}^1_1, b_1) \dots, (\mathsf{h}^0_{\mathsf{d}_i-1}, \mathsf{h}^1_{\mathsf{d}_i-1}, b_{\mathsf{d}_i-1}), \mathsf{rt}]$.
  2. *If* $\mathsf{rt}_i \neq \mathsf{rt}$*, then output* $\perp$*.*
  3. *If* $\mathsf{id} \neq \mathsf{h}^0_0$*, then output* $\perp$*.*

[9] Keeping this list is not necessary, but simplifies the presentation of the updates.

4. *If* $\mathsf{rt} = \mathrm{Hash}(\mathsf{hk}_{\mathsf{d}_i}, \mathsf{h}^0_{\mathsf{d}_i-1}||\mathsf{h}^1_{\mathsf{d}_i-1})$ *and* $\mathsf{h}^{b_j}_j = \mathrm{Hash}(\mathsf{hk}_j, \mathsf{h}^0_{j-1}||\mathsf{h}^1_{j-1})$ *for* all $j \in [\mathsf{d}_i - 1]$, *then output* $\mathrm{E}(\mathsf{h}^1_0, \mathsf{m}; \mathsf{r})$ *by using* h^1_0 *as the public key and* r *as the randomness, otherwise output* $\perp$.

Then, output $\mathsf{ct} := (\mathsf{pp}, \mathrm{Obf}(\mathrm{P}_1), \ldots, \mathrm{Obf}(\mathrm{P}_\eta))$ *where* Obf *is* IO *obfuscation.*

– $\mathrm{Upd}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{id}) \to \mathsf{u}$*: Letting* $\mathsf{aux} := (\mathsf{Tree}_1, \ldots, \mathsf{Tree}_\zeta)$ *and letting* i *be the index of the tree that holds* id*, return the whole Merkle opening of the path that leads to* id *in* Tree_i*.*
– $\mathrm{Dec}(\mathsf{sk}, \mathsf{u}, \mathsf{ct}) \to \mathsf{m}$*: Parse* $\mathsf{ct} = (\mathsf{pp}, \overline{\mathrm{P}}_1, \ldots, \overline{\mathrm{P}}_\eta)$*. Form* $\mathsf{m}_i = \mathrm{Dec}_{\mathsf{sk}}(\overline{\mathrm{P}}_i(\mathsf{u}))$ *for each program* $\overline{\mathrm{P}}_i$*. Output the first* $\mathsf{m}_i \neq \perp$*.*

Theorem 13. *The RBE of Construction 12 satisfies the compactness, completeness properties according to Definition 6 and security according to Definition 10.*

In the rest of this section, we prove Theorem 13. Along the way, we describe the modifications that are needed to Construction 12 to make it efficient according to Definition 8.

4.1 Proofs of Completeness, Compactness and Efficiency

Completeness is straightforward. Below we sketch why compactness holds.

Compactness of public parameters and updates. The public parameter's format is of the form $\mathsf{pp} = ((\mathsf{hk}_1, \ldots \mathsf{hk}_\kappa), (\mathsf{rt}_1, \mathsf{d}_1), \ldots (\mathsf{rt}_\eta, \mathsf{d}_\eta))$ where $\mathsf{rt}_i \in \{0,1\}^\kappa$. Also, the identities are of length κ, so the depth of each tree is at most κ bits. It only remains to show that the *number* of trees at any moment is at most $\log(n)$. This is because the trees are *full* binary trees (of size 2^{d_i}) and the size of the trees are always different (otherwise, the registration step keeps merging them). Therefore, $\eta \leq \log(n)$, and so the length of the pp_n will be at most $O(\kappa^2 + \kappa \cdot \log(n))$. In fact, we can optimize this length to be at most $O(\kappa \cdot \log(n))$ by only generating the hash keys when needed (i.e., when the registered population reaches 2^k, we will generate hk_k and put it in the public parameter). Compactness of updates is trivial.

Efficiency of runtime of registration and update. The efficiency of registration follows from the fact that the total number of merges is at most $\log n$. The efficiency of update runtime can also be easily guaranteed by using an appropriate data structure that maps a given identity to the leafs containing it in each tree (e.g., we can use a Trie data structure for this purpose to get such list in minimal time over the input length).

All other measures of efficiency either follows trivially, or by the $\log(n)$ upperbound on the number of merges.

4.2 Proof of Security

We now prove the security of Construction 12. We start by giving intuition about the security proof for a simple case. We will then give a detailed proof for the general case.

Simple case of one user. Consider the case in which only one user has registered, and that the adversary wants to distinguish between encryptions of $\mathsf{m} \in \{0,1\}$ made to that user. Let id^* be the identity of the user who has registered, and let $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathrm{G}(1^\kappa)$ be the pair of public/secret keys that the challenger Chal produced at the time of registration as per Definition 10. Since we have only one user, the public parameter is $\mathsf{pp} := \mathrm{Hash}(\mathsf{hk}, \mathsf{id}^*||\mathsf{pk}^*)$, where $\mathsf{hk} \leftarrow \mathrm{HGen}(1^\kappa, 0)$. Recall that w.l.o.g., we have $|\mathsf{id}^*| = |\mathsf{pk}^*| = |\mathsf{pp}| = \kappa$.

An encryption of a bit $\mathsf{m} \in \{0,1\}$ to identity id^* is an IO obfuscation of the circuit P in Figure 1.

Hardwired: $\mathsf{m} \in \{0,1\}$, id^*, pp, hk and randomness r

Input: $(\mathsf{id}, \mathsf{pk})$

1. If $\mathrm{Hash}(\mathsf{hk}, \mathsf{id}||\mathsf{pk}) \neq \mathsf{pp}$, then output $\perp$ and end.
2. If $\mathsf{id} \neq \mathsf{id}^*$, then output $\perp$ and end.
3. Output $\mathrm{E}(\mathsf{pk}, \mathsf{m}; \mathsf{r})$ and end.

Fig. 1: Circuit P used for encryption of m to identity id^*

Theorem 14 (Security). *For any* id^* *we have*

$$\mathrm{Obf}(\mathrm{P}[0, \mathsf{id}^*, \mathsf{pp}, \mathsf{hk}, \mathsf{r}]) \overset{c}{\approx} \mathrm{Obf}(\mathrm{P}[1, \mathsf{id}^*, \mathsf{pp}, \mathsf{hk}, \mathsf{r}]), \qquad (1)$$

for $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathrm{G}(1^\kappa)$, $\mathsf{hk} \leftarrow \mathrm{HGen}(1^\kappa, 0)$, $\mathsf{pp} := \mathrm{Hash}(\mathsf{hk}, \mathsf{id}^*||\mathsf{pk}^*)$, $\mathsf{r} \leftarrow \{0,1\}^*$.

Roadmap for the proof of Theorem 14. We first alter the circuit P to obtain a circuit P_1, which works similarly except that P_1 checks whether or not its given input path is exactly $(\mathsf{id}^*, \mathsf{pk}^*)$ (i.e., the already registered identity along with its public key); if not, P_1 will return $\perp$, even if the two leaves $(\mathsf{id}, \mathsf{pk})$ correctly hashe to pp. If yes, P_1 will encrypt the hardwired bit m under the public key pk^* and the hardwired randomness r. The circuit P_1 is defined in Figure 2.

Equipped with this new circuit P_1, first in Lemma 15 we show that under P_1 we may switch the underlying hardwired plaintext bit m from 0 to 1 while keeping the obfuscations of the resulting circuits indistinguishable. Then, in Lemma 16 we will show that for any fixed plaintext bit m, the obfuscations of P and P_1 are computationally indistinguishable. These two lemmas imply Theorem 14.

We start by defining the circuit P_1, which is a modified version of P.

