Cuckoo Commitments: Registration-Based Encryption and Key-Value Map Commitments for Large Spaces

Dario Fiore[1], Dimitris Kolonelos[1,2], and Paola de Perthuis[3,4]

[1] IMDEA Software Institute, Madrid, Spain
{dario.fiore,dimitris.kolonelos}@imdea.org
[2] Universidad Politénica de Madrid, Madrid, Spain
[3] DIENS, École Normale Supérieure, CNRS, Inria, PSL University, Paris, France
paola.de.perthuis@ens.fr
[4] Cosmian, Paris, France

Abstract. Registration-Based Encryption (RBE) [Garg et al. TCC'18] is a public-key encryption mechanism in which users generate their own public and secret keys, and register their public keys with a central authority called the key curator. Similarly to Identity-Based Encryption (IBE), in RBE users can encrypt by only knowing the public parameters and the public identity of the recipient. Unlike IBE, though, RBE does not suffer the key escrow problem—one of the main obstacles of IBE's adoption in practice—since the key curator holds no secret.

In this work, we put forward a new methodology to construct RBE schemes that support large users identities (i.e., arbitrary strings). Our main result is the first efficient pairing-based RBE for large identities. Prior to our work, the most efficient RBE is that of [Glaeser et al. ePrint'22] which only supports small identities. The only known RBE schemes with large identities are realized either through expensive non-black-box techniques (ciphertexts of 3.6 TB for 1000 users), via a specialized lattice-based construction [Döttling et al. Eurocrypt'23] (ciphertexts of 2.4 GB), or through the superfluous complexity of Registered Attribute-Based Encryption [Hohenberger et al. Eurocrypt'23]. By unlocking the use of pairings for RBE with large identity space, we enable a further improvement of three orders of magnitude, as our ciphertexts for a system with 1000 users are 1.7 MB.

The core technique of our approach is a novel use of cuckoo hashing in cryptography that can be of independent interest. We give two main applications. The first one is the aforementioned RBE methodology, where we use cuckoo hashing to compile an RBE with small identities into one for large identities. The second one is a way to convert any vector commitment scheme into a key-value map commitment. For instance, this leads to the first algebraic pairing-based key-value map commitments.

1 Introduction

Registration-Based Encryption (RBE), introduced by Garg et al. [GHMR18], is a public key encryption mechanism in which users generate their own public and secret keys, and register their public keys with a central authority called the *Key Curator* (KC). The responsibility of the KC is to maintain the system's public parameters updated every time a new user joins. In RBE, Alice can send an encrypted message to Bob by only knowing the public parameters and Bob's identity. On the other hand, in order to decrypt, Bob uses his secret key and a small piece of information, *the opening*, that can be retrieved from the KC. An RBE scheme should have compact public parameters, and its algorithms for encryption and decryption should be sublinear in the number of registered users. In terms of security, RBE guarantees that messages encrypted under an identity id stay confidential (in a usual semantic security fashion) as long as id is an honest user or id did not register in the system.

Registration-based encryption can be seen as an hybrid between traditional Public-Key Encryption (PKE) [DH76, RSA78] and Identity-Based Encryption (IBE) [Sha84]. The most appealing feature of RBE is to remove the need of trusted parties, which is a common issue, for different reasons, in PKE and IBE. In IBE, a trusted authority is responsible to generate users' secret keys and thus can decrypt any message in the system, a problem known as key escrow. In traditional PKE, one needs a trusted authority, the PKI, in order to certify ownership of public keys; PKIs are however complex to implement and manage. In contrast, while an RBE system still involves an authority, the key curator, the main benefit is that *the KC does not hold any secret* and *its behavior is completely transparent*, to the point that it can be replicated (and thus audited) by any user in the system. Therefore, RBE can be a promising alternative to realize public key encryption with simple, safe, and transparent key management.

The approaches used to construct the first proposals of RBE [GHMR18, GHM$^+$19] rely either on indistinguishability obfuscation or the garbled circuit tree technique of [CDG$^+$17]. In spite of their power, these techniques are prohibitively expensive. For instance, based on estimations from [CES21] an RBE based on garbled circuits with a thousand users would have ciphertexts of 3.6 terabytes (which [CES21] can reduce by approximately 45%). As observed in [GKMR22], this high cost is (partially) due to their *non-black-box* use of cryptographic schemes—an approach that is notoriously expensive.

Two very recent works [GKMR22, DKL$^+$23] have filled this gap by proposing efficient, *black-box* constructions of RBE that are based on bilinear pairings and lattices respectively. On the good side, the schemes of [GKMR22, DKL$^+$23] achieve feasible efficiency—both report implementations confirming encryption and decryption time in the order of milliseconds, public parameters in the order of a few MBs. On the other hand, this efficiency profile comes at the price of some limitations. The work of Glaeser et al. [GKMR22] achieves their efficiency by limiting the identity space to the set of polynomial-size integers $\{1, \dots, n\}$. Although this identity space fits a few application scenarios (e.g., if identities are phone numbers), it rules out many more. In practice, the desiderata is to support identities that can be arbitrary strings (e.g., email addresses, arbitrary usernames). The work of Döttling et al. [DKL$^+$23] manages to solve this issue. They propose an RBE construction for arbitrary identities based on the LWE problem. Nevertheless, their ciphertext size is still far from desirable in practice: for n registered users their ciphertexts consist of $\approx 2\lambda \log n$ LWE ciphertexts (concretely, 2.4 GB for a system with 1024 registered users).

Finally, another recent work, by Hohenberger et al. [HLWW23], introduces the notion of Registered Attribute-Based Encryption (R-ABE) and gives *black-box* constructions from composite-order bilinear groups. One can generically transform an R-ABE to an RBE scheme with unbounded identities. Unfortunately, the construction of of [HLWW23] inherits the complexity of the enhanced functionality of ABE, therefore the resulting RBE with unbounded identities would be overly complicated and concretely inefficient.

1.1 Our Contributions

In this work, we continue the line of research on constructing efficient and black-box registration-based encryption.

Pairing-based RBE. Our main result is *the first RBE scheme for unbounded identity spaces that is black-box and based on prime-order bilinear groups.* The interest of an RBE from pairings is twofold. First, we show how to support large identities using an algebraic structure that is substantially more limited than lattices. Second, pairings lend themselves to efficient implementations and in fact our scheme achieves much shorter ciphertexts than the state-of-the-art RBE for large identities from [DKL$^+$23]. Concretely, a ciphertext of our RBE is 1.67MB for 1024 users and identity space $\{0, 1\}^{2\lambda}$. In other words, by unlocking the use of pairings for RBE with large identities we show yet another three-orders-of-magnitude improvement in this research line.

We should highlight that, as mentioned above, an RBE from pairings can also be constructed using the R-ABE scheme of [HLWW23]. However, it would be over composite order bilinear groups, where the order has an unknown factorization, making it less efficient and cumbersome for implementations.

We provide a comparison of our schemes with the state-of-the-art black-box constructions in Table 1. A thorough analysis of the table can be found in Section 5.3.

	Setting	$\mathcal{ID}$	Compactness	$\lvert\mathsf{ct}\rvert$	#updates	$\lvert\mathsf{pp}\rvert + \lvert\mathsf{crs}\rvert$
[HLWW23]	Pairings (C)	$\{0,1\}^*$	Adaptive	$O(\lambda \log n)$	$\log n$	$O(\lambda n^{2/3} \log n)$
[GKMR22]	Pairings (P)	$[1, n]$	Adaptive	$4 \log n$	$\log n$	$O(\sqrt{n} \log n)$
Ours P1	Pairings (P)	$\{0,1\}^*$	Adaptive	$6\lambda \log n$	$\log n$	$O(\sqrt{\lambda n} \log n)$
Ours P2	Pairings (P)	$\{0,1\}^*$	Selective	$12 \log n$	$\log n$	$O(\sqrt{n} \log n)$
[DKL+23]	Lattices	$\{0,1\}^*$	Adaptive	$(2\lambda + 1) \log n$	$\log n$	$O(\log n)$
Ours L	Lattices	$\{0,1\}^*$	Selective	$4 \log^2 n$	$\log n$	$O(\log n)$

Table 1: Comparison of the schemes resulting from different instantiations of our compiler. n is the maximum number of users to be registered. Parings (P) indicates prime order groups and Pairings (C) composite order groups respectively. $\lvert\mathsf{ct}\rvert$ in the pairing construction is measured in group elements and in the Lattice constructions LWE ciphertexts.

Novel construction methodology for RBE. To achieve this milestone, our technical contribution is a novel methodology to construct black-box RBE schemes that can accommodate exponentially large identity spaces, i.e., $\mathsf{id} \in \{0,1\}^*$. Prior to our work, this was a challenging problem solved either through the use of non-black-box techniques [GHMR18, GHM+19], via a specialized construction based on LWE [DKL+23] or going through the heavier notion of R-ABE. Our approach instead consists of a generic compiler that yields several RBE instantiations based on a variety of assumptions, in the random oracle model.

The core technique of our approach is a *novel use of cuckoo hashing* [PR04] in cryptography that can be of independent interest. Cuckoo hashing is a powerful (probabilistic) technique to store elements from a large universe $\mathcal{X}$ into a small table $\boldsymbol{T}$ so that one can later access them in constant-time. Concretely, the latter means that for an element x the cuckoo hash returns $k = O(1)$ possible locations of $\boldsymbol{T}$ where to find x; the cuckoo hashing algorithms take care of resolving collisions by reallocating elements in $\boldsymbol{T}$ whenever a collision occurs.

In this work, we present a compiler that takes an RBE scheme for a polynomial-size identity space $\overline{\mathcal{ID}} = \{1, \ldots, n\}$ and boosts it to become an RBE for large identity space $\mathcal{ID} = \{0,1\}^*$. We start with the idea of using cuckoo hashing to map identities in $\mathcal{ID}$ to polynomial-size integers in $\overline{\mathcal{ID}}$ so that user id becomes user $H(\mathsf{id})$ in the underlying RBE. Unsurprisingly, this simple idea does not work straightforwardly. The main obstacle is that the cuckoo hashing algorithms "move" elements around different locations during the lifetime of the system. This implies that a user id assigned to location $j = H(\mathsf{id})$ might decrypt ciphertexts that were previously generated for another user id^* that was assigned to the same location j in the past. In our compiler, we resolve these "collisions" thanks to a novel combination of the RBE with *Witness Encryption for Vector Commitments* (WE for VC), and a secret sharing scheme. A Vector Commitment (VC) scheme [LY10, CF13] allows one to compute a short commitment to a vector $\boldsymbol{v}$ and later locally open at a specific position j. A WE for VC is a special-purpose witness encryption [GGSW13] thanks to which a party can encrypt a message m w.r.t. a commitment C, position j, and value y, and m can be decrypted by anyone holding a valid opening of C at the correct value $y = v_j$. Interestingly, we show how to construct this class of WE based on well established assumptions over pairings (DHE [BGW05]) and lattices (LWE [Reg05]). We refer to our technical overview (Section 2) for more details.

Additional contributions. To confirm the power of our cuckoo hashing technique, we show additional results that we discuss hereafter.

New lattice-based RBE. Through our RBE compiler, we also obtain new RBE schemes based on LWE. We do this by instantiating the RBE of [DKL$^+$23] with a small identity space and then boosting it to large identities through our compiler. This instantiation though does not improve over the large-identity instantiation of [DKL$^+$23]; this is due to the fact that we need a robust[5] cuckoo hash [Yeo23] that produces $k = \lambda$ indices for every element and blows our ciphertexts by a factor λ. Interestingly, though, we need the robustness property of cuckoo hashing only to ensure that the public parameters stay polylogarithmic *in the worst case*. Based on this observation, we can also use a (non-robust) cuckoo hashing where $k = 2$ and obtain an LWE-based RBE that has shorter ciphertexts than [DKL$^+$23] (ours has of $4 \log^2 n$ LWE encryptions, as opposed to $2\lambda \log n$). Our RBE scheme is correct and secure, but achieves compact parameters only against selective adversaries. We refer to Section 4.4 for more details on this compactness model.

Application to key-value map commitments and accumulators. Based on the cuckoo hashing idea described above, we present a construction that compiles *any* vector commitment into a key-value map commitment (KVC) [BBF19, AR20] for arbitrary-size keys. In a nutshell, a KVC is a generalization of VCs in which one commits to a collection of key-value pairs $(\mathsf{k}_i, \mathsf{v}_i)$, i.e., VCs are a special case where keys are integers in $\{1, \ldots, n\}$. Thus the interesting problem is to realize KVCs with large keys, e.g., $\mathsf{k} \in \{0, 1\}^*$. Existing schemes are based on hidden-order groups [BBF19, AR20], Merkle trees or, very recently, lattices [dCP23].[6] In Section 6, we present a generic and black-box construction of (updatable) KVC obtained by combining any (updatable) VC and cuckoo hashing. Through this generic construction, we obtain new efficient KVCs; notably, the first updatable KVCs for large keys based on pairings.

Finally, we observe that KVCs (for large keys) imply accumulators (for large universe). By putting this observation together with our VC-to-KVC compiler, we obtain a way to convert VCs into accumulators. This connection was previously shown by Catalano and Fiore in [CF13] *but only for small universe*. Our results thus bridge this gap. Furthermore, we close the circle in showing the equivalence of VCs and universal accumulators, since the reversed implication (i.e., building VCs from universal accumulators) has been recently shown by Boneh, Bunz and Fisch [BBF19]. An outstanding implication is that our result yields the first accumulator for large universe based on the CDH problem in bilinear groups. Prior to our work, this result could only be achieved by using non-black-box techniques (e.g., a Merkle tree with a CDH-based VC).

Cuckoo hashing applications. Cuckoo Hashing has been used extensively in many contexts in cryptography, mainly to boost efficiency in oblivious two-party computations (*e.g.* in [PR10, PSSZ15, ACLS18, PPYY19]). However, in most of these contexts, due to the oblivious security model, the adversary does not have direct access to the cuckoo hash functions. Only recently, a new work has discussed cuckoo hashing in this perspective [Yeo23].

[5] Informally, a CH is robust if its correctness error is negligible for adversarially chosen inputs; standard correctness holds only for inputs chosen before public parameters.

[6] One can also use polynomial commitments, e.g., [KZG10], in combination with interpolation but to the best of our knowledge this KVC is not updatable.

In our work, we propose new cryptographic applications where cuckoo hashes can be publicly computed. In a way, our results show how vector commitment techniques can mitigate the shortcomings of publicly computable cuckoo hashing, as combining them with vector commitments enable their use while keeping constructions succinct and efficient. We believe that this approach can serve as inspiration for future applications.

1.2 Related Work

Here we mention more prior works that are relevant to the topic of ours.

Registered Encryption Primitives. As we mentioned, the first works on registration-based encryption (with large identities) were non-black box: [GHMR18] introduced the notion, [GHM$^+$19] showed a construction with more efficient registration computational complexity, [GV20] introduced the notion of *verifiability* for RBE and [CES21] improved the efficiency of the previous works by replacing the Merkle tree with a form of PATRICIA trie. Lately, there has been an increasing interest in generalizing RBE to registered fine-grained encryption such as Registered Attribute-Based Encryption [HLWW23] and Registered Functional Encryption [FFM$^+$23, DP23].

Cuckoo hashing in cryptography. Cuckoo hashing has been used in Cryptography in oblivous access primitives such as Oblivious RAM [PR10], Private Set Intersection [PSSZ15], Private Infomration Retrieval [ACLS18], and Searchable Encryption [PPYY19]. Recently, Yeo gave a formal treatment from a cryptographic perspective [Yeo23], again with the objective of discussing applications to PIR. To the best of our knowledge, our work is the first that uses cuckoo hashing in the context of fine-grained encryption and commitment schemes.

Key-Value Map Commitments and Accumulators. The notion Key-Value Map Commitments was introduced by Boneh et al. [BBF19] where they also presented a construction from Groups of Unknown order. Different KVC constructions from Groups of Unknown order exist [CFG$^+$20, AR20]. KVCs can also be realized by Merkle Trees. Recently deCastro and Peikert [dCP23] showed a construction from Lattices.

Accumulators were introduced by Benaloh and de Mare [Bd94]. Constructions for large universe exist from RSA groups [BP97, CL02], Groups of Unknown Order [Lip12, BBF19], q-type assumptions in bilinear groups [Ngu05], and Merkle trees. The recent work of de Castro and Peikert [dCP23] also implies an accumulator from lattices.

Lite-WE flavors. Witness Encryption for Merkle trees implicitly appears in [CDG$^+$17, DG17, GHMR18], using non-black box techniques (Garbling). Witness Encryption flavors for special purpose relation, with the objective to have a more efficient instantiation, have also been introduced in prior works [BL20, CDK$^+$22, CFK22, DHMW22]

2 Technical Overview

We give here an informal overview of the techniques that we introduce in this work to obtain our Registration-Based Encryption (RBE) and Key-Value Map Commitments (KVC) results. To put some context, we recall first Vector Commitments [CF13, LY10] a fundamental primitive for both RBE and KVC.

Vector Commitments. A Vector Commitment (VC) is a cryptographic primitive with which one can commit to a vector of elements in such a way that, at a later point, one can selectively open any position of the vector. Importantly, the commitments and the openings should be succinct (sublinear or polylogarithmic) in the size of the vector. The simplest form of VCs are Merkle trees.