Hardwired: $\mathsf{m} \in \{0,1\}$, id^*, pk^*, pp, hk and randomness r

Input: $(\mathsf{id}, \mathsf{pk})$

1. If $(\mathsf{id}, \mathsf{pk}) \neq (\mathsf{id}^*, \mathsf{pk}^*)$, then output $\perp$ and end.
2. Output $\mathrm{E}(\mathsf{pk}, \mathsf{m}; \mathsf{r})$ and end.

Fig. 2: Circuit P_1

We now formally show that under P_1 we may switch the underlying plaintext bit while keeping their obfuscations indistinguishable.

Lemma 15. *For any* id^* *and* hk *we have*

$$\mathrm{Obf}(\mathrm{P}_1[0, \mathsf{id}^*, \mathsf{pk}^*, \mathsf{pp}, \mathsf{hk}, \mathsf{r}]) \overset{c}{\approx} \mathrm{Obf}(\mathrm{P}_1[1, \mathsf{id}^*, \mathsf{pk}^*, \mathsf{pp}, \mathsf{hk}, \mathsf{r}]), \tag{2}$$

where $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathrm{G}(1^\kappa)$*,* $\mathsf{r} \leftarrow \{0,1\}^*$ *and* $\mathsf{pp} := \mathrm{Hash}(\mathsf{hk}, \mathsf{id}^*||\mathsf{pk}^*)$*.*

Proof. Fix id^* and hk. We slightly change the circuit P_1 into a circuit P_2, so that the circuit P_2, instead of getting m, pk^* and r hardwired into itself, it gets the resulting ciphertext c^* hardwired, and it will return this ciphertext if the check inside the program holds. This new circuit P_2 is defined in Figure 3.

Notice that for all fixed $\mathsf{m} \in \{0,1\}$, id^*, pk^*, r and $\mathsf{pp} := \mathrm{Hash}(\mathsf{hk}, \mathsf{id}^*||\mathsf{pk}^*)$,

$$\mathrm{Obf}(\mathrm{P}_1[\mathsf{m}, \mathsf{id}^*, \mathsf{pk}^*, \mathsf{pp}, \mathsf{hk}, \mathsf{r}]) \overset{c}{\approx} \mathrm{Obf}(\mathrm{P}_2[\mathsf{id}^*, \mathsf{pp}, \mathsf{hk}, \mathsf{c}^*]), \tag{3}$$

where $\mathsf{c}^* := \mathrm{E}(\mathsf{pk}^*, \mathsf{m}; \mathsf{r})$. The reason behind Equation 3 is that the underlying two circuits are functionally equivalent, and so their obfuscations must be computationally indistinguishable by the property of IO.

We now show that under P_2 we may switch the hardwired ciphertext from an encryption of zero to one, by relying on semantic security of the PKE. Formally,

$$\mathrm{Obf}(\mathrm{P}_2[\mathsf{id}^*, \mathsf{pp}, \mathsf{hk}, \mathsf{c}_0^*]) \overset{c}{\approx} \mathrm{Obf}(\mathrm{P}_2[\mathsf{id}^*, \mathsf{pp}, \mathsf{hk}, \mathsf{c}_1^*]), \tag{4}$$

for $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathrm{G}(1^\kappa)$, $\mathsf{c}_0^* \leftarrow \mathrm{E}(\mathsf{pk}^*, 0)$, $\mathsf{c}_1^* \leftarrow \mathrm{E}(\mathsf{pk}^*, 1)$, $\mathsf{pp} := \mathrm{Hash}(\mathsf{hk}, \mathsf{id}^*||\mathsf{pk}^*)$. Equation 4 directly follows from the semantic security of the underlying public-key encryption scheme. Finally, note that Equations 4 and 3 imply Equation 2 of the lemma, and so we are done. □

Hardwired: id^*, pp, hk and c^*

Input: $(\mathsf{id}, \mathsf{pk})$

1. If $(\mathsf{id}, \mathsf{pk}) \neq (\mathsf{id}^*, \mathsf{pk}^*)$, then output $\perp$ and end.
2. Output c^* and end.

Fig. 3: Circuit P_2

We now show that for any fixed plaintext $\mathsf{m} \in \{0,1\}$, the obfuscations of the two circuits P and P_1 are computationally indistinguishable.

Lemma 16. *For fixed* $\mathsf{m} \in \{0,1\}, \mathsf{id}^* \in \{0,1\}^\kappa, \mathsf{pk}^* \in \{0,1\}^\kappa$ *and randomness* r*, it holds that*

$$\mathrm{Obf}(\mathrm{P}[\mathsf{m}, \mathsf{id}^*, \mathsf{pp}, \mathsf{hk}, \mathsf{r}]) \overset{c}{\approx} \mathrm{Obf}(\mathrm{P}_1[\mathsf{m}, \mathsf{id}^*, \mathsf{pk}^*, \mathsf{pp}, \mathsf{hk}, \mathsf{r}]), \tag{5}$$

where $\mathsf{hk} \leftarrow \mathrm{HGen}(1^\kappa, 0)$ *and* $\mathsf{pp} := \mathrm{Hash}(\mathsf{hk}, \mathsf{id}^* || \mathsf{pk}^*)$*.*

Proof. Let a hash key hk_1 be sampled as follows: $\mathsf{hk}_1 \leftarrow \mathrm{HGen}(1^\kappa, 1)$. We show that Equation 5 will hold if hk is replaced with hk_1. This will complete our proof because by the index hiding property of (HGen, Hash) we know $\mathsf{hk} \overset{c}{\approx} \mathsf{hk}_1$. Thus, it only remains to prove

$$\mathrm{Obf}(\mathrm{P}[\mathsf{m}, \mathsf{id}^*, \mathsf{pk}^*, \mathsf{pp}, \mathsf{hk}_1, \mathsf{r}]) \overset{c}{\approx} \mathrm{Obf}(\mathrm{P}_1[\mathsf{m}, \mathsf{id}^*, \mathsf{pk}^*, \mathsf{pp}, \mathsf{hk}_1, \mathsf{r}]), \tag{6}$$

where $\mathsf{hk}_1 \leftarrow \mathrm{HGen}(1^\kappa, 1)$ and $\mathsf{pp} := \mathrm{Hash}(\mathsf{hk}_1, \mathsf{id}^* || \mathsf{pk}^*)$. To prove Equation 6 we claim that the underlying two circuits are functionally equivalent; namely,

$$\mathrm{P}[\mathsf{m}, \mathsf{id}^*, \mathsf{pk}^*, \mathsf{pp}, \mathsf{hk}_1, \mathsf{r}] \equiv \mathrm{P}_1[\mathsf{m}, \mathsf{id}^*, \mathsf{pk}^*, \mathsf{pp}, \mathsf{hk}_1, \mathsf{r}]. \tag{7}$$

Note that by security definition of IO, Equation 7 implies Equation 6, and thus we just need to prove Equation 7. To prove equivalence of the circuits, assume to the contrary that there exists an input $(\mathsf{id}, \mathsf{pk})$ for which we have $\mathrm{P}(\mathsf{id}, \mathsf{pk}) \neq \mathrm{P}_1(\mathsf{id}, \mathsf{pk})$. (Here for better readability we dropped the hardwired values.) By simple inspection, we can see that we have $\mathrm{P}(\mathsf{id}, \mathsf{pk}) \neq \mathrm{P}_1(\mathsf{id}, \mathsf{pk})$ iff all the following conditions hold:

1. $\mathrm{Hash}(\mathsf{hk}_1, (\mathsf{id}, \mathsf{pk})) = \mathsf{pp}$; and
2. $\mathsf{id} = \mathsf{id}^*$; and
3. $\mathsf{pk} \neq \mathsf{pk}^*$.