We guide the reader through an example, the Libert-Yung VC [LY10], that we will also use in this work. It works over pairings, using a common reference string (CRS) $\mathsf{crs} = (g^{\alpha}, \dots, g^{\alpha^n}, g^{\alpha^{n+2}}, \dots, g^{\alpha^{2n}})$ and we denote $g_i = g^{\alpha^i}$. Committing to a vector $\boldsymbol{x} = (x_1, \dots, x_n)$ happens as $C = \prod_{i\in[n]} g_i^{x_i}$. To open the position i (to value x_i) we compute $\Lambda_i = \prod_{j\neq i}(g_{n+1-i+j})^{x_j}$. For the verification of the opening we check if $e(C, g_{n+1-i}) = e(\Lambda_i, g) \cdot e(g_i^{x_i}, g_{n+1-i})$. The VC is position binding under the n-Diffie-Hellman Exponent assumption [BGW05], a well-established q-type (falsifiable) assumption. Observe that C, Λ are just a single group element each, and the verification time is independent of the size of the vector.

2.1 Registration-Based Encryption with Unbounded Identity Space

Prior black-box RBE constructions. To date, the only RBE constructions that are black-box (i.e., they do not encode cryptographic operations in the circuit of another cryptographic primitive such as a Garbled Circuit) are the ones of Glaeser et al. [GKMR22] (henceforth GMKMR) and Döttling et al. [DKL$^+$23] (henceforth DKLLMR). The former works over pairings and the latter over lattices. For the sake of this overview we are only concerned with the former RBE. Furthermore, to simplify the exposition we omit efficiency tricks that retain the efficiency properties (compactness, number of updates) of RBE. We discuss them extensively in the main body of our work.

The GKMR RBE. The GKMR RBE [GKMR22] roughly works as follows. It uses [LY10] as an underlying vector commitment in order to commit (in a compressing way) to the public keys of all the users.

In more detail, the user i samples a secret key sk_i randomly and sends $\mathsf{pk}_i = g_i^{\mathsf{sk}_i}$ to the Key Curator (KC). Then the KC compresses the public keys of all users by computing $C = \prod_{i\in[n]} \mathsf{pk}_i = \prod_{i\in[n]} g_i^{\mathsf{sk}_i}$, and sets the public parameters as $\mathsf{pp} \leftarrow C$. In essence, C is a vector commitment to the vector of the secret keys $\mathbf{sk} = (\mathsf{sk}_1, \dots, \mathsf{sk}_n)$ of all registered user.

GKMR introduced a simple technique to encypt a message $m \in \mathbb{G}_T$ to the user i by only having C and, crucially, without having pk_i: Recalling that $e(C, g_{n+1-i}) = e(\Lambda_i, g) \cdot e(g_i^{\mathsf{sk}_i}, g_{n+1-i})$, one defines the ciphertext as $(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3) = (g^r, e(C, g_{n+1-i})^r, e(g_i, g_{n+1-i})^r \cdot m)$. Observe that $e(C, g_{n+1-i})^r = e(\Lambda_i, g)^r \cdot e(g_i, g_{n+1-i})^{r\cdot\mathsf{sk}_i}$, and thus $\left(\mathsf{ct}_2 \cdot e(\Lambda_i, \mathsf{ct}_1)^{-1}\right)^{\mathsf{sk}_i^{-1}} = \mathsf{ct}_3/m$. Hence, the user i, knowing sk_i and additionally Λ_i, can decrypt as $m^* = \mathsf{ct}_3/\left(\mathsf{ct}_2 \cdot e(\Lambda_i, \mathsf{ct}_1)^{-1}\right)^{\mathsf{sk}_i^{-1}}$.

In RBE terms, Λ_i represents the update information of user i that should be periodically fetched from the KC (whenever it is changed). The final note is that naively computing $\Lambda_i = \prod_{j\neq i}(g_{n+1-i+j})^{\mathsf{sk}_j}$ would need knowledge of $\mathbf{sk}$ to be computed. However, each user j can compute the cross-terms $(g_{n+1-i+j})^{\mathsf{sk}_j}$ for each $i \neq j$ previously registered, and send them to the KC to enable the KC to compute the Λ_i's. We summarize the GKMR RBE below:

$$\begin{aligned}
&\mathsf{crs} = \{g, g^{\alpha}, \dots, g^{\alpha^N}, g^{\alpha^{N+2}}, \dots, g^{2N}\}; && \mathsf{pk}_i = g_i^{\mathsf{sk}_i};\\
&\mathsf{pp} = C := g_1^{\mathsf{sk}_1} g_2^{\mathsf{sk}_2} \dots g_N^{\mathsf{sk}_N}; && \mathsf{u}_i = \Lambda_i := \prod_{j\neq i}(g_{N+1-i+j})^{\mathsf{sk}_j};\\
&\mathsf{ct} = \left(g^r, e(C, g_{N+1-i})^r, e(g_i, g^{N+1-i})^r \cdot m\right); && m^* = \mathsf{ct}_3/\left(\mathsf{ct}_2 \cdot e(\Lambda_i, \mathsf{ct}_1)^{-1}\right)^{\mathsf{sk}_i^{-1}}.
\end{aligned}$$

The limitation of bounded identities. In the scheme above, i plays the role of the user's identity. It is apparent from the construction that i should lie in $[1, n]$ and since the CRS is linear in n, n must be polynomially bounded, and so must be the RBE identity space. This limitation is acknowledged in [GKMR22] and is the main drawback of the, otherwise highly efficient, scheme.

In the following we describe our technique to overcome this limitation.

Our approach: Cuckoo Hashing. One may be tempted to use a hash function to map larger identities to $[1, n]$. However, naively this cannot work because of collisions: since $[1, n]$ is polynomial-size collisions are inevitable.

Our idea is to use Cuckoo Hashing (CH) [PR04] for the mapping $\{0, 1\}^* \to [1, n]$. Cuckoo Hashing is a powerful (probabilistic) technique to store elements from a large universe $\mathcal{X}$ in a small table $\boldsymbol{T}$ in constant time so that one can later efficiently access them, in constant time. Hence it is an inherent method to deal with collisions in a small space.

To put some context, we describe a simple version of Cuckoo Hashing with a stash [KMW10]. For this, we have 2 hash functions h_1, h_2, a table $\boldsymbol{T}$ of size $4n$ and a (unordered) set S, called the 'stash'. To insert a new element x, one first computes $x^{(1)} = h_1(x)$ and if $\boldsymbol{T}[x^{(1)}] = \mathsf{empty}$ then stores x in $\boldsymbol{T}[x^{(1)}]$. Otherwise, if $\boldsymbol{T}[x^{(1)}] = y$ then x 'evicts' y; namely, x is stored in $\boldsymbol{T}[x^{(1)}]$ and y is inserted in $\boldsymbol{T}[y^{(2)}]$ (assume for this example that y was previously 'sent' to $\boldsymbol{T}[x^{(1)}]$ using h_1). Subsequently, if $\boldsymbol{T}[y^{(2)}]$ is occupied by z then y 'evicts' z, and z gets sent to the location specified by the alternative hash function. Observe that one always begin with h_1 for a new element and when the element is evicted always uses the next hash function (and if the last is reached, then the first one again). This procedure continues until either an empty position is found or M attempts have been made. If the latter event occurs, then the last element that was evicted gets stored in the stash S. It can be shown that for random hash functions h_1, h_2, if $M = O(\lambda \log n)$ then the size of the stash is in $O(\log n)$ with overwhelming probability [ADW14].

There are many variants of the above mechanism: Cuckoo Hashing with $k > 2$ hash functions [FPSS03] or having tables where every position/bucket has capacity $\ell > 1$ [DW07]. We refer to [W$^+$17] for an insightful systematization of knowledge and [Yeo23] for an overview from a cryptographic perspective.

In our work, we formally define a *Cuckoo Hashing scheme* in a cryptographic manner in Section 3.2, closely following the definitions of [Yeo23]. For the rest, we mostly treat Cuckoo Hashing as a black-box, assuming that it uses k hash functions with buckets of size $\ell = 1$.

Cuckoo Hashing in RBE. Now let's see how one would use Cuckoo Hashing in the above RBE scheme in order to map large identities $\mathsf{id} \in \{0, 1\}^*$ to small representatives in $[1, n]$. From now on, we denote $\mathsf{id}^{(\eta)} = h_\eta(\mathsf{id})$ for short.

A user id who wishes to register in the system, computes $\mathsf{id}^{(1)} = h_1(\mathsf{id}), \ldots, \mathsf{id}^{(k)} = h_k(\mathsf{id})$, samples k different secret keys $\mathsf{sk}^{(1)}, \ldots, \mathsf{sk}^{(k)}$ and sends the corresponding public keys $g_{\mathsf{id}^{(1)}}^{\mathsf{sk}^{(1)}}, \ldots, g_{\mathsf{id}^{(k)}}^{\mathsf{sk}^{(k)}}$ to the KC. Then the KC inserts id in the system by Cuckoo Hashing it (KC keeps the table $\boldsymbol{T}$ of currently hashed identities). Assuming that id is eventually stored in $\boldsymbol{T}$ at position $\mathsf{id}^{(\eta)}$, KC has $\mathsf{pk}_{\mathsf{id}^{(\eta)}}$. To give an example, a potential instance of such a system could be:

$$\begin{array}{rcccccccc}
C = & g_1^{\mathsf{sk}_b^{(2)}} \cdot 1\cdot & g_3^{\mathsf{sk}_a^{(1)}} \cdot & g_4^{\mathsf{sk}_e^{(3)}} \cdot & g_5^{\mathsf{sk}_c^{(1)}} \cdot 1\cdot & & 1\cdot & g_8^{\mathsf{sk}_d^{(2)}} \\
\Big(\mathbf{sk} = (& \mathsf{sk}_b^{(2)},\ 0, & \mathsf{sk}_a^{(1)}, & \mathsf{sk}_e^{(3)}, & \mathsf{sk}_c^{(1)},\ 0, & & 0, & \mathsf{sk}_d^{(2)}) \Big) \\
\boldsymbol{T} = (& b,\quad 0, & a, & e, & c,\quad 0, & & 0, & d) \\
\hline
 & 1 \quad\ 2 & 3 & 4 & 5 \quad\ 6 & & 7 & 8
\end{array}$$

where $n = 8$, a, b, c, d, e are identities and $h_2(b) = 1$, $h_1(a) = 3$, $h_3(e) = 4$, $h_1(c) = 5$, $h_2(d) = 8$ (recall, $\mathbf{sk}$ is not known explicitly to the KC; and to highlight this is written with brackets in the examples).

Until now, we have resolved the collisions in a pragmatic way: thanks to cuckoo hashing, no collisions of identities are stored in the public parameters. This is however not clear from the Encryptor's perspective. The Encryptor wishing to encrypt for id does not have $\boldsymbol{T}$ and thus does not know in which position among $\mathsf{id}^{(1)}, \ldots, \mathsf{id}^{(k)}$ the identity is placed. Hence, she does not know which position to encrypt for, and could compromise security by encrypting for a position occupied by another identity id', in which case id' would read the message intended for id.

The missing piece: Witness Encryption for Vector Commitments. To solve the issue explained above, our approach is to use another Vector Commitment, D, this time to commit to the actual table $\boldsymbol{T}$ of identities rather than the secret keys. The above example is modified as follows:

$$\begin{array}{rcccccccc}
C = & g_1^{\mathsf{sk}_b^{(2)}} \cdot & 1\cdot & g_3^{\mathsf{sk}_a^{(1)}} \cdot & g_4^{\mathsf{sk}_e^{(3)}} \cdot & g_5^{\mathsf{sk}_c^{(1)}} \cdot & 1\cdot & 1\cdot & g_8^{\mathsf{sk}_d^{(2)}} \\
\Big(\mathsf{sk} = (& \mathsf{sk}_b^{(2)}, & 0, & \mathsf{sk}_a^{(1)}, & \mathsf{sk}_e^{(3)}, & \mathsf{sk}_c^{(1)}, & 0, & 0, & \mathsf{sk}_d^{(2)}) \Big) \\
D = & g_1^b\cdot & 1\cdot & g_3^a\cdot & g_4^e\cdot & g_5^c\cdot & 1\cdot & 1\cdot & g_8^d \\
\boldsymbol{T} = (& b, & 0, & a, & e, & c, & 0, & 0, & d) \\
\hline
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8
\end{array}$$

For such a configuration of the system, the wish is a cryptographic primitive that allows encrypting, when having in hands only D and the target id, and that should work as follows: If $\boldsymbol{T}[\mathsf{id}^{(\eta)}] = \mathsf{id}$ is in the committed vector, then anyone having the corresponding opening $\varPsi_{\mathsf{id}^{(\eta)}}$ can decrypt. Otherwise, if $\boldsymbol{T}[\mathsf{id}^{(\eta)}] \neq \mathsf{id}$ then the ciphertext is computationally indistinguishable for everyone.[7]

This mechanism is reminiscent of Witness Encyption (WE) [GGSW13], but only for a specific NP language and a slightly different notion of security. We formalize such a primitive and call it *Witness Encryption for Vector Commitments* (VCWE, see Section 4.1). Using a VCWE, the encryptor can:

1. Secret share the message m into two shares m_1, m_2
2. Encrypt m_1 for the position $\mathsf{id}^{(1)}$ using the above RBE for small identities.
3. Encrypt m_2 with the VCWE for commitment D, position $\mathsf{id}^{(1)}$, value id.

and repeat this *for every possible position* of id in the table, i.e. $\mathsf{id}^{(2)}, \ldots, \mathsf{id}^{(k)}$.

To argue the security of this idea, we note that:

- If $\boldsymbol{T}[\mathsf{id}^{(\eta)}] = \mathsf{id}$, then *everybody* can decrypt the second part of ciphertext (for security, we consider that $\boldsymbol{T}$ and thus $\varPsi_{\mathsf{id}^{(\eta)}}$ are public) and obtain m_2. On the other hand, $\boldsymbol{T}[\mathsf{id}^{(\eta)}] = \mathsf{id}$ means that the 'correct' user is registered in that position, hence only id can obtain the first share m_1.
- If $\boldsymbol{T}[\mathsf{id}^{(\eta)}] \neq \mathsf{id}$ then *nobody* can decrypt the second part of the ciphertext and obtain m_2. This follows from the security of VCWE.

VCWE constructions. Witness Encryption for all NP is notoriously hard to achieve in efficient ways with currently known constructions from multilinear maps [GGSW13, GLW14], indistinguishability obfuscation [GGH$^+$13, JLS21] or, recently, non-standard non-falsifiable lattice assumptions [Tsa22, VWW22].

Nevertheless, it turns out that for the specific relation above, there are surprisingly simple and efficient black-box solutions. Therefore, the VCWE ciphertext imposes a minimal overhead to the size of the overall ciphertext. In Section 5, we provide two simple VCWE schemes, over Pairings and Lattices respectively.

[7] We note that if $\boldsymbol{T}[\mathsf{id}^{(\eta)}] \neq \mathsf{id}$ then from position-binding of the VC no PPT party can compute a $\varPsi$ that verifies for id in position $\mathsf{id}^{(\eta)}$.

Final RBE scheme. In conclusion the Cuckoo Hashing technique in combination with the VCWE allow us to have a secure RBE with unbounded identities.

Dealing with the Stash. Finally, if the CH scheme has a stash S, then we demand that this stash is small (polylogarithmic or sublinear). This is because we store $S = \{(\mathsf{pk}_1, \mathsf{id}_1), \ldots, (\mathsf{pk}_s, \mathsf{id}_s)\}$ in the public parameters, and anyone who wants to encrypt w.r.t an id_i in the stash can do it, using a regular Public Key Encryption scheme. As we discuss in Section 3.2 there are CH schemes that have $|S| = \log n$ or even $|S| = 0$ even in the worst case.

2.2 Generalization and other implications

General Compiler. It is not difficult to see that the procedure described above, to enlarge the identity space, in a semi-generic way through the GKMR RBE, can be generalized. That is, we give a generic compiler that boosts *any* RBE scheme with small identity to one with large identity space, using Cuckoo Hashing, and Witness Encryption for Vector Commitments.

Lattice-based RBE with shorter ciphertexts. Our general compiler applies naturally to Lattice-Based RBE schemes. As previously mentioned, the DKLLMR RBE scheme [DKL+23] already allows for unbounded identities.

In spite of this, their ciphertext size is logarithmic in the size of the identity space: $|\mathsf{ct}| = \log(|\mathcal{ID}|) \log n$ LWE ciphertexts. This stems from the fact that their construction works with a (sparse) Merkle tree with one leaf per element of $\mathcal{ID}$, thus $|\mathcal{ID}|$ leaves, and then the ciphertext is roughly one LWE ciphertext per level of the tree. For a virtually unbounded identity space we need $\mathcal{ID} = \{0,1\}^{2\lambda}$ (so that we can use a collision resistant hash function $H : \{0,1\}^* \to \{0,1\}^{2\lambda}$), meaning $|\mathsf{ct}| = 2\lambda \log n$. This means that there is a Merkle tree with $2^{2\lambda}$ leaves, while only $n = \mathsf{poly}(\lambda)$ are going to be occupied.

Our idea is the following: Say that we want to support $n = \mathsf{poly}(\lambda)$ users, then we could start from a DKLLMR RBE with exactly n leaves (i.e. bounded identities), thus $|\mathsf{ct}| = \log^2 n$. Then we could apply our compiler to boost it to a full-fledged RBE with unbounded identities. Our hope is that, after applying our compiler, we could obtain an RBE with smaller ciphertexts than DKLLMR instantiated for $\mathcal{ID} = \{0,1\}^{2\lambda}$.

Our general compiler yields about $2k|\mathsf{ct}|$-sized ciphertexts (assuming that VCWE has roughly the same size as RBE which turns out to be the case, see Section 5.2. If we desire our RBE to have adversarial compactness then for technical reason related to CH (see Section 3.2we need to fix $k = \lambda$. This unfortunately does not let us achieve an improvement. However, if one relaxes compactness to hold only against selective adversaries, meaning adversaries who need to choose the set of identities they wish to register before seeing the Cuckoo hash functions, one can set $k = 2$ and obtain a significant improvement: $|\mathsf{ct}| \approx 4 \log^2$ in constast to $2\lambda \log n$ (initial DKLLMR), which concretely saves an $\frac{\lambda}{2 \log n}$ factor from the ciphertext. We defer the discussion and the formal definition of this selective notion to Section 4.4.

2.3 Key-Value Map Commitments and Accumulators

Finally, we informally describe how we can use our cuckoo hashing technique in the context of vector commitments, specifically to transform *any* VC scheme into a key-value map commitment for keys from a large space.

Assuming one needs to commit to a key-value map consisting of n key-value pairs $(\mathsf{k}_i, \mathsf{v}_i)$ for $i = 1$ to n, one can "cuckoo hash" all the keys so as to obtain a table $\boldsymbol{T}$, a vector, that stores all the keys at certain positions. Then, one can compute a vector commitment $C_{\boldsymbol{T}}$ to $\boldsymbol{T}$ and another vector commitment $C_{\boldsymbol{V}}$ to a vector $\boldsymbol{V}$ built in such a way that $\boldsymbol{V}[j]$ stores a value v if the key k associated to v is stored in $\boldsymbol{T}[j]$. Namely, each pair $(\mathsf{k}_i, \mathsf{v}_i)$ is stored in $\boldsymbol{T}$ and $\boldsymbol{V}$ at the *same* position j. By correctness of cuckoo hashing, for every k such an index j exists. In order to open the commitment $(C_{\boldsymbol{T}}, C_{\boldsymbol{V}})$ to a key k one can use cuckoo hashing to find the set of h candidate indices $(j_1, \ldots, j_h)$ where k is (potentially) stored and open $C_{\boldsymbol{T}}$ at those positions, to find the index j^* such that $\boldsymbol{T}[j^*] = \mathsf{k}$. One then also opens $C_{\boldsymbol{V}}$ to position j^* and its value v. The verifier then would run similarly: for a key-value (k, v), she runs the cuckoo hashing to find out $(j_1, \ldots, j_h)$ associated to k, verify the openings of $C_{\boldsymbol{T}}$ to $(\boldsymbol{T}[j_1], \ldots, \boldsymbol{T}[j_h])$, and the opening of $C_{\boldsymbol{V}}$ to v in the position j^* such that $\boldsymbol{T}[j^*] = \mathsf{k}$. For security it is essential that all $(j_1, \ldots, j_h$ of $(C_{\boldsymbol{T}}, C_{\boldsymbol{V}})$ are opened so that the fact that k is stored in exactly one position can be verified.

In Section 6 we give more details on other technicalities of this construction, such as how to: deal with elements in the stash, prove that a key is not committed, reduce key-binding to the position binding of the VC. Notably, this transformation is black-box, *i.e.*, it works by only invoking the algorithms of the underlying VC. This stands in contrast to, *e.g.*, Verkle tree approaches [CFM08, Kus18].

3 Preliminaries

Notation. An integer $\lambda \in \mathbb{N}$ will denote the security parameter, $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$ polynomial and negligible functions respectively. Vectors are written in bold font (*e.g.* $\boldsymbol{v}$), and given vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$, $\mathbf{cat}((\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m))$ will be the vector of concatenated vectors $\boldsymbol{v}_1 \| \ldots \| \boldsymbol{v}_m$. For any positive integer $n \in \mathbb{Z}$ we denote by $[n]$ the set of integers $\{1, \ldots, n\}$ and, more generally, by $[A, B]$ the set $\{A, \ldots, B\}$ for any $A, B \in \mathbb{Z}$, $A \leq B$. $x \overset{\$}{\leftarrow} X$ will mean that x is being uniformly sampled from a finite set X. Throughout this work “PPT" stands for Probabilistic Polynomial-Time.

3.1 Public Key Encryption

Public-Key Encryption (PKE) allows all users aware of some public information to encrypt messages that only some users aware of secret information will be able to decrypt to access these messages. It has extensively been used and developed in cryptography during the last fifty years. In short, a PKE scheme PKE consists of three algorithms:

- $\mathsf{PKE.KeyGen}(1^\lambda)$: this algorithm outputs a public key pk and a secret key sk to use in the scheme with security on λ bits;
- $\mathsf{PKE.Enc}(\mathsf{pk}, m)$: the encryption algorithm outputs a ciphertext C encrypting the message m using the public key pk;
- $\mathsf{PKE.Dec}(\mathsf{sk}, C)$: this algorithm returns the message m encrypted in the ciphertext C using the secret key sk.

3.2 Cuckoo Hashing

Cuckoo Hashing (CH)[PR04] is a technique to store a set of m elements from a large universe $\mathcal{X}$ into a linear-size data structure that allows efficient memory accesses. In our work we abstract away the properties of a family of cuckoo hashing constructions that can be used in our RBE and KVC constructions. We do this by defining the notion of *Cuckoo Hashing schemes.* Our definition is a variant of the one recently offered by Yeo [Yeo23]; in our definition, we use *deterministic* Insert algorithms.

In a nutshell, a cuckoo hashing scheme inserts n elements $x_1, \ldots, x_n \in \mathcal{X}$ in a vector $\boldsymbol{T}$ so that each element x_i can be found exactly once in $\boldsymbol{T}$, or in a stash set S. The efficient memory access comes from the fact that for a given x one can efficiently compute the k indices $i_1, \ldots, i_k$ such that $x \in \{\boldsymbol{T}[i_1], \ldots, \boldsymbol{T}[i_k]\} \cup S$. The idea of cuckoo hashing constructions is to sample k random hash functions $H_1, \ldots, H_k : \mathcal{X} \to [n]$ and use them to allocate x in one of the k indices $H_1(x), \ldots, H_k(x)$. Each construction uses a specific algorithm to search the index allocated to x, requiring to move existing elements whenever a position is going to be allocated to another element. The most efficient algorithms are local search allocation [Kho13] and random walks [FMM09, FPS13, FJ16, Wal22, Yeo23].

We define a Cuckoo Hashing scheme with the following algorithms:

Definition 1 (Cuckoo Hashing Schemes Algorithms). *A Cuckoo Hashing scheme* $\mathsf{CH} = (\mathsf{Setup}, \mathsf{Insert}, \mathsf{Lookup})$ *consists of the following algorithms:*

- $\mathsf{Setup}(1^\lambda, \mathcal{X}, n) \to (\mathsf{pp}, \boldsymbol{T}, S)$: *is a probabilistic algorithm that on input the security parameter, the space of input values* $\mathcal{X}$ *and a bound* n *on the number of insertions, outputs public parameters* pp*,* $k \geq 2$*, an empty vector* $\boldsymbol{T}$ *with* N *entries (with* N *a multiple of* k*), along with an empty stash set* S*, (denoting* $s \geq 0$ *its size, at this point,* $s = 0$*);*
- $\mathsf{Insert}(\mathsf{pp}, \boldsymbol{T}, S, x_1, \ldots, x_m) \to (\boldsymbol{T}', S')$: *is a deterministic algorithm that on input vector* $\boldsymbol{T}$ *where each non-empty component contains an element in* $\mathcal{X} \in \mathsf{pp}$*, inserts each* $x_1, \ldots, x_m \in \mathcal{X}$ *in the vector exactly once and returns the updated vector with moved elements,* $\boldsymbol{T}', S'$*.*
- $\mathsf{Lookup}(\mathsf{pp}, x) \to (i_1, \ldots, i_k)$: *is a deterministic algorithm that on input public parameters* pp *and* $x \in \mathcal{X}$*, returns* $(i_1, \ldots, i_k)$*, the candidate indices where* x *could be stored.*

Remark 1. Our Cuckoo Hashing schemes are, overall, probabilistic with the probability taken over the choice of pp. Once pp is fixed, everything is deterministic; Insert and Lookup, that take pp as input, are deterministic algorithms.

Our definition above differs from the one in [Yeo23] in the following aspects. First, we consider dynamic cuckoo hashing schemes in which one can keep inserting elements, while [Yeo23] considers the static case in which the set is hashed all at once. Second, in our notion each entry of $\boldsymbol{T}$ can store a single element, whereas [Yeo23] considers the more general case where it can store $\ell \geq 1$ elements, which occurs in some constructions.

We define correctness of cuckoo hashing by looking at the probability that either the insertion algorithm fails or, if it does not fail, an inserted element is not stored in the appropriate indices returned by Lookup. To model this notion we give two definitions. The first one is the "classical" correctness definition of cuckoo hashing that takes this probability over any choice of inputs but for a random and independent sampling of the hash functions. Intuitively this models the scenario where an adversary for correctness does not have explicit access to the hash functions, but can still choose any input set.

Definition 2 (Correctness). *A cuckoo hashing scheme* CH *is* ϵ*-**correct** if for any* n*, any set of* $m \leq n$ *items* $x_1, \ldots, x_m \in \mathcal{X}$ *such that* $x_i \neq x_j$ *for all* $i \neq j$ *and any* $\ell \in [m]$*:*

$$\Pr\left[\begin{array}{c} \boldsymbol{T}' = \bot \\ \vee\ (\boldsymbol{T}' \neq \bot\ \wedge \\ x_\ell \notin \{\boldsymbol{T}'[i_1], \ldots, \boldsymbol{T}'[i_k]\} \cup S') \end{array} : \begin{array}{r} (\mathsf{pp}, \boldsymbol{T}, S) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{X}, n) \\ (\boldsymbol{T}', S') \leftarrow \mathsf{Insert}(\mathsf{pp}, \boldsymbol{T}, S, x_1, \ldots, x_m) \\ (i_1, \ldots, i_k) \leftarrow \mathsf{Lookup}(\mathsf{pp}, x_\ell) \end{array}\right] \leq \epsilon$$

and one simply says that CH *is **correct** if it is* ϵ*-correct with* $\epsilon = \mathsf{negl}(\lambda)$*.*

Robust Cuckoo Hashing The second definition (introduced by Yeo [Yeo23]) instead considers the case of inputs that are chosen by a PPT adversary after having seen the hash functions. This models the scenario where an adversary has explicit access to the hash functions before choosing the set of elements.

Definition 3 (Robustness). *A cuckoo hashing scheme* CH *is* ϵ-***robust*** *if for any* n*, any PPT adversary* $\mathcal{A}$*:*

$$\Pr\left[\begin{array}{c} \boldsymbol{T}' = \bot \\ \vee\, (\boldsymbol{T}' \neq \bot \,\wedge \\ x_\ell \notin \{\boldsymbol{T}'[i_1],\ldots,\boldsymbol{T}'[i_k]\} \cup S') \end{array} : \begin{array}{r} (\mathsf{pp},\boldsymbol{T},S) \leftarrow \mathsf{Setup}(1^\lambda,\mathcal{X},n) \\ \{x_1,\ldots,x_m,\ell\} \leftarrow \mathcal{A}(\mathsf{pp}) \\ x_i \neq x_j \forall i \neq j \in [m] \\ (\boldsymbol{T}',S') \leftarrow \mathsf{Insert}(\mathsf{pp},\boldsymbol{T},S,x_1,\ldots,x_m) \\ (i_1,\ldots,i_k) \leftarrow \mathsf{Lookup}(\mathsf{pp},x_\ell) \end{array}\right] \leq \epsilon$$

Efficiency parameters of cuckoo hashing For our applications, the following parameters will dictate the efficiency of a cuckoo hashing scheme: k, the number of possible indices (and of hash functions); N, the size of the table $\boldsymbol{T}$; s, the size of the stash S; d, the number of changes in the table (i.e., number of evictions) after a single insertion. While in most constructions, the parameters k and N are fixed at Setup time, in some cuckoo hashing schemes the values of s and d may depend on the randomness and the choice of inputs. As in the case of correctness vs. robustness, we define s and d in the average case (i.e., for any set of inputs and for random and independent execution of Setup) or in the worst case (i..e, for adversarial choice of inputs after seeing pp).

Existing cuckoo hashing schemes The following theorem encompasses a few existing cuckoo hashing schemes.

Theorem 1. *For a security parameter* λ *and an upper bound* n*, there exist the following cuckoo hashing schemes:*

- CH_2 *where* $k = 2$*,* $N = 2kn$*, that achieves* $\mathsf{negl}(\lambda)$*-correctness, and average case* $s = \log n$*,* $d = O(1)$ *[KMW10].*
- $\mathsf{CH}_2^{(\mathsf{rob})}$ *where* $k = 2$*,* $N = 2kn$*, that achieves* $\mathsf{negl}(\lambda)$*-robustness, and worst case* $s = n$*,* $d = O(1)$ *[KMW10, Yeo23] in the Random Oracle Model.*
- $\mathsf{CH}_\lambda^{(\mathsf{rob})}$ *where* $k = \lambda$*,* $N = 2\lambda n$*, that achieves* $\mathsf{negl}(\lambda)$*-robustness, and worst case* $s = 0$*,* $d = \lambda$ *[Yeo23] in the Random Oracle Model.*

3.3 Vector Commitments

Vector commitment (VC) schemes [LY10, CF13] allow a party to compute a commitment to a vector $\boldsymbol{v}$ and later to locally open a specific position v_i. A VC guarantees that it is hard to open a commitment to two distinct values at the same position – what is called "position binding" – and should have short (i.e., polylogarithmic in $|\boldsymbol{v}|$) commitments and openings. Formally:

Definition 4 (Vector Commitment [CF13]). *A Vector Commitment (VC) scheme* $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver}$ *consists of the following algorithms:*

- $\mathsf{Setup}(1^\lambda, n) \rightarrow \mathsf{crs}$: *on input the security parameter* λ *and an integer* n *expressing the length of the vectors to be committed, returns the common reference string* crs*.*
- $\mathsf{Com}(\mathsf{crs}, \boldsymbol{v}) \rightarrow (C, \mathsf{aux})$: *on input a common reference string* crs *and a vector* $\boldsymbol{v}$*, returns a commitment* C*.*

- $\mathsf{Open}(\mathsf{crs}, \mathsf{aux}, i) \to \Lambda$: *on input an auxiliary information as produced by* Com *and a position* $i \in [n]$*, returns an opening proof* Λ.
- $\mathsf{Ver}(\mathsf{crs}, C, \Lambda, i, v) \to b$: *on input a commitment* C*, returns a bit* $b \in \{0; 1\}$ *to check whether* Λ *is a valid opening of* C *to* v *at position* i.

Correctness. VC *is* perfectly correct *if for any vector* $\boldsymbol{v}$*:*

$$\Pr\left[\mathsf{Ver}(\mathsf{crs}, C, \mathsf{Open}(\mathsf{crs}, \mathsf{aux}, i), i, v_i)) = 1 : \begin{array}{c} \mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, n) \\ (C, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{crs}, \boldsymbol{v}) \end{array}\right] = 1$$

Position binding. VC *satisfies position binding if for any PPT* $\mathcal{A}$

$$\Pr\left[\begin{array}{c} \mathsf{Ver}(\mathsf{crs}, C, \Lambda, i, v)) = 1 \\ \wedge\, \mathsf{Ver}(\mathsf{crs}, C, \Lambda, i, v')) = 1 \\ \wedge\, v \neq v' \end{array} : \begin{array}{c} \mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, n) \\ (C, i, v, \Lambda, v', \Lambda') \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] = \mathsf{negl}(\lambda)$$

Succinctness. VC *is* succinct *if for any* $\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, n)$*, any vector* $\boldsymbol{v}$*, any* $(C, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{crs}, \boldsymbol{v})$*, any* $i \in [n]$ *and* $\Lambda \leftarrow \mathsf{Open}(\mathsf{crs}, \mathsf{aux}, i)$*, the bitsize of* C *and* Λ *is polylogarithmic in* n*,* i.e.*, is bounded by a fixed polynomial* $p(\lambda, \log n)$.

In this work we use the notion of *updatable* vector commitments [CF13], which informally provides the functionality that, given a commitment C and opening Λ corresponding to a vector $\boldsymbol{v}$, one can update them into values C' and Λ' corresponding to a vector $\boldsymbol{v}'$. Notably, this update should be efficient, i.e., in time proportional to the number of different positions in $\boldsymbol{v}$ and $\boldsymbol{v}'$, and thus faster than recomputing them from scratch. More formally:

Definition 5 (Updatable VCs [CF13]). *A vector commitment scheme* VC *is updatable if there are two algorithms* $(\mathsf{ComUpdate}, \mathsf{ProofUpdate})$ *such that:*

- $\mathsf{ComUpdate}(\mathsf{crs}, C, i, v, v') \to C'$: *on input a commitment* C*, a position* i *and two values* v*,* v'*, outputs an updated commitment* C'.
- $\mathsf{ProofUpdate}(\mathsf{crs}, \Lambda, i, v, v') \to \Lambda'$: *on input an opening proof* Λ *(for some position* j*), a position* i *and two values* v*,* v'*, returns an updated opening* Λ'.