This, however, is a contradiction because by the somewhere statistical binding property of (HGen, Hash) and by the fact that $\mathsf{hk}_1 \leftarrow \mathrm{HGen}(1^\kappa, 1)$, Conditions 1 and 2 imply $\mathsf{pk} = \mathsf{pk}^*$, a contradiction to Condition 3. □

General case of multiple users. We will prove our security for the case in which at the time of encryption, we only have one tree (of any arbitrary depth). This is without loss of generality for the following reason. Recall that for encryption, if we have m roots, we obfuscate a circuit individually for each root. Suppose at the time of encryption, we have m trees with respective roots $\mathsf{rt}_1, \ldots, \mathsf{rt}_m$. Then, between the two main hybrids which correspond to an encryption of zero and an encryption of one, we may consider m intermediate hybrids, where under the ith hybrid we encrypt 0 under the roots $\{\mathsf{rt}_1, \ldots, \mathsf{rt}_i\}$ and we encrypt 1 under the roots $\{\mathsf{rt}_{i+1}, \ldots, \mathsf{rt}_m\}$. Thus, using a hybrid argument, the result will follow.

Roadmap of the security proof. We will define four hybrids, where the first hybrid corresponds to an encryption of bit 0 and the last hybrid corresponds to an encryption of bit 1. We will prove that the views of the adversary in each of the two adjacent hybrids are computationally indistinguishable.

High-level proof sketch. Let Tree be the underlying tree at the time of encryption. An encryption of a bit m to an identity id corresponds to an IO obfuscation of a circuit P, which takes as input a path, and which will release an encryption of m under a public key given as a leaf of the path, if the given path is "valid." As a hybrid, we will consider a circuit P_1, which does all the checks that are already performed by P, but which also does the following: if the given path is not *present* in the tree, then P_1 will return $\perp$, even if the path is valid. We will show that for any fixed bit m, if we encrypt m by obfuscating either the circuit P or P_1, the result will be indistinguishable. We will make use of the somewhere statistical binding and index hiding of the underlying hash function in order to prove this. Now under an obfuscation of P_1, one may easily switch the hardwired plaintext bit. The reason is that since under P_1, a given input path to the circuit must be present in the tree, and since the challenge identity id^* is registered only once (say under a public key pk), one may consider a related circuit which, instead of hardwiring a plaintext bit m, it hardwires into itself an encryption $c \leftarrow \mathrm{E}(\mathsf{pk}, \mathsf{m})$. The rest follows by semantic security of the PKE scheme.

We now go over the formal proof. We start by defining some notation.

Notation. Consider a path $\mathsf{pth} := [(\mathsf{id}, \mathsf{pk}), (\mathsf{h}_1^0, \mathsf{h}_1^1, b_1), \ldots, (\mathsf{h}_{t-1}^0, \mathsf{h}_{t-1}^1, b_{t-1}), \mathsf{rt}]$ where rt is the root and id and pk are the two leaves and $b_1, \ldots, b_{t-1} \in \{\mathsf{left}, \mathsf{right}\}$. For a tree Tree of depth t, we write $\mathsf{pth} \subseteq \mathsf{Tree}$ if pth is a valid path in Tree in the usual sense. The procedure $\mathrm{Valid}(\mathsf{hk}_1, \ldots, \mathsf{hk}_t, \mathsf{pth})$ checks if the given path is a 'valid path' according to the given hash keys $\mathsf{hk}_1, \ldots, \mathsf{hk}_t$ then it output $\top$, otherwise outputs $\perp$. For a path pth and interger i we write $\mathrm{Last}(\mathsf{pth}, i)$ to refer to the last i node "elements" in pth. Note that we do not consider the left-or-right bits as part of this counting. For example, letting pth be as above,

$$\mathrm{Last}(\mathsf{pth}, 5) = ((\mathsf{h}_{t-2}^0, \mathsf{h}_{t-1}^1, b_{t-2}), (\mathsf{h}_{t-1}^0, \mathsf{h}_{t-1}^1, b_{t-1}), \mathsf{rt}).$$

We also extend the notation $\subseteq$ given above to define $\mathrm{Last}(\mathsf{pth}, i) \subseteq \mathsf{Tree}$ in the straightforward way.

Hardwired: $\mathsf{m} \in \{0, 1\}$, id^*, rt, $\mathsf{hk}_1, \ldots, \mathsf{hk}_t$ and randomness r

Input: $\mathsf{pth} := [(\mathsf{id}, \mathsf{pk}), (\mathsf{h}_1^0, \mathsf{h}_1^1, b_1), \ldots, (\mathsf{h}_{t-1}^0, \mathsf{h}_{t-1}^1, b_t), \mathsf{rt}']$

1. If $\mathsf{id} \neq \mathsf{id}^*$, $\mathsf{rt} \neq \mathsf{rt}'$ or $\mathrm{Valid}(\mathsf{hk}_1, \ldots, \mathsf{hk}_t, \mathsf{pth}) \neq \top$, then output $\perp$ and end.
2. Output $\mathrm{E}(\mathsf{pk}, \mathsf{m}; \mathsf{r})$.

Fig. 4: Circuit P

Circuit P_1

Hardwired: $\mathsf{m} \in \{0,1\}$, id^*, $\mathsf{pth}^*, \mathsf{rt}$, $\mathsf{hk}_1, \ldots, \mathsf{hk}_t$ and randomness r

Input: $\mathsf{pth} := [(\mathsf{id}, \mathsf{pk}), (\mathsf{h}_1^0, \mathsf{h}_1^1, b_1), \ldots, (\mathsf{h}_{t-1}^0, \mathsf{h}_{t-1}^1, b_t), \mathsf{rt}']$

1. If $\mathsf{pth} = \mathsf{pth}^*$, then output $\mathrm{E}(\mathsf{pk}, \mathsf{m}; \mathsf{r})$ and end.
2. Else, output $\perp$ and end.

Fig. 5: Circuit P_1

Notation used in hybrids. We will write $\mathsf{id}^* \leftarrow \mathsf{Adv}(\mathsf{hk}_1, \ldots, \mathsf{hk}_\kappa)$ to mean that the adversary Adv receives $\mathsf{pp} := (\mathsf{hk}_1, \ldots, \mathsf{hk}_\kappa)$ as input, interacts with the challenger Chal as per Definition 10 and outputs id^* as the challenge identity.

– **Hybrid $\mathbf{H}_1$: Encrypt $\mathsf{m} = 0$ using** P**.** The ciphertext ct given to the adversary is formed as follows.
  1. For $j \in [\kappa]$ sample $\mathsf{hk}_j \leftarrow \mathrm{HGen}(1^\kappa, 0)$.
  2. $\mathsf{id}^* \leftarrow \mathsf{Adv}(\mathsf{hk}_1, \ldots, \mathsf{hk}_\kappa)$.
  3. $\mathsf{ct} \leftarrow \mathrm{Obf}(\mathrm{P}[0, \mathsf{id}^*, \mathsf{rt}, \mathsf{hk}_1, \ldots, \mathsf{hk}_t, r])$, where rt is the root of the tree, t is the depth of the tree, and $\mathsf{r} \leftarrow \{0,1\}^*$.

– **Hybrid $\mathbf{H}_2$: Encrypt $\mathsf{m} = 0$ using** P_1**.** The ciphertext ct given to the adversary is formed as follows.
  1. For $j \in [\kappa]$ sample $\mathsf{hk}_j \leftarrow \mathrm{HGen}(1^\kappa, 0)$.
  2. $\mathsf{id}^* \leftarrow \mathsf{Adv}^{\mathrm{Reg}_{\mathsf{sel}}, \mathrm{Reg}_{\mathsf{smp}}}(\mathsf{hk}_1, \ldots, \mathsf{hk}_\kappa)$.
  3. $\mathsf{ct} \leftarrow \mathrm{Obf}(\mathrm{P}_1[0, \mathsf{id}^*, \mathsf{pth}^*, \mathsf{rt}, \mathsf{hk}_1, \ldots, \mathsf{hk}_t, r])$, where pth^* is the path in the tree leading to the challenge node, rt is the root of pth^*, t is the depth of the tree, and $\mathsf{r} \leftarrow \{0,1\}^*$.