Correctness. *An updatable VC is perfectly correct if for honestly generated* $\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, n)$*, any vector* $\boldsymbol{v}$*, initial commitment* $(C, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{crs}, \boldsymbol{v})$*, position* $i \in [n]$*,* $\Lambda \leftarrow \mathsf{Open}(\mathsf{crs}, \mathsf{aux}, i)$*, and any sequence of valid updates* $\{(i_k, v_{i_k}, v'_{i_k})\}_{k\in[m]}$ *that result into a vector* $\boldsymbol{v}^*$*, commitment* C^* *and opening* Λ^**,* $\mathsf{Ver}(\mathsf{crs}, C^*, \Lambda^*, i, v^*_i)) = 1$ *holds with probability 1.*

Efficiency. *An updatable VC is* efficient *if its algorithms* $\mathsf{ComUpdate}$ *and* $\mathsf{ProofUpdate}$ *run in polylogarithmic time given polylogarithmic inputs.*

3.4 Registration-based Encryption

We recall the original defintion of Registration-Based Encryption [GHMR18] with the modification of [GKMR22] that allows for a structured common reference string crs and a bound n on the number of users that can be registered. In case a crs is not involved or the scheme allows for an unbounded number of registered users we consider $\mathsf{crs} = \emptyset$ and $N = \infty$ respectively.

For completeness we recall how an RBE system evolves: At the beginning a one-time setup algorithm generates the common reference string. Then there are two types of parties: the Key Curator

(KC) and the users, each represented by an identity id from a pre-specified identity space $\mathcal{ID}$. The KC is completely transparent (and deterministic) and her role is solely to ease the computational burden of each user. Each user, upon entering the system generates their own public-secret key-pair (pk, sk) and registers their public key with the Key Curator, who computes the updated public parameters pp after the new registration. Anyone can encrypt a message $m \in \mathcal{M}$ for an identity id by having access to the crs and the current pp (without knowing the corresponding pk of id). Finally the identity can decrypt the ciphertext ct using their secret key sk and an update information u that is computed by the KC and given to the user. [8]

We further enhance the RBE definition with the functionality of deletion of users from the system. We call an RBE that supports this functionality an *RBE with deletions*. Below is the formal definition.

Definition 6 (Registration-Based Encryption (RBE) $\boxed{\textbf{with deletions}}$). *A registration-based encryption scheme with identity space $\mathcal{ID}$ and message space space $\mathcal{M}$ consists of six/$\boxed{\textit{seven}}$ PPT algorithms* (Setup, Gen, Reg, $\boxed{\mathsf{Del},}$ Enc, Upd, Dec) *working as follows.*

- $\mathsf{Setup}(1^\lambda, N) \rightarrow \mathsf{crs}$: *On input the security parameter λ and a positive integer N indicating the maximum number of users that can be registered, the randomized setup algorithm samples a common reference string* crs*.*
- $\mathsf{Gen}(\mathsf{crs}, \mathsf{id}) \rightarrow (\mathsf{pk}, \mathsf{sk})$: *On input the common reference string* crs *and an identity* id*, the randomized algorithm key generation algorithm outputs a pair of public and secret keys* (pk, sk)*.*
- $\mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk}) \rightarrow \mathsf{pp}'$: *On input the common reference string* crs*, the current public parameters* pp*, an identity* $\mathsf{id} \in \mathcal{ID}$*, and a public key* pk*, the deterministic registration algorithm outputs the new public parameters* pp'*. The* Reg *algorithm has* read *and* write *oracle access to the auxiliary information* aux *which is updated into* aux' *during registration. (The system is initialized with public parameters* pp *and auxiliary information* aux *set to* $\perp$*.)*
- $\mathsf{Del}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}) \rightarrow \mathsf{pp}'$: *On input the common reference string* crs*, the current public parameters* pp*, and an identity* $\mathsf{id} \in \mathcal{ID}$ *the deterministic registration algorithm outputs the new public parameters* pp' *or* $\perp$ *if* id *was not registered before. The* Del *algorithm has* read *and* write *oracle access to the auxiliary information* aux *which is updated into* aux' *during the process.*
- $\mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, m) \rightarrow \mathsf{ct}$: *On input the common reference string* crs*, the current public parameters* pp*, a recipient identity* $\mathsf{id} \in \mathcal{ID}$ *and a message* $m \in \mathcal{M}$*, the randomized encryption algorithm outputs a ciphertext* ct*.*
- $\mathsf{Upd}^{[\mathsf{aux}]}(\mathsf{pp}, \mathsf{id}) \rightarrow \mathsf{u}$: *On input the current public parameters* pp *and a registered identity* id*, the deterministic update algorithm outputs an update information* u *that can help* id *to decrypt its messages. It has* read only *oracle access to* aux*.*
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{u}, \mathsf{ct}) \rightarrow m$: *On input the secret* sk*, the (current) update information* u *and a ciphertext* ct*, the deterministic decryption algorithm outputs a message* $m \in \{0,1\}^*$ *or in* $\{\perp, \mathsf{GetUpd}\}$*. The symbol* $\perp$ *indicates a syntax error while* GetUpd *indicates that more recent update information might be needed for decryption.*

Below is the formal definition of completeness and the efficiency requirements of RBE as described in [GHMR18] with two modifications: (1) we additionally take into account deletions, (2) and define a computational version, *i.e.* with a PPT adversary instead of an unbounded one.

Definition 7 (Completeness, compactness, and efficiency of RBE). *For any interactive PPT adversary $\mathcal{A}$, consider the following game $\mathsf{Comp}_{\mathcal{A}}(\lambda)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.*

[8] The update information does not have to be secret and is only computed by KC and not by the user for efficiency.

1. **Initialization.** *$\mathcal{C}$ sets* $\mathsf{pp} \leftarrow \perp$, $\mathsf{aux} \leftarrow \perp$, $\mathsf{u} \leftarrow \perp$, $\mathcal{D} \leftarrow \emptyset$, $\mathsf{id}^* \leftarrow \perp$, $t \leftarrow 0$, $\hat{N} \leftarrow 0$, $\hat{M} \leftarrow 0$ *and* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, N)$, *and sends the sampled* crs *to* $\mathcal{A}$.
2. *Until $\mathcal{A}$ continues, proceed as follows. At every iteration, $\mathcal{A}$ chooses exactly one of the actions below to be performed.*
   (a) **Registering new (non-target) identity.** *If $|\mathcal{D}| = N$ skip this step. $\mathcal{A}$ sends some* $\mathsf{id} \notin \mathcal{D}$ *and* pk *in the support of the* $\mathsf{Gen}(\mathsf{crs})$ *algorithm, to $\mathcal{C}$. $\mathcal{C}$ registers* $(\mathsf{id}, \mathsf{pk})$ *by letting* $\mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk})$ *and* $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathsf{id}\}$, $\hat{N} \leftarrow \hat{N} + 1$.
   (b) **Deleting existing (non-target) identity.** *$\mathcal{A}$ sends some* $\mathsf{id} \in \mathcal{D}$ *to $\mathcal{C}$. $\mathcal{C}$ un-registers* id *by letting* $\mathsf{pp} \leftarrow \mathsf{Del}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id})$ *and* $\mathcal{D} \leftarrow \mathcal{D} \setminus \{\mathsf{id}\}$, $\hat{M} \leftarrow \hat{M} + 1$.
   (c) **Registering the target identity.** *If* $\mathsf{id}^* \neq \perp$ *or* $|\mathcal{D}| = N$ *, skip this step. Otherwise, $\mathcal{A}$ sends some* $\mathsf{id}^* \notin \mathcal{D}$ *to $\mathcal{C}$. $\mathcal{C}$ then samples* $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{Gen}(\mathsf{crs}, \mathsf{id}^*)$, *updates* $\mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*)$ *and* $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathsf{id}^*\}$, $\hat{N} \leftarrow \hat{N} + 1$, *and sends* pk^* *to* $\mathcal{A}$.
   (d) **Deleting the target identity.** *If* $\mathsf{id}^* \notin \mathcal{D}$, *skip this step. Otherwise, $\mathcal{C}$ updates* $\mathsf{pp} \leftarrow \mathsf{Del}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*)$ *and* $\mathcal{D} \leftarrow \mathcal{D} \setminus \{\mathsf{id}^*\}$, $\hat{M} \leftarrow \hat{M} + 1$.
   (e) **Encrypting for the target identity.** *If* $\mathsf{id}^* = \perp$, *skip this step. Otherwise, $\mathcal{C}$ sets $t \leftarrow t+1$. $\mathcal{A}$ sends some $m_t \in \mathcal{M}$ to $\mathcal{C}$ who sends back a corresponding ciphertext* $\mathsf{ct}_t \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, m_t)$ *to* $\mathcal{A}$.
   (f) **Decryption by target identity.** *$\mathcal{A}$ sends a $j \in [t]$ to $\mathcal{C}$. $\mathcal{C}$ then lets* $m'_j = \mathsf{Dec}(\mathsf{sk}^*, \mathsf{u}, \mathsf{ct}_j)$. *If* $m'_j = \mathsf{GetUpd}$, *then $\mathcal{C}$ obtains the update* $\mathsf{u}^* = \mathsf{Upd}^{[\mathsf{aux}]}(\mathsf{pp}, \mathsf{id}^*)$ *and then lets* $m'_j = \mathsf{Dec}(\mathsf{sk}^*, \mathsf{u}^*, \mathsf{ct}_j)$.
3. *The adversary $\mathcal{A}$ wins the game if there is some $j \in [t]$ for which $m'_j \neq m_j$.*

Let $Q \in$ $\mathsf{poly}(\lambda)$ be an upper bound on the number of queries issued by $\mathcal{A}$. Let $\mathcal{D}_q$ be the set of identities after the q-th query. We require the following properties to hold for any PPT adversary $\mathcal{A}$.

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

Compactness of public parameters and updates. *For all queries $q \in [Q]$, let* pp_q *be the public parameters after the q-th query. Then* $|\mathsf{pp}_q|$ *is sublinear in* $|\mathcal{D}_q|$. *Moreover, for all* $\mathsf{id} \in \mathcal{D}$, *the size of the corresponding update* $|\mathsf{u}_q|$ *is also sublinear in* $|\mathcal{D}_q|$.

Efficiency of the number of updates. *The total number of invocations of* Upd *for identity* id^* *in Step 2(f) of the game* $\mathsf{Comp}_{\mathcal{A}}(\lambda)$ *is sublinear in* $\hat{N}$.

Remark 2 (Efficiency of Registration and Updates). The initial work of Garg et al. [GHMR18] considers a fourth strigent efficiency requirement, that the running times of Reg, Del and Upd should be $\mathsf{polylog}(N)$. Constructions using iO [GHMR18] and garbled circuits [GHM$^+$19] satisfy this, however to date there is no black-box construction with this property. Additionally our concrete compilers do not (asymptotically) affect the running times of Reg, Del and Upd. Therefore, to avoid overwhelming the reader we do not consider this property.

For the security of RBE, the adversary can control all users except for a target identity id^* of their choice. Then we demand ciphertext indistinguishability for encrypted messages under this id^*. Below is the formal security definition taken almost verbatim from [GHMR18], where we additionally consider deletions.

Definition 8 (Security of RBE). *For any interactive PPT adversary $\mathcal{A}$, consider the following game* $\mathsf{Sec}_{\mathcal{A}}(\lambda)$ *between $\mathcal{A}$ and a challenger $\mathcal{C}$.*

1. **Initialization.** *$\mathcal{C}$ sets* $\mathsf{pp} = \perp$, $\mathsf{aux} = \perp$, $\mathcal{D} = \emptyset$, $\mathsf{id}^* = \perp$, $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ *and sends the sampled* crs *to* $\mathcal{A}$.

2. *Until $\mathcal{A}$ continues (which is at most $\mathsf{poly}(\lambda)$ steps), proceed as follows. At every iteration, $\mathcal{A}$ chooses exactly one of the actions below to be performed.*
   (a) **Registering new (non-target) identity**. *$\mathcal{A}$ sends some $\mathsf{id} \notin \mathcal{D}$ and pk to $\mathcal{C}$. $\mathcal{C}$ registers $(\mathsf{id}, \mathsf{pk})$ by letting $\mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk})$ and $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathsf{id}\}$.*
   (b) **Deleting an existing (non-target) identity.** *$\mathcal{A}$ sends some $\mathsf{id} \in \mathcal{D}$ to $\mathcal{C}$. $\mathcal{C}$ un-registers id by letting $\mathsf{pp} \leftarrow \mathsf{Del}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id})$ and $\mathcal{D} \leftarrow \mathcal{D} \setminus \{\mathsf{id}\}$.*
   (c) **Registering the target identity.** *If $\mathsf{id}^* \neq \bot$, skip this step. Otherwise, $\mathcal{A}$ sends an $\mathsf{id}^* \notin \mathcal{D}$ to $\mathcal{C}$. $\mathcal{C}$ then samples $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{Gen}(\mathsf{crs}, \mathsf{id}^*)$, updates $\mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*)$, $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathsf{id}^*\}$, and sends pk^* to $\mathcal{A}$.*
   (d) **Deleting the target identity.** *If $\mathsf{id}^* = \bot$, skip this step. Otherwise, $\mathcal{C}$ updates $\mathsf{pp} \leftarrow \mathsf{Del}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*)$ and $\mathcal{D} \leftarrow \mathcal{D} \setminus \{\mathsf{id}^*\}$, $\mathsf{id}^* \leftarrow \bot$.*
3. **Encrypting for the target identity.** *$\mathcal{A}$ sends some $\mathsf{id} \notin \mathcal{D} \setminus \{\mathsf{id}^*\}$ and two messages (m_0, m_1) and $\mathcal{C}$ generates $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, m_b)$, where $b \leftarrow \{0, 1\}$ is a random bit, and sends ct to $\mathcal{A}$.*
4. *The adversary $\mathcal{A}$ outputs a bit b' and wins the game if $b = b'$.*

We call an RBE scheme secure if there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all PPT adversaries $\mathcal{A}$ it holds that $\Pr[\mathcal{A}\, wins\, \mathsf{Sec}_{\mathcal{A}}(\lambda)] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$.

Remark 3 (RBE with deletions–constructions). Although the notion of deletions has not been previously formally in the context of RBE, all known RBE constructions [GHMR18, GHM+19, GV20, GKMR22, DKL+23]can be enhanced in a straightforward way with this functionality.

Laconic Encryption Döttling et al. [DKL+23] introduced the notion of Laconic Encryption which is essentially the same as RBE but dropping the 'Efficiency of the number of updates' requirement. They additionally showed a generic transformation from any Laconic Encryption scheme to an RBE scheme with Efficient updates, generalizing the tranformations of Garg et al. [GHMR18] and Glaeser et al. [GKMR22]. The same transformation was presented in the context of Registered Attribute-Based Encryption by Hohenberger et et al. [HLWW23].

We summarize the tranformation in the following theorem, slightly extending it to include deletions.

Theorem 2 ([DKL+23]). *Assume any Laconic Encryption scheme LE with deletions, with worst-case:*

$$\textbf{\textit{Compactness:}}\ |\mathsf{pp}|, |\mathsf{u}| = o(N), \qquad \textbf{\textit{Ciphertext size:}}\ |\mathsf{ct}|,$$

then there exists an RBE scheme (without deletions) with worst-case:

$$\textbf{\textit{Compactness:}}\ |\mathsf{pp}| \log(\hat{N}), |\mathsf{u}|, \qquad \textbf{\textit{Ciphertext size:}}\ |\mathsf{ct}| \log(\hat{N}),$$
$$\textbf{\textit{Number of updates:}}\ \log(\hat{N}).$$

For conciseness in the rest of this work we will consider Laconic Encryption, and then apply the above Theorem 2 to achieve a fully efficient RBE.

4 RBE with Unbounded Identity Space from CH

Here we show our compiler that boosts any RBE scheme with small identity space to an RBE with large identity space. On the core of compiler are Cuckoo Hashing and the notion of Witness Encryption for Vector Commitments that we define next (Section 4.1). For the intuition of the transformation we refer to the technical overview of Section 2.

4.1 Witness Encryption for Vector Commitments

As mentioned in Section 2, a building block of our RBE construction is a specialized witness encryption scheme. We call this primitive VCWE and intuitively it works as follows. One encrypts a message m with respect to a statement consisting of a commitment C, a position i and a value v, and decryption is achieved by using a valid opening that shows that v is indeed the value at position i in the vector committed in C. We notice that VCWE can be seen as a special case of the notion of 'WE for functional commitments' recently proposed by Campanelli, Fiore and Khoshakhlagh [CFK22].

Although VCWE has a witness encryption flavor, its semantic security notion is weaker than standard WE. Notably, in WE semantic security for false statements should hold statistically, whereas in VCWE is computational. Also, we define semantic security in such a way that the experiment is falsifiable and can check whether a statement is true or false. For details, see the definition provided hereafter.

Definition 9 (Witness Encryption for Vector Commitments). *Let* $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ *be a vector commitment scheme. A witness encryption scheme with respect to* VC*,* VCWE *for short, consist of PPT algorithms* $(\mathsf{Enc}, \mathsf{Dec})$*:*

- $\mathsf{Enc}(\mathsf{crs}, C, i, v, m) \to \mathsf{ct}$: *on input a vector commitment common reference string* crs*, a commitment C, a position i, a value v and a message m outputs a witness-encryption of m under the statement (C, i, v).*
- $\mathsf{Dec}(\mathsf{crs}, \Lambda, \mathsf{ct}) \to m^*$: *on input a vector commitment common reference string* crs*, a witness Λ and a ciphertext m outputs a decryption message m^*.*

Furthermore, VCWE *should satisfy the following properties:*

Correctness. *For any security parameter $\lambda \in \mathbb{N}$, any $N = \mathsf{poly}(\lambda)$, any $\boldsymbol{v} = (v_1, \ldots, v_N)$ in the domain of* VC*, $i \in [N]$, $m \in \mathcal{M}$,* $\mathsf{crs} \leftarrow \mathsf{VC.Setup}(1^\lambda, N)$*, $C \leftarrow \mathsf{VC.Com}(\mathsf{crs}, \boldsymbol{v})$, and $\Lambda \leftarrow$* $\mathsf{Open}(\mathsf{crs}, C, \boldsymbol{v}, i)$*:*

$$\Pr[\mathsf{Dec}(\mathsf{crs}, \Lambda, \mathsf{ct}) = m : \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, C, i, v, m)] = 1$$

Semantic Security. *For any security parameter $\lambda \in \mathbb{N}$, $N = \mathsf{poly}(\lambda)$ and any PPT adversary $\mathcal{A}$:*

$$\left| \Pr\left[b' = b : \begin{array}{l} \mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, N), \\ (\boldsymbol{v}, i, v, m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{crs}), \\ C \leftarrow \mathsf{Com}(\mathsf{crs}, \boldsymbol{v}), b \leftarrow\!\!\$\ \{0,1\}, \\ \textbf{if } \boldsymbol{v}[i] = v \textbf{ then } \mathsf{ct} \leftarrow \bot \\ \textbf{else } \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, C, i, v, m_b), \\ b' \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{ct}) \end{array} \right] - \frac{1}{2} \right| = \mathsf{negl}(\lambda)$$

4.2 RBE with Unbounded Identity Space

Assume any RBE scheme with deletions, $\widetilde{\mathsf{RBE}} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Reg}, \mathsf{Enc}, \mathsf{Upd}, = \mathsf{Dec})$ that supports bounded identities, i.e. $\log(|\mathcal{ID}|) < 2\lambda$. We show a transformation that boosts it to an RBE scheme that supports unbounded identities $\mathcal{ID} = \{0,1\}^{2\lambda}$.[9]

For presentational convenience we show the compiler for Laconic Encryption, that is from $\widetilde{\mathsf{LE}}$ with $|\mathcal{ID}| < 2\lambda$ to LE with $\mathcal{ID} = \{0,1\}^{2\lambda}$. Recall from Section 3.4 $\widetilde{\mathsf{LE}}$ is essentially $\widetilde{\mathsf{RBE}}$ without

[9] We consider the identity space $\{0,1\}^{2\lambda}$ virtually unbounded since one can always use a collision-resistant hash function $H : \{0,1\}^* \to \{0,1\}^{2\lambda}$ to support unbounded identities.

the efficiency on the number of updates requirement. Then we make use of Theorem 2 with which we can obtain an RBE with a logarithmic number of updates.

From now on we will be using small n as an upper bound on the number of RBE users and capital $N > n$ as the resulting number of entries of the Cuckoo Hashing table (see Section 3.2).

Building Blocks. For our compiler we need the following primitives:

1. A Cuckoo Hashing scheme $\mathsf{CH} = (\mathsf{Setup}, \mathsf{Insert}, \mathsf{Lookup})$.
2. A Witness Encryption scheme $\mathsf{VCWE} = (\mathsf{Enc}, \mathsf{Dec})$ w.r.t. a Vector Commitment scheme $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Com},\ \ \mathsf{Open}, \mathsf{Ver})$.
3. A Public Key Encryption scheme $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.
4. A Secret Sharing scheme $\mathsf{Sh} = (\mathsf{Share}, \mathsf{Rec})$.

We note that RBE trivially implies a PKE by registering a single user. Furthermore, any RBE scheme implies a (weakly position binding) VC scheme. For the secret sharing scheme, if the message space of the RBE is a group then there exist simple information-theoretic constructions. Therefore, in essence we only assume a Cuckoo Hashing and a Witness Encryption scheme for the underlying Vector Commitment. However, one may desire to instantiate the primitives with different constructions therefore we explicitly mention them distinctly.

Construction. We denote the initial LE scheme as $\widetilde{\mathsf{LE}}$. Throughout the description of our construction below, for every identity id we use the notation $\mathsf{id}^{(\eta)}$ for η'th coordinate of $\mathsf{Lookup}(\mathsf{pp}, \mathsf{id})$, i.e. by defintion $(\mathsf{id}^{(1)}, \ldots, \mathsf{id}^{(k)}) = \mathsf{Lookup}(\mathsf{pp}, \mathsf{id})$.

Our compiler yields a Laconic Encryption scheme RBE that works as follows:

- $\mathsf{Setup}(1^\lambda, n) \to \mathsf{crs}$:
  1. $\mathsf{chpp} \leftarrow \mathsf{CH.Setup}(1^\lambda, n)$,
  2. $\widetilde{\mathsf{crs}} \leftarrow \widetilde{\mathsf{LE}}.\mathsf{Setup}(1^\lambda, n)$,
  3. $\mathsf{vccrs} \leftarrow \mathsf{VC.Setup}(1^\lambda, N)$.

  $$\mathsf{crs} = (\widetilde{\mathsf{crs}}, \mathsf{chpp}, \mathsf{vccrs})$$
- $\mathsf{Gen}(\mathsf{crs}, \mathsf{id}) \to (\mathsf{pk}, \mathsf{sk})$:
  1. $(\mathsf{pk}^{(\eta)}, \mathsf{sk}^{(\eta)}) \leftarrow \widetilde{\mathsf{LE}}.\mathsf{Gen}(\widetilde{\mathsf{crs}}, \mathsf{id}^{(\eta)})$ for each $\eta \in [k]$,
  2. $(\mathsf{pk}^{(k+1)}, \mathsf{sk}^{(k+1)}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

  $$\mathsf{pk} = \left((\mathsf{pk}^{(\eta)})_{\eta \in [k]}, \mathsf{pk}^{(k+1)}\right), \quad \mathsf{sk} = \left((\mathsf{sk}^{(\eta)})_{\eta \in [k]}, \mathsf{sk}^{(k+1)}\right)$$
- $\mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk}) \to \mathsf{pp}'$:

  Auxiliary information. The auxiliary information of the Key Curator consists of $\mathsf{aux} := (\widetilde{\mathsf{aux}}, \boldsymbol{I})$, where $\boldsymbol{I} := (\mathsf{id}_1, \ldots, \mathsf{id}_N)$ is the vector of the (previously) registered identities in the corresponding positions (if no identity is registered in position j then $\mathsf{id}_j = 0$).

  Public Parameters. The public parameters of the system consist of $\mathsf{pp} := (\widetilde{\mathsf{pp}}, D, S)$, where D is the current vector commitment to the identities $\boldsymbol{I} := (\mathsf{id}_1, \ldots, \mathsf{id}_N)$ and $S = \{(\mathsf{id}_{N+1}, \mathsf{pk}_{N+1}), \ldots, (\mathsf{id}_{N+s}, \mathsf{pk}_{N+s})\}$ is the set of identities that are stored in the stash.

  Insert id. Runs $(\boldsymbol{I}', S') \leftarrow \mathsf{CH.Insert}(\mathsf{chpp}, \boldsymbol{I}, S, \mathsf{id})$ and if $\mathsf{CH.Insert}$ failed outputs $\perp$. Otherwise for every identity $\bar{\mathsf{id}}$ that was moved, i.e. its position in $(\boldsymbol{I}', S')$ and $(\boldsymbol{I}, S)$ differs, re-registers it as $\bar{\mathsf{id}}^{(\eta^\dagger)}$ (η^* indicates the hash function with which id was placed before, in $\boldsymbol{I}$, and $\eta^\dagger$ the one after, in $\boldsymbol{I}'$). For every $\bar{\mathsf{id}}$ that was moved:

1. $\boldsymbol{I}[\bar{\mathsf{id}}^{(\eta^*)}] \leftarrow 0$,
2. $\widetilde{\mathsf{pp}} \leftarrow \widetilde{\mathsf{LE}}.\mathsf{Del}^{[\widetilde{\mathsf{aux}}]}(\widetilde{\mathsf{crs}}, \widetilde{\mathsf{pp}}, \bar{\mathsf{id}}^{(\eta^{(*)})}, \bar{\mathsf{pk}})$.

Then for every $\bar{\mathsf{id}}$ that was moved to $\boldsymbol{I}'$ (including id that was freshly inserted):

3. $\boldsymbol{I}[\bar{\mathsf{id}}^{(\eta^\dagger)}] \leftarrow \bar{\mathsf{id}}$,
4. $\widetilde{\mathsf{pp}} \leftarrow \widetilde{\mathsf{LE}}.\mathsf{Reg}^{[\widetilde{\mathsf{aux}}]}(\widetilde{\mathsf{crs}}, \widetilde{\mathsf{pp}}, \bar{\mathsf{id}}^{(\eta^{(\dagger)})}, \bar{\mathsf{pk}})$.

For every $\bar{\mathsf{id}}$ that was moved to the stash:

5. $S \leftarrow S \cup \{(\bar{\mathsf{id}}, \bar{\mathsf{pk}})\}$.

Finally, computes:

6. $D = \mathsf{VC.Com}(\mathsf{vccrs}, \boldsymbol{I})$. [10]

Output.

$$\mathsf{pp}' = (\widetilde{\mathsf{pp}}, D, S)$$

- $\mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, m) \rightarrow \mathsf{ct}$: Searches the stash S and then, according to whether $(\mathsf{id}, \cdot) \in S$, proceeds as follows:

  Identity in the table. If $(\mathsf{id}, \cdot) \notin S$ then for each $\eta \in [k]$:

  1. $(m_1^{(\eta)}, m_2^{(\eta)}) \leftarrow \mathsf{Sh.Share}(m)$,
  2. $\mathsf{ct}_1^{(\eta)} \leftarrow \widetilde{\mathsf{LE}}.\mathsf{Enc}(\widetilde{\mathsf{crs}}, \widetilde{\mathsf{pp}}, \mathsf{id}^{(\eta)}, m_1^{(\eta)})$,
  3. $\mathsf{ct}_2^{(\eta)} \leftarrow \mathsf{VCWE.Enc}(\mathsf{crs}, D, \mathsf{id}^{(\eta)}, \mathsf{id}, m_2^{(\eta)})$.

  The final ciphertext is:

  $$\mathsf{ct} = \Big((\mathsf{ct}_1^{(1)}, \mathsf{ct}_2^{(1)}), \ldots, (\mathsf{ct}_1^{(k)}, \mathsf{ct}_2^{(k)})\Big)$$

  Identity in the stash. If $(\mathsf{id}, \mathsf{pk}) \in S$ for some pk then:

  $$\mathsf{ct} = \mathsf{PKE.Enc}(\mathsf{pk}^{(k+1)}, m)$$

- $\mathsf{Upd}^{[\mathsf{aux}]}(\mathsf{pp}, \mathsf{id}) \rightarrow \mathsf{u}$: Finds the $\eta^* \in [k]$ such that $\boldsymbol{I}[\mathsf{id}^{(\eta^*)}] = \mathsf{id}$. If such η^* does not exist outputs $\mathsf{u} = \mathsf{Stash}$. Otherwise computes:

  1. $\widetilde{\mathsf{u}} \leftarrow \widetilde{\mathsf{LE}}.\mathsf{Upd}^{[\widetilde{\mathsf{aux}}]}(\widetilde{\mathsf{pp}}, \mathsf{id}^{(\eta^*)})$,
  2. $\Psi_{\mathsf{id}^{(\eta^*)}} \leftarrow \mathsf{VC.Open}(\mathsf{vccrs}, \boldsymbol{I}, \mathsf{id}^{(\eta^*)})$

  and outputs:

  $$\mathsf{u} = (\tilde{\mathsf{u}}, \Psi_{\mathsf{id}^{(\eta^*)}}, \eta^*)$$

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{u}, \mathsf{ct}) \rightarrow m$ or GetUpd :

  Identity in the table. If the ciphertext is an LE ciphertext:

  1. $m_1^* \leftarrow \widetilde{\mathsf{LE}}.\mathsf{Dec}(\mathsf{sk}^{(\eta^*)}, \tilde{u}, \mathsf{ct}_1^{(\eta^*)})$,
  2. $m_2^* \leftarrow \mathsf{VCWE.Dec}(\mathsf{vccrs}, \Psi, \mathsf{ct}_2^{(\eta^*)})$.

  $$m^* = \mathsf{Sh.Rec}(m_1^*, m_2^*)$$

  Identity in the Stash. If ct is a PKE ciphertext then

  $$m^* = \mathsf{PKE.Dec}(\mathsf{sk}^{(k+1)}, \mathsf{ct})$$

[10] In case the VC is updatable, the updated D can computed efficiently without having to recompute it from scratch. For simplicity we do not make this explicit in the construction.

4.3 Completeness, Security and Efficiency

Completeness. Directly follows from completeness of $\widetilde{\mathsf{LE}}$ and correcntess of VC, VCWE and Sh, as long as id is always either in the table or in the stash thoughout the lifetime of the system. That is, we desire that $\mathsf{Insert}(\boldsymbol{I}, S, \mathsf{id})$ never outputs $\perp$ for any id during the registration algorithm, even if the adversary chooses the identities that register. This is guaranteed by the $\mathsf{negl}(\lambda)$-robustness property of Cuckoo Hashing (see Section 3.2).

Theorem 3 (Completeness). *If* CH *is a* $\mathsf{negl}(\lambda)$*-robust Cuckoo Hashing scheme,* VC*,* VCWE*,* PKE*,* Sh *are correct and* $\widetilde{\mathsf{LE}}$ *is a complete Laconic Encryption with deletions then the* LE *scheme presented above is complete against any PPT adversary.*

Security. The security of our LE follows by the security of the underying building blocks. If the target identity turns out to be in the stash, then the the ciphertext is simply a PKE one, and therefore we rely on the security of PKE.

If not, then the ciphertext has k components, one for each possible position of id^*, *i.e.* the first is w.r.t. position $\mathsf{id}^{(1)}$, the second w.r.t position $\mathsf{id}^{(2)}$, etc. Each component consists of a VCWE ciphertext and a $\widetilde{\mathsf{LE}}$ ciphertext. For the positions $i \in [k]$ such that eventually id^* is in fact not registered we can rely on the fact that $\boldsymbol{I}[i] \neq \mathsf{id}^*$ which allows to argue that the VCWE gives us indistinguishability for one of the two components. On the other hand, for the position(s) $i \in [k]$ such that id^* is registered one can use $\mathsf{id}^{*(i)}$ as a target for the $\widetilde{\mathsf{LE}}$ security, which gives indistinguishability.