– **Hybrid $\mathbf{H}_3$: Encrypt $\mathsf{m} = 1$ using** P_1**.** The ciphertext ct given to the adversary is formed as follows.
  1. For $j \in [\kappa]$ sample $\mathsf{hk}_j \leftarrow \mathrm{HGen}(1^\kappa, 0)$.
  2. $\mathsf{id}^* \leftarrow \mathsf{Adv}(\mathsf{hk}_1, \ldots, \mathsf{hk}_\kappa)$.
  3. $\mathsf{ct} \leftarrow \mathrm{Obf}(\mathrm{P}_1[1, \mathsf{id}^*, \mathsf{pth}^*, \mathsf{rt}, \mathsf{hk}_1, \ldots, \mathsf{hk}_t, r])$, where pth^* is the path in the tree leading to the challenge node, rt is the root of pth^*, t is the depth of the tree, and $\mathsf{r} \leftarrow \{0,1\}^*$.

– **Hybrid $\mathbf{H}_4$: Encrypt $\mathsf{m} = 1$ using** P**.** The ciphertext ct given to the adversary is formed as follows.
  1. For $j \in [\kappa]$ sample $\mathsf{hk}_j \leftarrow \mathrm{HGen}(1^\kappa, 0)$.
  2. $\mathsf{id}^* \leftarrow \mathsf{Adv}(\mathsf{hk}_1, \ldots, \mathsf{hk}_\kappa)$.
  3. $\mathsf{ct} \leftarrow \mathrm{Obf}(\mathrm{P}[1, \mathsf{id}^*, \mathsf{rt}, \mathsf{hk}_1, \ldots, \mathsf{hk}_t, r])$, where rt is the root of the underlying tree, t is the depth of the tree, and $\mathsf{r} \leftarrow \{0,1\}^*$.

Notation. We use $\mathsf{ct}\langle \mathrm{H}_i \rangle$ to denote the value of the ciphertext ct in Hybrid H_i.

Lemma 17. *We have,*

1. $\mathsf{ct}\langle \mathrm{H}_1 \rangle \overset{c}{\approx} \mathsf{ct}\langle \mathrm{H}_2 \rangle$,
2. $\mathsf{ct}\langle \mathrm{H}_3 \rangle \overset{c}{\approx} \mathsf{ct}\langle \mathrm{H}_4 \rangle$.

Proof. We will prove Part 1, and the proof for Part 2 will be exactly the same.

Recall that in hybrid H_1 we encrypt $\mathsf{m} = 0$ by obfuscating P and that in hybrid H_2 we encrypt $\mathsf{m} = 0$ by obfuscating P_1. Let t be the depth of the tree at the time of encryption.

We will define intermediate hybrids $\mathrm{P}_{2,i}$ for $i \in [2t+1]$, and we will show $\mathrm{P} \equiv \mathrm{P}_{2,1}$, $\mathrm{P}_1 \equiv \mathrm{P}_{2,2t+1}$ and for all $i \in [2t]$, $\mathrm{Obf}[\mathrm{P}_{2,i}] \overset{c}{\approx} \mathrm{Obf}[\mathrm{P}_{2,i+1}]$. These circuit programs are given in Figure 6.

Informally, the program $\mathrm{P}_{2,i}$ works as follows: it checks whether its given path is "correct" and whether, in addition, the last i elements of the path are in accordance with the challenge path pth^* that was hardwired into the program. For example, if $i = 5$, then the root of the path and the two levels below it (five nodes in total) should match the corresponding nodes in the challenge path pth^*. If both these conditions hold, then $\mathrm{P}_{2,i}$ will encrypt the hardwired plaintext bit ($\mathsf{m} = 0$) using the public key provided in the corresponding leave of the path.

We will now define a Hybrid $\mathrm{H}_{2,i}$ below, which uses program $\mathrm{P}_{2,i}$.

– **Hybrid $\mathbf{H}_{2,i}$: Encrypt $\mathsf{m} = 0$ using $\mathrm{P}_{2,i}$.** The given ciphertext ct is as:
  1. For $j \in [\kappa]$ sample $\mathsf{hk}_j \leftarrow \mathrm{HGen}(1^\kappa, 0)$.
  2. $\mathsf{id}^* \leftarrow \mathsf{Adv}(\mathsf{hk}_1, \ldots, \mathsf{hk}_\kappa)$.
  3. $\mathsf{ct} \leftarrow \mathrm{Obf}(\mathrm{P}_{2,i}[0, \mathsf{id}^*, \mathsf{pth}^*, \mathsf{rt}, \mathsf{hk}_1, \ldots, \mathsf{hk}_t, r])$, where pth^* is the challnege path in the system, rt is the root of pth^*, t is the depth of the tree, and $r \leftarrow \{0,1\}^*$.

First, by inspection we can see that $\mathsf{ct}\langle \mathrm{H}_1 \rangle \overset{c}{\approx} \mathsf{ct}\langle \mathrm{H}_{2,1} \rangle$ and $\mathsf{ct}\langle \mathrm{H}_2 \rangle \overset{c}{\approx} \mathsf{ct}\langle \mathrm{H}_{2,2t+1} \rangle$. This is because the underlying two circuits P and $\mathrm{P}_{2,1}$ are functionally equivalent. Same holds for P_1 and $\mathrm{P}_{2,2t+1}$.

Thus, for any fixed $w \in [2t]$ we just need to prove

$$\mathsf{ct}\langle \mathrm{H}_{2,w} \rangle = \mathsf{ct}\langle \mathrm{H}_{2,w+1} \rangle. \tag{8}$$

Below, we fix $w \in [2t]$. To prove Equation 8, we introduce two hybrids $\mathrm{H}'_{2,w}, \mathrm{H}'_{2,w+1}$ and show

$$\mathsf{ct}\langle \mathrm{H}_{2,w} \rangle \overset{c}{\approx} \mathsf{ct}\langle \mathrm{H}'_{2,w} \rangle \overset{c}{\approx} \mathsf{ct}\langle \mathrm{H}'_{2,w+1} \rangle \overset{c}{\approx} \mathsf{ct}\langle \mathrm{H}_{2,w+1} \rangle. \tag{9}$$

This will establish Equation 8.

Informally, the hybrids $\mathrm{H}'_{2,w}$ and $\mathrm{H}'_{2,w+1}$ are defined similarly to $\mathrm{H}_{2,w}$ and $\mathrm{H}_{2,w+1}$, except that one of the many hash keys is now sampled in a different way, in order to make some binding property happen.

For $z \in \{w, w+1\}$, the hybrid $\mathrm{H}'_{2,z}$ is defined as follows.

– **Hybrid $\mathbf{H}'_{2,z}$ for $z \in \{w, w+1\}$.** The given ciphertext ct is formed as follows.

1. Let $q := t - \lfloor \frac{w}{2} \rfloor - 1$ Intuitively, q denotes the level index in the tree for which we want to use a different hash key. For all $i \in [\kappa] \setminus \{q\}$: sample $\mathsf{hk}'_i \leftarrow \mathrm{HGen}(1^\kappa, 0)$. Sample
$$\mathsf{hk}'_q \leftarrow \mathrm{HGen}(1^\kappa, v), \text{ where } v := (w+1) \mod 2.$$
2. $\mathsf{id}^*_1 \leftarrow \mathsf{Adv}(\mathsf{hk}'_1, \ldots, \mathsf{hk}'_\kappa)$.
3. $\mathsf{ct} \leftarrow \mathrm{Obf}(\mathrm{P}_{2,i}[0, \mathsf{id}^*_1, \mathsf{pth}^*_1, \mathsf{rt}_1, \mathsf{hk}'_1, \ldots, \mathsf{hk}'_t, r])$, where pth^*_1 is the challnege path in the system, rt_1 is the root of pth^* and $\mathsf{r} \leftarrow \{0,1\}^*$.