In both cases one component is indistinguishable for the adversary, therefore one of the two shares of the secret sharing is “hidden” from the adversary. Thus the privacy of the secret sharing scheme gives us security. Below is the formal security theorem

Theorem 4 (Security). *If* $\widetilde{\mathsf{LE}}$ *is secure,* VCWE *w.r.t* VC *is (VCWE) semantically secure,* PKE *is (PKE) semantically secure and* Sh *is* $(2, 2)$ *private then the* LE *scheme presented above is a secure LE scheme.*

Proof. We use the following hybrid argument. We always consider stateful adversaries. Let Hybrid_0 be the initial Security of RBE game:

```
$\mathsf{Hybrid}_0$
1. $\mathsf{pp} \leftarrow \bot;\ \mathsf{aux} \leftarrow \bot;\ \mathcal{D} \leftarrow \emptyset;\ \mathsf{id}^* \leftarrow \bot;\ \mathsf{pk}^* \leftarrow \bot;\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$
2. for $i = 1$ to $Q$ do one of the following:
  $(a)$ $(\mathsf{id}, \mathsf{pk}) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*)$;
       if $\mathsf{id} \notin \mathcal{D}$ then update $\mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk});\ \mathcal{D} \leftarrow \mathcal{D} \cup \{\mathsf{id}\}$
  $(b)$ $\mathsf{id} \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*)$;
       if $\mathsf{id} \in \mathcal{D}$ then $\mathsf{pp} \leftarrow \mathsf{Del}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id});\ \mathcal{D} \leftarrow \mathcal{D} \setminus \{\mathsf{id}\}$
  $(c)$ if $\mathsf{id}^* = \bot$ then
         $\mathsf{id}^* \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*);\ (\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{Gen}(\mathsf{crs}, \mathsf{id}^*)$;
         $\mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*);\ \mathcal{D} \leftarrow \mathcal{D} \cup \{\mathsf{id}^*\}$
  $(d)$ if $\mathsf{id}^* \neq \bot$ then
         $\mathsf{pp} \leftarrow \mathsf{Del}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*);\ \mathcal{D} \leftarrow \mathcal{D} \setminus \{\mathsf{id}^*\};\ \mathsf{id}^* \leftarrow \bot$
3. $(\mathsf{id}, m_0, m_1) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*);\ b \leftarrow\!\!\$\ \{0, 1\}$;
     if $\mathsf{id}^* \notin S$ then
       for each $\eta \in [k]$ :
         $(m_{b,1}^{(\eta)}, m_{b,2}^{(\eta)}) \leftarrow \mathsf{Sh.Share}(m_b);\ \mathsf{ct}_1^{(\eta)} \leftarrow \widetilde{\mathsf{LE}}.\mathsf{Enc}(\widetilde{\mathsf{crs}}, \widetilde{\mathsf{pp}}, \mathsf{id}^{*(\eta)}, m_{b,1}^{(\eta)})$;
         $\mathsf{ct}_2^{(\eta)} \leftarrow \mathsf{VCWE.Enc}(\mathsf{crs}, D, \mathsf{id}^{*(\eta)}, \mathsf{id}, m_{b,2}^{(\eta)})$;
       $\mathsf{ct} \leftarrow (\mathsf{ct}_1^{(\eta)}, \mathsf{ct}_2^{(\eta)})_{\eta \in [k]}$
     else
       $\mathsf{ct} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{id}*}^{(k+1)}, m_b)$
4. $b' \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*, \mathsf{ct})$
Output 1 if $b = b'$
```

In the next hybrid 1 we substitute ct in case the target identity is in the stash, $\mathsf{id}^* \in S$ with $\mathsf{ct} = \mathsf{PKE.Enc}(\mathsf{pk}^{(k+1)}, 0)$

```
Hybrid$_1$
1. $\mathsf{pp} \leftarrow \bot;\ \mathsf{aux} \leftarrow \bot;\ \mathcal{D} \leftarrow \emptyset;\ \mathsf{id}^* \leftarrow \bot;\ \mathsf{pk}^* \leftarrow \bot;\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$
2. for $i = 1$ to $Q$ do one of the following:
  (a) $(\mathsf{id}, \mathsf{pk}) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*)$;
      if $\mathsf{id} \notin \mathcal{D}$ then $\mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk});\ \mathcal{D} \leftarrow \mathcal{D} \cup \{\mathsf{id}\}$
  (b) $\mathsf{id} \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*)$;
      if $\mathsf{id} \in \mathcal{D}$ then $\mathsf{pp} \leftarrow \mathsf{Del}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id});\ \mathcal{D} \leftarrow \mathcal{D} \setminus \{\mathsf{id}\}$
  (c) if $\mathsf{id}^* = \bot$ then
        $\mathsf{id}^* \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*);\ (\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{Gen}(\mathsf{crs}, \mathsf{id}^*)$;
        $\mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*);\ \mathcal{D} \leftarrow \mathcal{D} \cup \{\mathsf{id}^*\}$
  (d) if $\mathsf{id}^* \neq \bot$ then
        $\mathsf{pp} \leftarrow \mathsf{Del}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*);\ \mathcal{D} \leftarrow \mathcal{D} \setminus \{\mathsf{id}^*\};\ \mathsf{id}^* \leftarrow \bot$
3. $(\mathsf{id}, m_0, m_1) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*);\ b \leftarrow\!\!\$\ \{0, 1\}$;
     if $\mathsf{id}^* \notin S$ then
       for each $\eta \in [k]$ :
         $(m_{b,1}^{(\eta)}, m_{b,2}^{(\eta)}) \leftarrow \mathsf{Sh.Share}(m_b);\ \mathsf{ct}_1^{(\eta)} \leftarrow \widetilde{\mathsf{LE}}.\mathsf{Enc}(\widetilde{\mathsf{crs}}, \widetilde{\mathsf{pp}}, \mathsf{id}^{*(\eta)}, m_{b,1}^{(\eta)})$;
         $\mathsf{ct}_2^{(\eta)} \leftarrow \mathsf{VCWE.Enc}(\mathsf{crs}, D, \mathsf{id}^{*(\eta)}, \mathsf{id}, m_{b,2}^{(\eta)})$;
       $\mathsf{ct} \leftarrow (\mathsf{ct}_1^{(\eta)}, \mathsf{ct}_2^{(\eta)})_{\eta \in [k]}$
     else
       $\boxed{\mathsf{ct} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{id}^*}^{(k+1)}, 0)}$
4. $b' \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*, \mathsf{ct})$Output 1 if $b = b'$
```

From the semantic security of PKE, Hybrid_1 and Hybrid_0 are computationally indistinguishable.

In the next hybrid 2 we substitute $\mathsf{ct}_2^{(\eta)}$ with $\mathsf{VCWE.Enc}(\mathsf{crs}, D, \mathsf{id}^{*(\eta)}, \mathsf{id}, 0)$ in case $\boldsymbol{I}[\mathsf{id}^{*(\eta)}] \neq \mathsf{id}^*$.

```
$\mathsf{Hybrid}_2$
1. $\mathsf{pp} \leftarrow \bot;\ \mathsf{aux} \leftarrow \bot;\ \mathcal{D} \leftarrow \emptyset;\ \mathsf{id}^* \leftarrow \bot;\ \mathsf{pk}^* \leftarrow \bot;\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$
2. $\textbf{for } i = 1 \textbf{ to } Q \textbf{ do}$ one of the following:
  $(a)$ $(\mathsf{id}, \mathsf{pk}) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*);$
      $\textbf{if } \mathsf{id} \notin \mathcal{D} \textbf{ then } \mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk});\ \mathcal{D} \leftarrow \mathcal{D} \cup \{\mathsf{id}\}$
  $(b)$ $\mathsf{id} \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*);$
      $\textbf{if } \mathsf{id} \in \mathcal{D} \textbf{ then } \mathsf{pp} \leftarrow \mathsf{Del}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id});\ \mathcal{D} \leftarrow \mathcal{D} \setminus \{\mathsf{id}\}$
  $(c)$ $\textbf{if } \mathsf{id}^* = \bot \textbf{ then}$
        $\mathsf{id}^* \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*);\ (\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{Gen}(\mathsf{crs}, \mathsf{id}^*);$
        $\mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*);\ \mathcal{D} \leftarrow \mathcal{D} \cup \{\mathsf{id}^*\}$
  $(d)$ $\textbf{if } \mathsf{id}^* \neq \bot \textbf{ then}$
        $\mathsf{pp} \leftarrow \mathsf{Del}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*);\ \mathcal{D} \leftarrow \mathcal{D} \setminus \{\mathsf{id}^*\};\ \mathsf{id}^* \leftarrow \bot$
3. $(\mathsf{id}, m_0, m_1) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*);\ b \leftarrow\!\!\$\ \{0,1\};$
    $\textbf{if } \mathsf{id}^* \notin S \textbf{ then}$
      $\textbf{for each } \eta \in [k] \textbf{ s.t. } \boldsymbol{I}[\mathsf{id}^{*(\eta)}] \neq \mathsf{id}^* :$
        $(m_{b,1}^{(\eta)}, m_{b,2}^{(\eta)}) \leftarrow \mathsf{Sh.Share}(m_b);\ \mathsf{ct}_1^{(\eta)} \leftarrow \widetilde{\mathsf{LE}}.\mathsf{Enc}(\widetilde{\mathsf{crs}}, \widetilde{\mathsf{pp}}, \mathsf{id}^{*(\eta)}, m_{b,1}^{(\eta)});$
        $\boxed{\mathsf{ct}_2^{(\eta)} \leftarrow \mathsf{VCWE.Enc}(\mathsf{crs}, D, \mathsf{id}^{*(\eta)}, \mathsf{id}, 0)}$
      $\textbf{for each } \eta \in [k] \textbf{ s.t. } \boldsymbol{I}[\mathsf{id}^{*(\eta)}] = \mathsf{id}^* :$
        $(m_{b,1}^{(\eta)}, m_{b,2}^{(\eta)}) \leftarrow \mathsf{Sh.Share}(m_b);\ \mathsf{ct}_1^{(\eta)} \leftarrow \widetilde{\mathsf{LE}}.\mathsf{Enc}(\widetilde{\mathsf{crs}}, \widetilde{\mathsf{pp}}, \mathsf{id}^{*(\eta)}, m_{b,1}^{(\eta)});$
        $\mathsf{ct}_2^{(\eta)} \leftarrow \mathsf{VCWE.Enc}(\mathsf{crs}, D, \mathsf{id}^{*(\eta)}, \mathsf{id}, m_{b,2}^{(\eta)})$
    $\textbf{else}$
      $\boxed{\mathsf{ct} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{id}^*}^{(k+1)}, 0)}$
4. $b' \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*, \mathsf{ct})$
$\textbf{Output } 1 \textbf{ if } b = b'$
```

From the semantic security of VCWE, Hybrid_2 and Hybrid_1 are computationally indistinguishable. Observe that (1) $D = \mathsf{Com}(\mathsf{crs}, \boldsymbol{I})$ is honestly computed and (2) the condition of the Vector Commitment Witness Encryption, $\boldsymbol{I}[\mathsf{id}^{*(\eta)}] = \mathsf{id}^*$, does not hold, therefore the semantic secuirty applies.

In the next hybrid 3 we substitute $\mathsf{ct}_1^{(\eta)}$ with $\widetilde{\mathsf{LE}}.\mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^{*(\eta)}, 0)$ in the opposite case of the previous hybrid, $\boldsymbol{I}[\mathsf{id}^{*(\eta)}] = \mathsf{id}^*$.

```
$\mathsf{Hybrid}_3$
1. $\mathsf{pp} \leftarrow \bot;\ \mathsf{aux} \leftarrow \bot;\ \mathcal{D} \leftarrow \emptyset;\ \mathsf{id}^* \leftarrow \bot;\ \mathsf{pk}^* \leftarrow \bot;\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$
2. $\textbf{for } i = 1 \textbf{ to } Q \textbf{ do}$ one of the following:
  $(a)$ $(\mathsf{id}, \mathsf{pk}) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*)$;
      $\textbf{if } \mathsf{id} \notin \mathcal{D} \textbf{ then } \mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk});\ \mathcal{D} \leftarrow \mathcal{D} \cup \{\mathsf{id}\}$
  $(b)$ $\mathsf{id} \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*)$;
      $\textbf{if } \mathsf{id} \in \mathcal{D} \textbf{ then } \mathsf{pp} \leftarrow \mathsf{Del}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id});\ \mathcal{D} \leftarrow \mathcal{D} \setminus \{\mathsf{id}\}$
  $(c)$ $\textbf{if } \mathsf{id}^* = \bot \textbf{ then}$
        $\mathsf{id}^* \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*);\ (\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{Gen}(\mathsf{crs}, \mathsf{id}^*)$;
        $\mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*);\ \mathcal{D} \leftarrow \mathcal{D} \cup \{\mathsf{id}^*\}$
  $(d)$ $\textbf{if } \mathsf{id}^* \neq \bot \textbf{ then}$
        $\mathsf{pp} \leftarrow \mathsf{Del}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*);\ \mathcal{D} \leftarrow \mathcal{D} \setminus \{\mathsf{id}^*\};\ \mathsf{id}^* \leftarrow \bot$
3. $(\mathsf{id}, m_0, m_1) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*);\ b \leftarrow_{\$} \{0, 1\}$;
    $\textbf{if } \mathsf{id}^* \notin S \textbf{ then}$
      $\textbf{for each } \eta \in [k] \textbf{ s.t. } \boldsymbol{I}[\mathsf{id}^{*(\eta)}] \neq \mathsf{id}^*$ :
        $(m_{b,1}^{(\eta)}, m_{b,2}^{(\eta)}) \leftarrow \mathsf{Sh.Share}(m_b);\ \mathsf{ct}_1^{(\eta)} \leftarrow \widetilde{\mathsf{LE}}.\mathsf{Enc}(\widetilde{\mathsf{crs}}, \widetilde{\mathsf{pp}}, \mathsf{id}^{*(\eta)}, m_{b,1}^{(\eta)})$;
        $\boxed{\mathsf{ct}_2^{(\eta)} \leftarrow \mathsf{VCWE.Enc}(\mathsf{crs}, D, \mathsf{id}^{*(\eta)}, \mathsf{id}, 0)}$
      $\textbf{for each } \eta \in [k] \textbf{ s.t. } \boldsymbol{I}[\mathsf{id}^{*(\eta)}] = \mathsf{id}^*$ :
        $(m_{b,1}^{(\eta)}, m_{b,2}^{(\eta)}) \leftarrow \mathsf{Sh.Share}(m_b);\ \boxed{\mathsf{ct}_1^{(\eta)} \leftarrow \widetilde{\mathsf{LE}}.\mathsf{Enc}(\widetilde{\mathsf{crs}}, \widetilde{\mathsf{pp}}, \mathsf{id}^{*(\eta)}, 0)}$;
        $\mathsf{ct}_2^{(\eta)} \leftarrow \mathsf{VCWE.Enc}(\mathsf{crs}, D, \mathsf{id}^{*(\eta)}, \mathsf{id}, m_{b,2}^{(\eta)})$
    $\textbf{else}$
      $\boxed{\mathsf{ct} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{id}^*}^{(k+1)}, 0)}$
4. $b' \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*, \mathsf{ct})\textbf{Output } 1 \textbf{ if } b = b'$
```

From the security of LE, Hybrid_3 and Hybrid_2 are computationally indistinguishable. Let $\ell \in [k]$ such that $\boldsymbol{I}[\mathsf{id}^{*(\ell)}] = \mathsf{id}^*$, then $\mathsf{id}^{*(\ell)}$, with corresponding $\mathsf{pk}^{*(\ell)}$ and $\mathsf{sk}^{*(\ell)}$ honestly generated by the challenger, plays the role of the target identity in the security game of $\widetilde{\mathsf{LE}}$, therefore the security of $\widetilde{\mathsf{LE}}$ applies.

In our last hybrid 4 we substitute $m_{b,1}^{(\eta)}$ with $0_1^{(\eta)}$ where $(0_1^{(\eta)}, 0_2^{(\eta)}) \leftarrow \mathsf{Sh.Share}(0)$ for each $\eta \in [k]$ where $\boldsymbol{I}[\mathsf{id}^{*(\eta)}] \neq \mathsf{id}^*$ and in the opposite case, $m_{b,2}^{(\eta)}$ with $0_2^{(\eta)}$ where $(0_1^{(\eta)}, 0_2^{(\eta)}) \leftarrow \mathsf{Sh.Share}(0)$ for each $\eta \in [k]$ where $\boldsymbol{I}[\mathsf{id}^{*(\eta)}] = \mathsf{id}^*$.

```
$\mathsf{Hybrid}_4$
1. $\mathsf{pp} \leftarrow \bot$; $\mathsf{aux} \leftarrow \bot$; $\mathcal{D} \leftarrow \emptyset$; $\mathsf{id}^* \leftarrow \bot$; $\mathsf{pk}^* \leftarrow \bot$; $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$
2. for $i = 1$ to $Q$ do one of the following:
  (a) $(\mathsf{id}, \mathsf{pk}) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*)$;
      if $\mathsf{id} \notin \mathcal{D}$ then $\mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk})$; $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathsf{id}\}$
  (b) $\mathsf{id} \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*)$;
      if $\mathsf{id} \in \mathcal{D}$ then $\mathsf{pp} \leftarrow \mathsf{Del}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id})$; $\mathcal{D} \leftarrow \mathcal{D} \setminus \{\mathsf{id}\}$
  (c) if $\mathsf{id}^* = \bot$ then
        $\mathsf{id}^* \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*)$; $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{Gen}(\mathsf{crs}, \mathsf{id}^*)$;
        $\mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*)$; $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathsf{id}^*\}$
  (d) if $\mathsf{id}^* \neq \bot$ then
        $\mathsf{pp} \leftarrow \mathsf{Del}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*)$; $\mathcal{D} \leftarrow \mathcal{D} \setminus \{\mathsf{id}^*\}$; $\mathsf{id}^* \leftarrow \bot$
3. $(\mathsf{id}, m_0, m_1) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*)$; $b \leftarrow\!\$\ \{0, 1\}$;
     if $\mathsf{id}^* \notin S$ then
       for each $\eta \in [k]$ s.t. $\boldsymbol{I}[\mathsf{id}^{*(\eta)}] \neq \mathsf{id}^*$ :
         $(m_{b,1}^{(\eta)}, m_{b,2}^{(\eta)}) \leftarrow \mathsf{Sh.Share}(m_b)$; $\mathsf{ct}_1^{(\eta)} \leftarrow \widetilde{\mathsf{LE}}.\mathsf{Enc}(\widetilde{\mathsf{crs}}, \widetilde{\mathsf{pp}}, \mathsf{id}^{*(\eta)}, \boxed{0_1^{(\eta)}})$;
         $\boxed{\mathsf{ct}_2^{(\eta)} \leftarrow \mathsf{VCWE.Enc}(\mathsf{crs}, D, \mathsf{id}^{*(\eta)}, \mathsf{id}, 0)}$
       for each $\eta \in [k]$ s.t. $\boldsymbol{I}[\mathsf{id}^{*(\eta)}] = \mathsf{id}^*$ :
         $(m_{b,1}^{(\eta)}, m_{b,2}^{(\eta)}) \leftarrow \mathsf{Sh.Share}(m_b)$; $\boxed{\mathsf{ct}_1^{(\eta)} \leftarrow \widetilde{\mathsf{LE}}.\mathsf{Enc}(\widetilde{\mathsf{crs}}, \widetilde{\mathsf{pp}}, \mathsf{id}^{*(\eta)}, 0)}$;
         $\mathsf{ct}_2^{(\eta)} \leftarrow \mathsf{VCWE.Enc}(\mathsf{crs}, D, \mathsf{id}^{*(\eta)}, \mathsf{id}, \boxed{0_2^{(\eta)}})$
     else
       $\boxed{\mathsf{ct} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{id}^*}^{(k+1)}, 0)}$
4. $b' \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*, \mathsf{ct})$Output 1 if $b = b'$
```

Observe that in Hybrid_3 in both cases information-theoretically $\mathcal{A}$ learns no information about 1 of the 2 shares, therefore from the $(2, 2)$-privacy of Sh we get that Hybrid_4 and Hybrid_3 are indistinguishable.

In Hybrid_4, $\Pr[b = b'] = 1/2$ information-theoretically, which concludes our proof. □

Efficiency. Regarding efficiency, inspecting the scheme gives:

$$|\mathsf{crs}| = |\widetilde{\mathsf{crs}}| + |\mathsf{chpp}| + |\mathsf{vccrs}|; \qquad |\mathsf{pp}| = |\widetilde{\mathsf{pp}}| + |D| + |S|;$$
$$|\mathsf{u}| = |\widetilde{\mathsf{u}}| + |\Psi| + \log(k); \qquad |\mathsf{ct}| = k(|\widetilde{\mathsf{ct}}| + |\mathsf{ct}_{\mathsf{VCWE}}|).$$

Theorem 5 (Efficiency). *If* CH *is a* $\mathsf{negl}(\lambda)$*-robust Cuckoo Hashing scheme with* $s = o(N)$ *in the worst case,* VC *is a succinct VC with sublinear CRS and for any PPT adversary of* $\mathsf{Comp}_{\mathcal{A}}(\lambda)$ $\widetilde{\mathsf{LE}}$ *has:*

Compactness: $|\widetilde{\mathsf{crs}}|, |\widetilde{\mathsf{pp}}|, |\widetilde{\mathsf{u}}|$ ***Ciphertext size:*** $|\widetilde{\mathsf{ct}}|$

then the LE *scheme presented above has (in the worst-case):*

$$\textbf{\textit{Compactness:}}\ |\mathsf{crs}| = |\widetilde{\mathsf{crs}}| + |\mathsf{chpp}| + |\mathsf{vccrs}|, \quad |\mathsf{pp}| = |\widetilde{\mathsf{pp}}| + |D| + |S|,$$
$$|\mathsf{u}| = |\widetilde{\mathsf{u}}| + |\Psi| + \log(k);$$
$$\textbf{\textit{Ciphertext size:}}\ k(|\widetilde{\mathsf{ct}}| + |\mathsf{ct}_{\mathsf{VCWE}}|).$$

As discussed in Section 3.2 and Theorem 1 there exist a CH with $k = \lambda$ and $s = 0$. Furthermore if $|\widetilde{\mathsf{crs}}| \geq |\mathsf{vccrs}|$, $|\widetilde{\mathsf{pp}}| \geq |D|$, $|\widetilde{\mathsf{u}}| \geq |\Psi|$ and $|\mathsf{ct}| \geq \mathsf{ct}_{\mathsf{VCWE}}$ (as one will see in Section 5 there are VCs with corresponding VCWE that satisfy these conditions for the known $\widetilde{\mathsf{LE}}$ constructions) then $|\mathsf{crs}| = O(|\widetilde{\mathsf{crs}}|)$, $|\mathsf{pp}| = O(|\widetilde{\mathsf{pp}}|)$, $|\mathsf{u}| = O(|\widetilde{u}|)$[11] and $|\mathsf{ct}| = O(\lambda|\widetilde{\mathsf{ct}}|)$. This means that the only (asymptotic) overhead of our compiler for LE is on the ciphertext size. We elaborate more on the efficiency of concrete RBE constructions resulting from our compiler in Section 5.

Regarding the sublinear CRS requirement above, although pairing-based VC typically have linear-sized CRS, there is a well known trick [BGW05] that trades sublinear (square-root) CRS to square-root commitments, resulting to overall sublinear parameters $|\mathsf{crs}| + |D|$.

The final step is to show a tranformation from LE to an RBE with $\mathsf{polylog}(\lambda)$ number of u-updates for each user in the worst case. But this comes directly from Theorem 2.

Corollary 1. *If* CH *is a* $\mathsf{negl}(\lambda)$*-robust Cuckoo Hashing scheme with* $s = o(N)$ *in the worst case,* VC *is a succinct VC with sublinear CRS and for any PPT adversary of* $\mathsf{Comp}_{\mathcal{A}}(\lambda)$ $\widetilde{\mathsf{RBE}}$ *is compact then* RBE *is compact and efficient on the number of updates.*

4.4 A More Efficient Compiler with Selective-Secure Compactness

As argued previously, assuming that there are efficient instantiations of VC and VCWE, the above compiler adds a minimal efficiency overhead. However, the ciphertext size becomes k times larger, where k is the parameter from CH (number of hash functions). [Yeo23] showed that a $\mathsf{negl}(\lambda)$-robust Cuckoo Hashing requires either $k = \lambda$ hash functions, or $s = n$ stash size. We recall that the size of the stash impacts our public parameters as $|\mathsf{pp}| = |\widetilde{\mathsf{pp}}| + |D| + |S|$.

This leads us to the following relaxation: Let CH_2 be the cuckoo hashing scheme from Theorem 1 that has $k = 2$ and an unbounded stash. Theorem 1 indicates that CH_2 achieves $s = \log n$ if the adversary chooses the identities independently of the hash functions' representation (correctness), or $s = n$ in the worst case (robustness). Therefore, when using CH_2, the resulting RBE scheme is secure, complete, has smaller ciphertexts and $\log n$ number of updates (in the worst case). However, it is compact only assuming a selective adversary that chooses the identities independently of chpp.

A way of interpreting selective compactness is saying that public parameters are compact (in the worst case) as long as an adversary is not actively trying to blow them up. [Yeo23] presented an attack that requires heavy (but still polynomial) computations to expand the size of the stash, when having oracle access to the hash functions. Hence, though this is possible, the adversary should still dedicate substantial computational resources to blow up the size of the stash (and thus of the parameters). In our view, selective compactness is less weak than selective security (or a notion of selective completeness, where a similar attack could compromise the correct functioning of the system) in view of the fact that there is no clear motive for an adversary to expand the public parameters of the system just to make it inefficient (security and completeness are still computationally impossible to break). We leave as an interesting open problem the investigation of practical ways to mitigate this kind of DoS attacks.

[11] The $\log k = \log \lambda$ factor is in bits, while the rest are in cryptographic elements (e.g. Group elements or Lattice matrices) therefore $\log \lambda$ bits correspond to one element.

Below we formally define Selective Compactness, similar to compactness but with an adversary who chooses the identities to be registered before seeing chpp.

Definition 10 (Selective Compactness). *An RBE scheme has selective compactness, if in the following* $\mathsf{SelCompactness}_{\mathcal{A}}(\lambda)$ *game:* $|\mathsf{pp}_q|$ *is sublinear in* $|\mathcal{D}_q|$ *and for all* $\mathsf{id} \in \mathcal{D}$*,* $|\mathsf{u}_q|$ *is also sublinear in* $|\mathcal{D}_q|$*.*

```
$\mathsf{SelCompactness}_{\mathcal{A}}(\lambda)$
1. $\mathsf{pp} \leftarrow \bot$; $\mathsf{aux} \leftarrow \bot$; $\mathsf{u} \leftarrow \bot$; $\mathcal{D} \leftarrow \emptyset$; $\mathsf{id}^* \leftarrow \bot$; $t \leftarrow 0$; $\hat{N} \leftarrow 0$;
   $(\widetilde{\mathsf{crs}}, \mathsf{chpp}, \mathsf{vccrs}) \leftarrow \mathsf{Setup}(1^\lambda)$
2. for $i = 1$ to $Q$ do one of the following:
  (a) $(\mathsf{id}, \mathsf{pk}) \leftarrow \mathcal{A}(1^\lambda)$;
      if $|\mathcal{D}| < n \wedge \mathsf{id} \notin \mathcal{D} \wedge (\mathsf{pk}, \cdot) \in \mathsf{Gen}(\mathsf{crs}, \mathsf{id})$ then
        $\mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk})$; $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathsf{id}\}$; $\hat{N} \leftarrow \hat{N} + 1$
  (b) if $|\mathcal{D}| < n \wedge \mathsf{id}^* = \bot$ then
        $\mathsf{id}^* \leftarrow \mathcal{A}(1^\lambda)$; $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{Gen}(\mathsf{crs}, \mathsf{id}^*)$;
        $\mathsf{pp} \leftarrow \mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}^*, \mathsf{pk}^*)$; $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathsf{id}^*\}$; $\hat{N} \leftarrow \hat{N} + 1$
  (c) if $\mathsf{id}^* \neq \bot$ then
        $m_t \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*)$; $t \leftarrow t + 1$;
        $\mathsf{ct}_t \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{id}^*)$
  (d) $j \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pp}, \mathsf{aux}, \mathsf{pk}^*)$
      if $j \in [t]$ then
        $m_j \leftarrow \mathsf{Dec}(\mathsf{sk}^*, \mathsf{u}, \mathsf{ct}_j)$
        if $m'_j = \mathsf{GetUpd}$ then
          $\mathsf{u}^* \leftarrow \mathsf{Upd}^{[\mathsf{aux}]}(\mathsf{pp}, \mathsf{id}^*)$; $m_j \leftarrow \mathsf{Dec}(\mathsf{sk}^*, \mathsf{u}^*, \mathsf{ct}_j)$
```

For all queries $q \in [Q]$*,* pp_q *are the public parameters after the q-th query and* u_q *the corresponding update information of* id*.*

Theorem 6 (RBE with selective compactness). *There exists a* CH *scheme such that the resulting RBE scheme of the compiler,* RBE*, is secure, complete and has:*

- **Selective Compactness:** $|\mathsf{crs}| = |\widetilde{\mathsf{crs}}| + |\mathsf{chpp}| + |\mathsf{vccrs}|$, $|\mathsf{pp}| = (|\widetilde{\mathsf{pp}}| + |D| + \log n) \log n$, $|\mathsf{u}| = |\widetilde{\mathsf{u}}| + |\Psi| + 1$,
- **Number of Updates:** $\log n$
- **Ciphertext size:** $2\left(|\widetilde{\mathsf{ct}}| + |\mathsf{ct}_{\mathsf{VCWE}}|\right) \log n$

The proof of this theorem is straightforward, adapting the construction of Section 4.2 with CH_2. The main difference with Theorem 5 is that in CH_2 the number of hash function is $k = 2$, affecting the ciphertext-size directly.

5 Concrete RBE Schemes

In this section, we discuss two RBE constructions that result from instantiating the compiler of the previous section. The first RBE is from Pairings assuming the hardness of the (decisional) Bilinear Diffie-Hellman Exponent (DBDHE) problem, while the second RBE is from Lattices assuming the hardness the Learning with Errors problem.

To obtain these instantiations, we use that based on DBDHE (resp. LWE), there exist black-box: RBE schemes with small $\mathcal{ID}$ space [GKMR22] (resp. [DKL$^+$23][12]), and VC schemes [LY10] (resp. [LLNW16]). Additionally, there are PKE schemes from DDH [ElG85] and LWE [Reg05, GPV08]. Therefore, to complete the building blocks of our compiler, in this section we construct two Witness Encryption for Vector Commitments from the respective assumptions. we present them in Section 5.1 and Section 5.2. For completeness, and since the resulting RBE scheme over pairings comprises our central result, we also present explicitly our final pairing-based RBE construction in section 5.1.

5.1 Our Pairing-Based RBE

Preliminaries on Pairings For simplicity we use symmetric pairing groups, but our schemes can also work in assymetric pairing groups. A generator $\mathcal{BG}$ takes as input a security parameter 1^λ and outputs a description $\mathbb{G} := (p, \mathbb{G}, \mathbb{G}_T, g, e)$, where p is a prime of $\Theta(\lambda)$ bits, $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of order p, and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a non-degenerate bilinear map. We require that the group operations in $\mathbb{G}$, $\mathbb{G}_T$ and the bilinear map e are computable in deterministic polynomial time in λ. Let $g \in \mathbb{G}$ and $g_T = e(g_1, g_2) \in \mathbb{G}_T$ be the respective generators.

VCWE from dBDHE. We recall the [LY10] Vector Commitment:

- $\mathsf{Setup}(1^\lambda, N) \to \mathsf{crs}$: $\alpha \leftarrow\!\!\$\ \mathbb{Z}_p^*$, $g_i = g^{\alpha^i}$, $\mathsf{crs} = \{g_i\}_{i\in[2N], i\neq N+1}$.
- $\mathsf{Com}(\mathsf{crs}, \boldsymbol{v}) \to C$: Parses $\boldsymbol{v} := (v_1, \ldots, v_N)$, $C = \prod_{i=1}^{N} (g_i)^{v_i}$.
- $\mathsf{Open}(\mathsf{crs}, C, \boldsymbol{v}, i) \to \Lambda$: Parses $\boldsymbol{v} := (v_1, \ldots, v_N)$, $\Lambda_i = \prod_{j=1, j\neq i}^{N} (g_{N+1-i+j})^{v_j}$.
- $\mathsf{Ver}(\mathsf{crs}, C, \Lambda, v, i) \to b$: returns 1 iff $e(C, g_{N+1-i}) = e(\Lambda, g) e(g_i^{v_i}, g_{N+1-i})$.

Our Witness-Encryption is reminiscent of the encryption technique of [GKMR22] with the difference that in the case of VCWE we need to fix the value of the corresponding position i to v, so that one can decrypt exactly iff $\boldsymbol{v}[i] = v$.

The formal description of the VCWE with respect to the [LY10] VC is below.

- $\mathsf{VCWE.Enc}(\mathsf{crs}, C, i, v, m) \to \mathsf{ct}$: For a message $m \in \mathbb{G}_T$, samples $r \leftarrow\!\!\$\ \mathbb{Z}_p^*$ and computes:
$$\mathsf{ct} = \left(g^r, e(g_i^{-v} C, g_{N+1-i})^r \cdot m\right)$$
- $\mathsf{VCWE.Dec}(\mathsf{crs}, C, i, v, \Lambda, \mathsf{ct}) \to m^*$: Parses $\mathsf{ct} := (\mathsf{ct}_1, \mathsf{ct}_2)$ and computes:
$$m^* = \mathsf{ct}_2 / e(\Lambda, \mathsf{ct}_1)$$

Correctness. It follows directly from the construction that for $\mathsf{crs} = \{g_i\}_{i\in[2N], i\neq N+1}$, $C = \prod_{i=1}^{N} (g_i)^{v_i}$, $\Lambda = \prod_{j=1, j\neq i}^{N} (g_{N+1-i+j})^{v_j}$ we get:

$$\begin{aligned}\mathsf{Dec}(\mathsf{crs}, \Lambda, \mathsf{Enc}(\mathsf{crs}, C, i, v, m)) &= \frac{e(g_i^{-v} C, g_{N+1-i})^r \cdot m}{e(\Lambda, g^r)} \\ &= \frac{e(C, g_{N+1-i})}{e(\Lambda, g) e(g_i^{v}, g_{N+1-i})} \cdot m \\ &= m\end{aligned}$$

Where the last equality comes from the correctness property of the original [LY10] VC which indicates that $e(C, g_{N+1-i}) = e(\Lambda, g) e(g_i^{v_i}, g_{N+1-i})$.

[12] In [DKL$^+$23] $\mathcal{ID}$ can be arbitrarily large. We make use of the scheme with small identities to argue that compiling it to a large $\mathcal{ID}$ with our transformation instead can benefit efficiency.

Security. First, we recall the the (decision) Bilinear Diffie-Hellman assumption introduced by Boneh et. al. [BGW05] a standard assumptions over Bilinear Groups. Then our VCWE scheme can be proven secure in a straightforward way under the aforementioned assumption.