Toward proving Equation 9, first note that by the index hiding property of (HGen, Hash) we have $\mathsf{ct}\langle \mathrm{H}_{2,w} \rangle \overset{c}{\approx} \mathsf{ct}\langle \mathrm{H}'_{2,w} \rangle$ and $\mathsf{ct}\langle \mathrm{H}_{2,w+1} \rangle \overset{c}{\approx} \mathsf{ct}\langle \mathrm{H}'_{2,w+1} \rangle$. Thus, it remains to prove

$$\mathsf{ct}\langle \mathrm{H}'_{2,w} \rangle \overset{c}{\approx} \mathsf{ct}\langle \mathrm{H}'_{2,w+1} \rangle. \tag{10}$$

To prove Equation 10, we claim that the underlying two programs are equivalent,

$$\mathrm{P}_{2,w}[0, \mathsf{id}^*_1, \mathsf{pth}^*_1, \mathsf{rt}_1, \mathsf{hk}'_1, .., \mathsf{hk}'_t, r] = \mathrm{P}_{2,w+1}[0, \mathsf{id}^*_1, \mathsf{pth}^*_1, \mathsf{rt}_1, \mathsf{hk}'_1, .., \mathsf{hk}'_t, r]. \tag{11}$$

By IO security, Equation 11 implies Equation 10, and thus we just need to prove Equation 11. To prove equivalence of the two circuits in Equation 11, assume to the contrary that there exists an input pth for which we have $\mathrm{P}_{2,w}(\mathsf{pth}) \neq \mathrm{P}_{2,w+1}(\mathsf{pth})$. (Here for better readability we dropped the hardwired values.) By simple inspection we can see that we have $\mathrm{P}_{2,w}(\mathsf{pth}) \neq \mathrm{P}_{2,w+1}(\mathsf{pth})$ iff all the following conditions hold:

1. $\mathrm{Valid}(\mathsf{hk}'_1, \ldots, \mathsf{hk}'_t, \mathsf{pth}) = \top$; and
2. $\mathrm{Last}(\mathsf{pth}, w) \subseteq \mathsf{pth}^*_1$; and
3. $\mathrm{Last}(\mathsf{pth}, w+1) \not\subseteq \mathsf{pth}^*_1$.

This, however, is a contradiction because by the somewhere statistical binding property of (KGen, Hash) and by the way in which we have sampled hk'_q, Conditions 1 and 2 contradict Condition 3.

□

Description of Circuit $\mathrm{P}_{2,i}$.

Hardwired: $\mathsf{m} \in \{0,1\}$, id^*, $\mathsf{pth}^*, \mathsf{rt}, \mathsf{hk}_1, \ldots, \mathsf{hk}_t$ and randomness r

Input: $\mathsf{pth} := [(\mathsf{id}, \mathsf{pk}), (\mathsf{h}^0_1, \mathsf{h}^1_1, b_1), \ldots, (\mathsf{h}^0_{t-1}, \mathsf{h}^1_{t-1}, b_t), \mathsf{rt}']$

1. If $\mathsf{id} \neq \mathsf{id}^*$ or $\mathsf{rt} \neq \mathsf{rt}'$ or $\mathrm{Valid}(\mathsf{hk}_1, \ldots, \mathsf{hk}_t, \mathsf{pth}) \neq \top$, then output $\perp$ and end.
2. If $\mathrm{Last}(\mathsf{pth}, i) \subseteq \mathsf{pth}^*$, then output $\mathrm{E}(\mathsf{pk}, \mathsf{m}; \mathsf{r})$ and end.
3. Otherwise, output $\perp$ and end.

Fig. 6: Circuit $\mathrm{P}_{2,i}$ for $i \in [\ell]$

Lemma 18. $\mathsf{ct}\langle \mathrm{H}_2\rangle \overset{c}{\approx} \mathsf{ct}\langle \mathrm{H}_3\rangle$.

Proof. The proof is similar to the proof of Lemma 15. □

5 Basing Weakly-Efficient RBE on Standard Assumptions

In this section, we describe our construction of RBE based on *hash garbling* and is inspired by our IO based construction from previous section. This notion and its construction has been implicit in prior works [7,11], and it was shown [11,12,4] that hash garbling can be realized based on CDH, Factoring or LWE assumptions. Specifically, implicit in these prior works are constructions of hash garbling based on hash encryption and garbled circuits. Below, we abstract out this notion and use it in our work directly. This abstract primitive significantly simplifies exposition.

Definition 19 (Hash garbling). *A* hash garbling *scheme consists of four PPT algorithms* HGen, Hash, HG, *and* HInp, *defined as follows.*

- $\mathrm{HGen}(1^\kappa, 1^\ell) \to \mathsf{hk}$. *This algorithm takes the security parameter κ and an output length parameter 1^ℓ for $\ell \leq \mathrm{poly}(\kappa)$, and outputs a hash key* hk. *(*HGen *runs in* $\mathrm{poly}(\kappa)$ *time.)*
- $\mathrm{Hash}(\mathsf{hk}, x) = y$. *This takes* hk *and $x \in \{0,1\}^\ell$ and outputs $y \in \{0,1\}^\kappa$.*
- $\mathrm{HG}(\mathsf{hk}, \mathrm{C}, \mathsf{stt}) \to \widetilde{\mathrm{C}}$. *This algorithm takes a hash key* hk*, a circuit* C*, and a secret state* $\mathsf{stt} \in \{0,1\}^\kappa$ *as input and outputs a circuit* $\widetilde{\mathrm{C}}$.
- $\mathrm{HInp}(\mathsf{hk}, y, \mathsf{stt}) \to \widetilde{y}$. *This algorithm takes a hash key* hk*, a value $y \in \{0,1\}^\kappa$, and a secret state* stt *as input and outputs* $\widetilde{y}$.

We require the following properties for a hash garbling scheme:

- **Correctness.** *For all κ, ℓ,* $\mathsf{hk} \leftarrow \mathrm{HGen}(1^\kappa, 1^\ell)$*, circuit* C*, input $x \in \{0,1\}^\ell$,* $\mathsf{stt} \in \{0,1\}^\kappa$*,* $\widetilde{\mathrm{C}} \leftarrow \mathrm{HG}(\mathsf{hk}, \mathrm{C}, \mathsf{stt})$ *and* $\widetilde{y} \leftarrow \mathrm{HInp}(\mathsf{hk}, \mathrm{Hash}(\mathsf{hk}, x), \mathsf{stt})$*, then* $\widetilde{\mathrm{C}}(\widetilde{y}, x) = \mathrm{C}(x)$.
- **Security.** *There exists a PPT simulator* Sim *such that for all κ, ℓ (recall that ℓ is polynomial in κ) and PPT (in κ) $\mathcal{A}$ we have that*
$$(\mathsf{hk}, x, \widetilde{C}, \widetilde{y}) \overset{c}{\approx} (\mathsf{hk}, x, \mathsf{Sim}(\mathsf{hk}, x, 1^{|\mathrm{C}|}, \mathrm{C}(x))), \text{ where}$$
$\mathsf{hk} \leftarrow \mathrm{HGen}(1^\kappa, 1^\ell)$, $(\mathrm{C}, x) \leftarrow \mathcal{A}(\mathsf{hk})$, $\mathsf{stt} \leftarrow \{0,1\}^\kappa$, $\widetilde{\mathrm{C}} \leftarrow \mathrm{HG}(\mathsf{hk}, \mathrm{C}, \mathsf{stt})$ *and* $\widetilde{y} \leftarrow \mathrm{HInp}(\mathsf{hk}, \mathrm{Hash}(\mathsf{hk}, x), \mathsf{stt})$.

Notation on binary trees. Just like the IO construction, in our construction below, Tree is a *full* binary tree where the label of each node in Tree is calculated as the hash of its left and right children and, now additionally, with an an extra identity. Looking ahead, this identity will be the largest identity among the users registered in the left child. (Such information is useful if one wants to a binary search of an identity over this tree.) Just as in the IO-based construction, we define the size of a tree Tree as the number of its *leaves*, denoted by size(Tree), and we denote the root of Tree as $\mathsf{rt}(\mathsf{Tree})$, and use $\mathsf{d}(\mathsf{Tree})$ to refer to the depth of Tree. Again, when Tree is clear from the context, we use rt and d to denote the root and the depth of Tree.

Before describing the construction, recall that without loss of generality, we can assume that public keys, secret keys, and identities, are all of length security parameter κ.

Comparison with Construction 12 using signs *(=)* and *(⋆⋆)*. To help the reader familiar with Construction 12, we have denoted the steps that are identical to Construction 12 by *(=)* and the steps that are significantly *different* by *(⋆⋆)*. Other steps are close but not identical.

Construction 20 (Construction of RBE from hash garbling) *We will use a hash garbling scheme* $(\mathrm{HGen}, \mathrm{Hash}, \mathrm{HG}, \mathrm{HInp})$ *and a public key encryption scheme* $(\mathrm{G}, \mathrm{E}, \mathrm{D})$. *Using them we show how to implement the subroutines of RBE according to Definition 4.*

- $\mathrm{Stp}(1^\kappa) \to (\mathsf{pp}_0)$, *where* $\mathsf{pp}_0 = \mathsf{hk}$ *is sampled from* $\mathrm{HGen}(1^\kappa, 1^{3\kappa})$.
- $\mathrm{Reg}^{[\mathsf{aux}]}(\mathsf{pp}_n, \mathsf{id}, \mathsf{pk}) \to \mathsf{pp}_{n+1}$. *This algorithm works as follows:*
  1. (=) *Parse* $\mathsf{aux}_n := (\{\mathsf{Tree}_1, \ldots, \mathsf{Tree}_\eta\}, (\mathsf{id}_1, \ldots, \mathsf{id}_n))$ *where the trees have corresponding depths* $\mathsf{d}_1 > \mathsf{d}_2 \cdots > \mathsf{d}_\eta$, *and* $(\mathsf{id}_1, \ldots, \mathsf{id}_n)$ *is the order the identities registered.*[10]
  2. *Parse* pp_n *as a sequence* $(\mathsf{hk}, (\mathsf{rt}_1, \mathsf{d}_1), \ldots, (\mathsf{rt}_\eta, \mathsf{d}_\eta))$ *where* $\mathsf{rt}_i \in \{0,1\}^\kappa$ *represents the root of tree* Tree_i *and* d_i *represents the depth of* Tree_i.
  3. *Create a new tree* $\mathsf{Tree}_{\eta+1}$ *with leaves* id, pk *and set its root as* $\mathsf{rt}_{\eta+1} \leftarrow \mathrm{Hash}(\mathsf{hk}, \mathsf{id}||\mathsf{pk}||0^\kappa)$ *and thus its depth would be* $\mathsf{d}_{\eta+1} = 1$.
  4. (=) *Let* $\mathcal{T} = \{\mathsf{Tree}_1, \ldots, \mathsf{Tree}_{\eta+1}\}$. *(We will keep changing* $\mathcal{T}$ *in step below.)*
  5. *While there are two different trees* $\mathsf{Tree}_L, \mathsf{Tree}_R \in \mathcal{T}$ *of the same depth* d *and size* $s = 2^{\mathsf{d}}$ *(recall that our trees are always full binary trees).*
     (a) *Obtain new* Tree *of depth* $d+1$ *by merging the two trees* Tree_L *and* Tree_R *as follows.*
     (b) (⋆⋆) *Let* $\mathsf{id}_1 \ldots \mathsf{id}_{n'}$ *and* $\mathsf{pk}_1 \ldots \mathsf{pk}_{n'}$ *be the identities and public keys of* n' *users in both trees* Tree_L *and* Tree_R *combined in* sorted order *according to identities.*
     (c) *For each* $i \in [n']$, *let* $h_{0,i} := \mathrm{Hash}(\mathsf{hk}, \mathsf{id}_i||\mathsf{pk}_i||0^\kappa)$.
     (d) (⋆⋆) *Next for each* $j \in \{1, \ldots \log n'\}$ *and* $k \in \{0, \ldots, (n'/2^j) - 1\}$, *let*

$$h_{j,k} = \mathrm{Hash}(\mathsf{hk}, h_{j-1,2k}||h_{j-1,2k+1}||\mathsf{id}[j,k])$$

     where $\mathsf{id}[j,k]$ *is the largest identity in the left child (which is the node with label* $h_{j-1,2k}$*); namely* $\mathsf{id}[j,k] = \mathsf{id}_{(2k+1)\cdot 2^{j-1}}$. *This completes the description of* Tree.
     (e) (=) *Remove both of* $\mathsf{Tree}_L, \mathsf{Tree}_R$ *from* $\mathcal{T}$ *and add* Tree *to* $\mathcal{T}$ *instead.*

[10] Keeping this list is not necessary, but simplifies the presentation of the updates.

6. *Let* $\mathcal{T} = \{\mathsf{Tree}_1, \dots, \mathsf{Tree}_\zeta\}$ *where* $\mathsf{d}'_1 > \cdots > \mathsf{d}'_\zeta$ *is their corresponding depth and* $\mathsf{rt}'_1, \dots, \mathsf{rt}'_\zeta$ *is their corresponding roots. Set* $\mathsf{pp}_{n+1}, \mathsf{aux}_{n+1}$ *as*

$$\mathsf{aux}_{n+1} = (\mathcal{T}, (\mathsf{id}_1, \dots, \mathsf{id}_n, \mathsf{id}_{n+1} = \mathsf{id})), \mathsf{pp}_{n+1} = (\mathsf{hk}, (\mathsf{rt}'_1, \mathsf{d}'_1), \dots, (\mathsf{rt}'_\zeta, \mathsf{d}'_\zeta)).$$

– $\mathrm{Enc}(\mathsf{pp}, \mathsf{id}, \mathsf{m}) \to \mathsf{ct}$*:*
  1. *Parse* $\mathsf{pp} := (\mathsf{hk}, (\mathsf{rt}_1, \mathsf{d}_1), \dots, (\mathsf{rt}_\eta, \mathsf{d}_\eta))$.
  2. *For each* $i \in \{1, \dots \eta\}$ *and* $j \in \{1, \dots, \mathsf{d}_i\}$*, sample* $\mathsf{stt}_{i,j} \leftarrow \{0,1\}^\kappa$ *and generate* $\widetilde{\mathrm{P}}_{i,j} \leftarrow \mathrm{HG}(\mathsf{hk}, \mathrm{P}_{i,j}, \mathsf{stt}_{i,j})$*, where* $\mathrm{P}_{i,j}$ *is explained below.*
  3. *For each* $i \in [\eta]$ *obtain* $\widetilde{y}_{i,1} \leftarrow \mathrm{HInp}(\mathsf{hk}, \mathsf{rt}_i, \mathsf{stt}_{i,1})$.
  4. *Output the ciphertext* $\mathsf{ct} = (\mathsf{pp}, \{\widetilde{\mathrm{P}}_{i,j}\}_{i,j}, \{\widetilde{y}_{i,1}\}_i)$.

  The program $\mathrm{P}_{i,j}$ *works as follows:*

  Hardwired values: $\mathsf{rt}_i, \mathsf{d}_i, \mathsf{hk}, \mathsf{m}, \mathsf{id}, \mathsf{r}, \mathsf{stt}_{i,j+1}$ *(where* $\mathsf{stt}_{i,\mathsf{d}_i+1} = \perp$*)*

  Input: $a||b||\mathsf{id}^*$
  1. *If* $\mathsf{id}^* = 0^\kappa$[11] *and* $a = \mathsf{id}$ *then output* $\mathrm{E}(b, \mathsf{m}; \mathsf{r})$.
  2. *If* $\mathsf{id}^* = 0^\kappa$ *and* $a \neq \mathsf{id}$ *then output* $\perp$.
  3. *If* $\mathsf{id} > \mathsf{id}^*$ *then output* $\mathrm{HInp}(\mathsf{hk}, b, \mathsf{stt}_{i,j+1})$*, else output* $\mathrm{HInp}(\mathsf{hk}, a, \mathsf{stt}_{i,j+1})$.

– $\mathrm{Upd}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{id}) \to \mathsf{u}$*: If* id *is a leaf in a tree of* aux*, say* Tree*, return the* whole *Merkle opening* pth *of leaf* id *and its sibling* pk *to the root* $\mathsf{rt}(\mathsf{Tree})$*. Otherwise, return* $\perp$.

– $\mathrm{Dec}(\mathsf{sk}, \mathsf{u}, \mathsf{ct}) \to m$*: Parse* $\mathsf{ct} = (\mathsf{pp}, \{\widetilde{\mathrm{P}}_{i,j}\}_{i,j}, \{\widetilde{y}_{i,1}\}_i)$ *and* $\mathsf{u} := (z_1 \dots z_{\mathsf{d}_{i^*}})$*. Let* i^* *be the index of the tree that holds the corresponding identity.*[12] *Decryption proceeds as follows:*
  1. *For* $j = \{1 \dots \mathsf{d}_{i^*} - 1\}$ *do*
     - $\widetilde{y}_{i^*,j+1} = \widetilde{\mathrm{P}}_{i^*,j}(\widetilde{y}_{i^*,j}, z_j)$.
  2. *Let* $\mathsf{ct} := \widetilde{\mathrm{P}}_{i^*,\mathsf{d}_{i^*}}(\widetilde{y}_{i^*,\mathsf{d}_{i^*}}, z_{\mathsf{d}_{i^*}})$.
  3. *Output* $\mathrm{D}(\mathsf{sk}, \mathsf{ct})$.

Theorem 21. *The RBE of Construction 20 satisfies the compactness, completeness (Definition 6), and security (Definition 10) properties.*

In the rest of this section, we prove Theorem 21. The completeness and compactness properties are proved similar to those of Construction 12. We can again verify that over the course of the system's execution, the tree that holds a user id, will not be merged with other trees more than $\log n$ times. (Each merge increases the depth of the tree by one, and the depth cannot bypass $\log n$.) We may use this fact to conclude all the efficiency features for the RBE scheme.

In the rest of this section, we focus on proving security.

[11] Without loss of generality we assume that no user is assigned the identity 0^κ.

[12] Alternatively, we may perform this with respect to all values of i^*, which is up to the number of trees in the system.

5.1 Proof of Security

Similar to our presentation of the proof of Construction 12, here also we first start by giving the proof for the case in which only *one* user has registered. We will then present the general proof.

Hardwired: rt, hk, $\mathsf{m} \in \{0,1\}$, id', r and stt

Input: $(\mathsf{id}, \mathsf{pk}, \mathsf{id}^*)$

1. If $\mathsf{id}^* \neq 0^\kappa$ or $\mathsf{id} \neq \mathsf{id}'$, then output $\perp$ and end.
2. Output $\mathrm{E}(\mathsf{pk}, \mathsf{m}; \mathsf{r})$ and end.

Fig. 7: Circuit P used for encryption of m to identity id'

Theorem 22 (Security). *For any identity* id' *we have*

$$(\mathrm{HG}(\mathsf{hk}, \mathrm{P}_0, \mathsf{stt}), \mathrm{HInp}(\mathsf{hk}, \mathsf{rt}, \mathsf{stt})) \overset{c}{\approx} (\mathrm{HG}(\mathsf{hk}, \mathrm{P}_1, \mathsf{stt}), \mathrm{HInp}(\mathsf{hk}, \mathsf{rt}, \mathsf{stt})) \quad (12)$$

where $\mathsf{hk} \leftarrow \mathrm{HGen}(1^\kappa, 1^{3\kappa})$, $\mathsf{stt} \leftarrow \{0,1\}^\kappa$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathrm{G}(1^\kappa)$, $\mathsf{rt} := \mathrm{Hash}(\mathsf{hk}, (\mathsf{id}', \mathsf{pk}, 0^\kappa))$ *and for* $\mathsf{m} \in \{0,1\}$ *the circuit program* P_m *is defined as*

$$\mathrm{P}_\mathsf{m} := \mathrm{P}[\mathsf{rt}, \mathsf{hk}, \mathsf{m}, \mathsf{id}', \mathsf{r}, \mathsf{stt}]. \quad (13)$$

Proof. For $\mathsf{m} \in \{0,1\}$ let ct_m denote the challenge ciphertext, namely

$$\mathsf{ct}_\mathsf{m} := (\mathrm{HG}(\mathsf{hk}, \mathrm{P}_0, \mathsf{stt}), \mathrm{HInp}(\mathsf{hk}, \mathsf{rt}, \mathsf{stt})), \quad (14)$$

where all the variables are sampled as in the theorem. We need to show $\mathsf{ct}_0 \overset{c}{\approx} \mathsf{ct}_1$. By simulation security of the hash garbling scheme, for both $\mathsf{m} \in \{0,1\}$ we have

$$\mathsf{ct}_\mathsf{m} \overset{c}{\approx} \mathsf{Sim}(\mathsf{hk}, (\mathsf{id}', \mathsf{pk}, 0^\kappa), 1^{|\mathrm{P}_\mathsf{m}|}, \mathrm{E}(\mathsf{pk}, \mathsf{m}; \mathsf{r})). \quad (15)$$

By semantic security of the underlying public-key encryption scheme we have

$$\mathsf{Sim}(\mathsf{hk}, (\mathsf{id}', \mathsf{pk}, 0^\kappa), 1^{|\mathrm{P}_0|}, \mathrm{E}(\mathsf{pk}, 0; \mathsf{r})) \overset{c}{\approx} \mathsf{Sim}(\mathsf{hk}, (\mathsf{id}', \mathsf{pk}, 0^\kappa), 1^{|\mathrm{P}_1|}, \mathrm{E}(\mathsf{pk}, 1; \mathsf{r})), \quad (16)$$

and so we obtain $\mathsf{ct}_0 \overset{c}{\approx} \mathsf{ct}_1$. □

Proof for the general case. As in the proof in Section 4.2 we may assume that at the time of encryption we have only one tree. The proof for the case of multiple trees is the same.

Proof. Suppose at the time of encryption the underlying tree with root rt has depth d. In the sequel we shall write P_j for $j \in [\mathsf{d}]$ to refer to the circuit program $\mathrm{P}_{1,j}$ described in our RBE construction. That is,

$$\mathrm{P}_1 \equiv \mathrm{P}_{1,1}[\mathsf{rt}, \mathsf{d}, \mathsf{hk}, \mathsf{m}, \mathsf{id}, \mathsf{r}, \mathsf{stt}_{1,2}], \quad (17)$$

and for $j > 1$

$$\mathrm{P}_j \equiv \mathrm{P}_{1,j}[\mathsf{rt}, \mathsf{d}, \mathsf{hk}, \mathsf{m}, \mathsf{id}, \mathsf{r}, \mathsf{stt}_{1,j+1}], \tag{18}$$

where all the variables above are as in the encryption of the construction.

For $j \in [\mathsf{d}]$ we define rt_j to be the node in the jth level of the tree (where we consider the root as level one), whose sub-tree contains the leaf with label id.[13] For example, if the path leading to id is

$$[(\mathsf{id}, \mathsf{pk}, 0^\kappa), (a_1, b_1, \mathsf{id}_1, \mathsf{left}), \dots, (a_{\mathsf{d}-1}, b_{\mathsf{d}-1}, \mathsf{id}_{\mathsf{d}-1}, \mathsf{right}), \mathsf{rt}],$$

then $\mathsf{rt}_3 = b_{\mathsf{d}-1}$. For $j > 1$ we define

$$\widetilde{y}_j := \mathrm{HInp}(\mathsf{hk}, \mathsf{rt}_j, \mathsf{stt}_{1,j}). \tag{19}$$

We also define X_j for $j \in [t+1]$ to be the concatenate result of the node values in level j of the path leading to id. For instance, in the example above we have $X_1 = (a_{\mathsf{d}-1}, b_{\mathsf{d}-1}, \mathsf{id}_{\mathsf{d}-1})$.

Let $\mathsf{stt}_i := \mathsf{stt}_{1,i}$. Recall that P_i has stt_{i+1} hardwired, which is the state used to hash-garble P_{i+1}. Via a sequence of hybrids, we show how to replace garbled versions of P_i's, starting with $i = 1$, so that in the ith hybrid the values of $\mathsf{stt}_1, \dots, \mathsf{stt}_i$ are never used.

– **Hybrid** 0 **(true encryption)**: The ciphertext is $\mathsf{ct}_0 := (\widetilde{\mathrm{P}}_1, \widetilde{\mathrm{P}}_2, \dots, \widetilde{\mathrm{P}}_{\mathsf{d}}, \widetilde{y}_1)$, where all of the values are sampled as in the construction.
– **Hybrid** 1: The ciphertext is $\mathsf{ct}_1 := (\widetilde{\mathrm{P}}_{1,\mathsf{sim}}, \widetilde{\mathrm{P}}_2, \dots, \widetilde{\mathrm{P}}_{\mathsf{d}}, \widetilde{y}_{1,\mathsf{sim}})$, where $\widetilde{\mathrm{P}}_2, \dots, \widetilde{\mathrm{P}}_{\mathsf{d}}$ are sampled as in the construction, and where $\widetilde{\mathrm{P}}_{1,\mathsf{sim}}$ and $\widetilde{y}_{1,\mathsf{sim}}$ are sampled as follows:

$$(\widetilde{\mathrm{P}}_{1,\mathsf{sim}}, \widetilde{y}_{1,\mathsf{sim}}) \leftarrow \mathsf{Sim}(\mathsf{hk}, X_1, 1^{|\mathrm{P}_1|}, \widetilde{y}_2). \tag{20}$$

– **Hybird** $i \in [\mathsf{d}-1]$:

$$\mathsf{ct}_i := (\widetilde{\mathrm{P}}_{1,\mathsf{sim}}, \dots, \widetilde{\mathrm{P}}_{i,\mathsf{sim}}, \widetilde{\mathrm{P}}_{i+1}, \dots, \widetilde{\mathrm{P}}_{\mathsf{d}}, \widetilde{y}_{1,\mathsf{sim}}),$$

where for $j \in [i]$:

$$(\widetilde{\mathrm{P}}_{j,\mathsf{sim}}, \widetilde{y}_{j,\mathsf{sim}}) \leftarrow \mathsf{Sim}(\mathsf{hk}, X_{j+1}, 1^{|\mathrm{P}_j|}, \widetilde{y}_{j+1}) \tag{21}$$

– **Hybrid** d:

$$\mathsf{ct}_{\mathsf{d}} := (\widetilde{\mathrm{P}}_{1,\mathsf{sim}}, \dots, \widetilde{\mathrm{P}}_{\mathsf{d},\mathsf{sim}}, \widetilde{y}_{1,\mathsf{sim}})),$$

where for $j \in [\mathsf{d}-1]$:

$$(\widetilde{\mathrm{P}}_{j,\mathsf{sim}}, \widetilde{y}_{j,\mathsf{sim}}) \leftarrow \mathsf{Sim}(\mathsf{hk}, X_{j+1}, 1^{|\mathrm{P}_j|}, \widetilde{y}_{j+1}), \tag{22}$$

and

$$(\widetilde{\mathrm{P}}_{\mathsf{d},\mathsf{sim}}, \widetilde{y}_{\mathsf{d},\mathsf{sim}}) \leftarrow \mathsf{Sim}(\mathsf{hk}, (\mathsf{id}, \mathsf{pk}, 0^\kappa), 1^{|\mathrm{P}_{\mathsf{d}}|}, \mathrm{E}(\mathsf{pk}, \mathsf{m}; r)). \tag{23}$$

[13] Recall that by Definition 10 the challenge identity id must have been registered before, and exactly once.

Now exactly as in the proof of Theorem 22, using the simulation security of the underlying HO scheme, we can show the indistinguishability of each two adjacent hybrids. Moreover, in the last hybrid, again using simulation security and as in the proof of Theorem 22, we may switch the underlying bit value of m. The proof is now complete.

□

References

1. Sattam S Al-Riyami and Kenneth G Paterson. Certificateless public key cryptography. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 452–473. Springer, 2003.
2. Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.
3. Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.
4. Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous IBE, leakage resilience and circular security from new assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 535–564, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.
5. Liqun Chen, Keith Harrison, David Soldera, and Nigel P Smart. Applications of multiple trust authorities in pairing based cryptosystems. In *Infrastructure security*, pages 260–275. Springer, 2002.
6. Zhaohui Cheng, Richard Comley, and Luminita Vasiu. Remove key escrow from the identity-based encryption system. In *Exploring New Frontiers of Theoretical Informatics*, pages 37–50. Springer, 2004.
7. Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 33–65, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.
8. Sherman SM Chow. Removing escrow from identity-based encryption. In *International Workshop on Public Key Cryptography*, pages 256–276. Springer, 2009.
9. Clifford Cocks. An identity based encryption scheme based on quadratic residues. In Bahram Honary, editor, *8th IMA International Conference on Cryptography and Coding*, volume 2260 of *Lecture Notes in Computer Science*, pages 360–363, Cirencester, UK, December 17–19, 2001. Springer, Heidelberg, Germany.
10. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
11. Nico Döttling and Sanjam Garg. Identity-based encryption from the Diffie-Hellman assumption. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*,

pages 537–569, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

12. Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Daniel Masny. New constructions of identity-based and key-dependent message secure encryption schemes. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018: 21st International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 10769 of *Lecture Notes in Computer Science*, pages 3–31, Rio de Janeiro, Brazil, March 25–29, 2018. Springer, Heidelberg, Germany.
13. Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Foundations of Computer Science*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press.
14. Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 467–476, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.
15. Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th Annual ACM Symposium on Theory of Computing*, pages 365–377, San Francisco, CA, USA, May 5–7, 1982. ACM Press.
16. Vipul Goyal. Reducing trust in the PKG in identity based cryptosystems. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 430–447, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Heidelberg, Germany.
17. Vipul Goyal, Steve Lu, Amit Sahai, and Brent Waters. Black-box accountable authority identity-based encryption. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 427–436. ACM, 2008.
18. Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015: 6th Conference on Innovations in Theoretical Computer Science*, pages 163–172, Rehovot, Israel, January 11–13, 2015. Association for Computing Machinery.
19. Aniket Kate and Ian Goldberg. Distributed private-key generators for identity-based cryptography. In *International Conference on Security and Cryptography for Networks*, pages 436–453. Springer, 2010.
20. Kenneth G Paterson and Sriramkrishnan Srinivasan. Security and anonymity of identity-based encryption with multiple trusted authorities. In *International Conference on Pairing-Based Cryptography*, pages 354–375. Springer, 2008.
21. Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.
22. Phillip Rogaway. The moral character of cryptographic work. Cryptology ePrint Archive, Report 2015/1162, 2015. http://eprint.iacr.org/2015/1162.
23. Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Heidelberg, Germany.
24. Quanyun Wei, Fang Qi, and Zhe Tang. Remove key escrow from the BF and Gentry identity-based encryption with non-interactive key generation. *Telecommunication Systems*, pages 1–10, 2018.