Assumption 1 (Decision N-BDHE assumption)**.** *Let $\mathcal{BG}$ be a bilinear group generator,* $\mathsf{bg} := (p, \mathbb{G}, \mathbb{G}_T, g, e) \leftarrow\$ \mathcal{BG}(1^\lambda)$*,* $\alpha, r \leftarrow\$ \mathbb{Z}_p^*$ *and define*

$$\mathcal{D} = \left(\left\{g^{\alpha^i}\right\}_{i\in[N]}, g^r\right) \qquad \textit{and} \qquad R \leftarrow\$ \mathbb{G}_T.$$

Then for any PPT adversary $\mathcal{A}$ and any positive integer N the following is negligible in the security parameter:

$$\left|\Pr[\mathsf{Adv}\left(\mathsf{bg}, \mathcal{D}, e(g,g)^{r\alpha^{N+1}}\right) = 1] - \Pr[\mathsf{Adv}\left(\mathsf{bg}, \mathcal{D}, R\right) = 1]\right| = \mathsf{negl}(\lambda)$$

where the above probabilities are taken over the choices of bg, α, r *and* R*.*

Theorem 7. *The Vector Commitment-Witness Encryption described above satisfies* VCWE *semantic security, under the desicional N-BDHE assumption.*

Proof. The reduction goes as follows:

Let a PPT adversary $\mathcal{B}$ to the N-BDHE assumption, that receives $(\mathsf{bg}, \{g^{\alpha^i}\}_{i\in[N]}, g^r, R)$ (denote $g^{\alpha^i} = g_i, g^r := h$) and wants to distinguish whether R is of the form $e(g,g)^{r\alpha^{N+1}}$ or uniformly random.

Assume that a PPT adversary $\mathcal{A}$ wins the semantic security game with a non-negligible probability ϵ. We show that $\mathcal{B}$ can use $\mathcal{A}$ to break the (decisional) N-BDHE assumption. First $\mathcal{B}$ sets $\mathsf{crs} = (\mathsf{bg}, g_1, \ldots, g_N)$ and sends crs to $\mathcal{A}$ who returns $(\boldsymbol{v}, i, v, m_0, m_1)$. If $\boldsymbol{v}[i] = v$ then $\mathcal{B}$ sets $\mathsf{ct} = \perp$. Othewise computes $C = \prod_{i\in[N]} g_i^{v_i}$, samples $b \leftarrow\$ \{0,1\}$ and sets

$$\mathsf{ct} = \left(h, R^{v_i - v} \prod_{j\in[N], j\neq i} e(h, g_{N+1-i+j})^{v_j} \cdot m_b\right).$$

Then $\mathcal{B}$ sends ct to $\mathcal{A}$ who responds with a bit b'.

Note that If $R = e(g,g)^{r\alpha^{N+1}}$ then:

$$\begin{aligned}
\mathsf{ct} &= \left(g^r, e(g,g)^{r\alpha^{N+1}(v_i - v)} \prod_{j\in[N], j\neq i} e(g^r, g^{\alpha^{N+1-i+j}})^{v_j} \cdot m_b\right) = \\
&= \left(g^r, e(g^{\alpha^i(v_i - v)}, g^{\alpha^{N+1-i}})^r \prod_{j\in[N], j\neq i} e(g^{\alpha^j v_j}, g^{\alpha^{N+1-i}})^r \cdot m_b\right) = \\
&= \left(g^r, e(g_i^{-v} g_i^{v_i} \prod_{j\in[N], j\neq i} g_j^{v_j}, g_{N+1-i})^r \cdot m_b\right) \\
&= \left(g^r, e(g_i^{-v} C, g_{N+1-i})^r \cdot m_b\right)
\end{aligned}$$

is perfectly indistinguishable from a real one so $\Pr[b = b'] = \epsilon$. If R is uniformly random then the ciphertext leaks nothing about M_b, so $\Pr[b = b'] = 1/2$ and obviously the same holds when $\boldsymbol{v}[i] = v$ and $\mathsf{ct} = \perp$. It follows that $\mathcal{B}$ has advantage ϵ in distinguishing between $R = e(g,g)^{r\alpha^{N+1}}$ and a random R. In conclusion $\mathcal{B}$ has advantage ϵ solving the decisional N-BDHE problem, which contradicts the corresponding assumption. □

The Pairing-based RBE construction unfolded. We assume a cuckoo hashing scheme $\mathsf{CH} = (\mathsf{Setup}, \mathsf{Insert}, \mathsf{Lookup})$. For any id, we denote $(\mathsf{id}^{(1)}, \ldots, \mathsf{id}^{(k)}) \leftarrow \mathsf{Lookup}(\mathsf{pp}, \mathsf{id})$. As in the previous section we present the Laconic Encryption version of the protocol.

- $\mathsf{Setup}(1^\lambda, n) \rightarrow \mathsf{crs}$:
  1. $\mathsf{chpp} \leftarrow \mathsf{CH.Setup}(1^\lambda, n)$,
  2. $(p, \mathbb{G}, \mathbb{G}_T, g, e) \leftarrow\!\!\$\ \mathcal{BG}(1^\lambda)$,
  3. $\alpha \leftarrow\!\!\$\ \mathbb{Z}_p^*$ and $g_i = g^{\alpha^i}$ for each $i \in [2N]$.

  $$\mathsf{crs} = \big(\mathsf{chpp}, \mathsf{bg}, \{g_i\}_{i \in [2N], i \neq N+1}\big)$$

- $\mathsf{Gen}(\mathsf{crs}, \mathsf{id}) \rightarrow (\mathsf{pk}, \mathsf{sk})$:
  1. $x^{(\eta)} \leftarrow\!\!\$\ \mathbb{Z}_p^*$
  2. $\mathsf{pk}^{(\eta)} = (g_{\mathsf{id}^{(\eta)}})^{x^{(\eta)}}$ for each $\eta \in [k]$,
  3. for each $\eta \in [k]$

  $$\boldsymbol{u}^{(\eta)} = \Big((g_{\mathsf{id}^{(\eta)}+N})^{x^{(\eta)}}, \ldots, (g_{2+N})^{x^{(\eta)}}, \emptyset, (g_n)^{x^{(\eta)}}, \ldots,$$
  $$\ldots, (g_{\mathsf{id}^{(\eta)}+1})^{x^{(\eta)}}, (g_{\mathsf{id}^{(\eta)}})^{x^{(\eta)}}\Big),$$

  4. $x^{(k+1)} \leftarrow\!\!\$\ \mathbb{Z}_p^*$
  5. $\mathsf{pk}^{(k+1)} = g^{x^{(k+1)}}$,

  $$\mathsf{pk} = \Big((\mathsf{pk}^{(\eta)})_{\eta \in [k]}, \mathsf{pk}^{(k+1)}, (\boldsymbol{u}^{(\eta)})_{\eta \in [k]}\Big), \quad \mathsf{sk} = \Big((x^{(\eta)})_{\eta \in [k]}, x^{(k+1)}\Big)$$

- $\mathsf{Reg}^{[\mathsf{aux}]}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, \mathsf{pk}) \rightarrow \mathsf{pp}'$: The parameters (before the first registration) are initialized as $\mathsf{pp} = (1, \ldots, 1) \in \mathbb{G}^N$ and $\mathsf{aux} = (\{1\}, \ldots, \{1\}) \in \mathbb{G}^N$.
  1. Validates $\boldsymbol{u}^{(\eta)} := (u_i^{(\eta)})_{i \in [N+1]}$:

  $$e(u_1^{(\eta)}, g) = \ldots = e(u_{\mathsf{id}^{(\eta)}-1}^{(\eta)}, g_{\mathsf{id}^{(\eta)}-2}) = e(u_{\mathsf{id}^{(\eta)}+1}^{(\eta)}, g_{\mathsf{id}^{(\eta)}}) = \ldots$$
  $$\ldots = e(u_N^{(\eta)}, g_{N-1}) = e(\mathsf{pk}^{(\eta)}, g_N)$$

  for each $\eta \in [k]$. If the above does not hold outputs $\perp$.
  2. Parses $\mathsf{aux} := (\boldsymbol{I}, D, \boldsymbol{\Psi}, \mathbf{pk}, C, \boldsymbol{\Lambda}, S)$ which is the current state of the system:
     (a) $\boldsymbol{I} = (\mathsf{id}_1, \ldots, \mathsf{id}_N)$ is the vector of the (previously) registered identities in the corresponding positions (if no identity is registered in position j then $\mathsf{id}_j = 0$),
     (b) $D := \prod_{i \in [N]} g_i^{\mathsf{id}_i}$ is the current vector commitment to the vector of identities $\boldsymbol{I}$,
     (c) $\boldsymbol{\Psi} = (\Psi_1, \ldots, \Psi_N) := \Big(\prod_{j \in [N], j \neq i} (g_{N+1-i+j})^{\mathsf{id}_j}\Big)_{i \in [N]}$ are the opening proofs wrt the VC of identities D,
     (d) $\mathbf{pk} := (\mathsf{pk}_{\mathsf{id}_1}, \ldots, \mathsf{pk}_{\mathsf{id}_N}) := \Big((\mathsf{pk}_{\mathsf{id}_i}^{(\eta)})_{\eta \in [k]}, \mathsf{pk}_{\mathsf{id}_i}^{(k+1)}, (\boldsymbol{u}_{\mathsf{id}_i}^{(\eta)})_{\eta \in [k]}\Big)_{i \in [N]}$ (if no identity is registered in position j the $\mathsf{pk}_j = \big((1)_{\eta \in [k]}, 1, (\mathbf{1})_{\eta \in k}\big)$),
     (e) $C := \prod_{i \in [N]} g_i^{\mathsf{sk}_{\mathsf{id}_i}^{(\eta^*)}}$ is the the VC of secret keys (implicitly computed using $\mathbf{pk}$, without knowledge of the actual $\mathbf{sk}$), where $\eta^* \in [k]$ is the hash function that mapped id_i to i,
     (f) $\boldsymbol{\Lambda} = (\Lambda_1, \ldots, \Lambda_N) := \Big(\prod_{j \in [N], j \neq i} (g_{N+1-i+j})^{\mathsf{sk}_{\mathsf{id}_j}^{(\eta^*)}}\Big)_{i \in [N]}$ are the opening proofs wrt the VC of secret keys C (implicitly computed using $\mathbf{pk}$, without knowledge of the actual $\mathbf{sk}$), where $\eta^* \in [k]$ is the hash function that mapped id_j to j.

(g) $S = \Big((\mathsf{id}_{N+1}, \mathsf{pk}_{N+1}^{(k+1)}), \ldots, (\mathsf{id}_{N+s}, \mathsf{pk}_{N+s}^{(k+1)})\Big)$ is a set of size s that represents the stash of the cuckoo hashing scheme.

3. Inserts the new identity id. Runs $(\boldsymbol{I}', S') \leftarrow \mathsf{CH.Insert}(\mathsf{chpp}, \boldsymbol{I}, S, \mathsf{id})$ and if $\mathsf{CH.Insert}$ failed outputs $\perp$ otherwise for every identity $\bar{\mathsf{id}}$ that was moved, i.e. its different position in $(\boldsymbol{I}', S')$ and $(\boldsymbol{I}, S)$ differs, does the following.

   Remove $\tilde{\mathsf{id}}$. Assume that $\bar{\mathsf{id}}$ was in position $\bar{\mathsf{id}}^{(\eta)} = i^*$ for some $\eta \in [k]$:
   (a) $\boldsymbol{I}[i^*] \leftarrow 0$,
   (b) $D \leftarrow D / g_{i^*}^{\tilde{\mathsf{id}}}$,
   (c) $\varPsi_j \leftarrow \varPsi_j / (g_{N+1-i^*+j})^{\bar{\mathsf{id}}}$ for each $j \in [N], j \neq i^*$,
   (d) $\bar{\mathsf{pk}} \leftarrow \mathsf{pk}_{i^*}$ and $\mathbf{pk}[i^*] \leftarrow \mathbf{1}$,
   (e) $C \leftarrow C / \bar{\mathsf{pk}}^{(\eta)}$,
   (f) $\varLambda_j \leftarrow \varLambda_j / \bar{\boldsymbol{u}}^{(\eta)}[j]$ for each $j \in [N], j \neq i^*$.

   Add $(\tilde{\mathsf{id}}, \tilde{\mathsf{pk}})$ to the new position. assume that $\bar{\mathsf{id}}$ goes to position $\bar{\mathsf{id}}^{(\zeta)} = i^\dagger$ for some $\zeta \in [k]$:
   (a) $\boldsymbol{I}[i^\dagger] \leftarrow \bar{\mathsf{id}}$,
   (b) $D \leftarrow D \cdot g_{i^\dagger}^{\bar{\mathsf{id}}}$,
   (c) $\varPsi_j \leftarrow \varPsi_j \cdot (g_{N+1-i^\dagger+j})^{\bar{\mathsf{id}}}$ for each $j \in [N], j \neq i^\dagger$,
   (d) $\mathsf{pk}_{i^\dagger} \leftarrow \bar{\mathsf{pk}}$,
   (e) $C \leftarrow C \cdot \bar{\mathsf{pk}}^{(\zeta)}$,
   (f) $\varLambda_j \leftarrow \varLambda_j \cdot \bar{\boldsymbol{u}}^{(\zeta)}[j]$ for each $j \in [N], j \neq i^\dagger$.

   In the case where $\bar{\mathsf{id}}$ moved to the stash then simply sets $S \leftarrow S \cup \{(\bar{\mathsf{id}}, \bar{\mathsf{pk}}^{(k+1)})\}$.

   For the above to be correct we assume that first all the removeals happen and then all the (re-)additions.

Return:

$$\mathsf{pp}' = (C, D, S)$$

- $\mathsf{Enc}(\mathsf{crs}, \mathsf{pp}, \mathsf{id}, m) \rightarrow \mathsf{ct}$: Parses $\mathsf{pp} := (C, D, S)$ and preceeds as follows.

  Identity in the table. If $\mathsf{id} \notin S$ then for each $\eta \in [k]$:
  1. $r^{(\eta)}, z^{(\eta)} \leftarrow\!\$\ \mathbb{Z}_p^*$,
  2. $m_1^{(\eta)} \leftarrow\!\$\ G_T$ and computes $m_2^{(\eta)}$ s.t. $m = m_1^{(\eta)} \cdot m_2^{(\eta)}$,
  3. computes:

  $$\mathsf{ct}^{(\eta)} = \Big(C, g^{r^{(\eta)}}, e(C, g_{N+1-\mathsf{id}^{(\eta)}})^{r^{(\eta)}}, e(g_{\mathsf{id}^{(\eta)}}, g_{N+1-\mathsf{id}^{(\eta)}})^{r^{(\eta)}} \cdot m_1^{(\eta)},$$
  $$g^{z^{(\eta)}}, e((g_{\mathsf{id}^{(\eta)}})^{-\mathsf{id}} D, g_{N+1-\mathsf{id}^{(\eta)}})^{z^{(\eta)}} \cdot m_2^{(\eta)}\Big).$$

  The final ciphertext is:

  $$\mathsf{ct} = \Big(\mathsf{ct}^{(1)}, \ldots, \mathsf{ct}^{(k)}\Big)$$

  Identity in the stash. If $\mathsf{id} \in S$:
  1. Identify i^* such that $\mathsf{id}_{N+i^*} = \mathsf{id}$,
  2. $r \leftarrow\!\$\ \mathbb{Z}_p^*$,
  3. Return:

  $$\mathsf{ct} = (g^r, m \cdot (\mathsf{pk}_{N+i^*}^{(k+1)})^r)$$

- $\mathsf{Upd}^{[\mathsf{aux}]}(\mathsf{pp}, \mathsf{id}) \rightarrow \mathsf{u}$: Parses $\mathsf{aux} := (\boldsymbol{I}, D, \boldsymbol{\varPsi}, \mathbf{pk}, C, \boldsymbol{\varLambda}, S)$ and identifies the $\eta^* \in [k]$ such that $\boldsymbol{I}[\mathsf{id}^{(\eta^*)}] = \mathsf{id}$ and outputs

  $$\mathsf{u} = (\varPsi_{\mathsf{id}^{(\eta^*)}}, \varLambda_{\mathsf{id}^{(\eta^*)}}, \eta^*).$$

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{u}, \mathsf{ct}) \to m$ or GetUpd :
  Idenity in the table. If ct consists of $6k$ elements:
  1. If $e(\mathsf{ct}_1, g_{n+1-i}) \neq e(\Lambda_i, g) \cdot e((g_i)^{\mathsf{sk}}, g_{n+1-i})$, then outputs GetUpd.
  2. Otherwise computes:
  $$m_1^* = \frac{\mathsf{ct}_4^{(\eta^*)}}{\left(e(\Lambda_i, \mathsf{ct}_3^{(\eta^*)})^{-1} \cdot \mathsf{ct}_2^{(\eta^*)}\right)^{\mathsf{sk}^{-1}}} \quad \text{and} \quad m_2^* = \frac{\mathsf{ct}_6^{(\eta^*)}}{e(\Psi_i, \mathsf{ct}_5^{(\eta^*)})}$$
  3. outputs:
  $$m^* = m_1^* \cdot m_2^*$$

  Identity in the Stash. If ct consists of 2 group elements then outputs:
  $$m^* = \mathsf{ct}_2/(\mathsf{pk}^{(k+1)})^{\mathsf{sk}^{(k+1)}}$$

The protocol as presented above has linear-sized crs. As in [GKMR22] we apply the a well-known square-root tradeoff (appeared first in [BGW05]) that gives us $\sqrt{N}$-sized crs at the cost of having $\sqrt{N}$-sized pp. In conclusion, similarly to [GKMR22] we get square-root compactness.

5.2 From Lattices

In this section we show that a slight modification on the underlying encryption technique of [DKL$^+$23] gives us a Witness Encyption for the [LLNW16] Vector Commitment. The latter is essentially a Merkle tree that uses a clever combination of the Ajtai's [Ajt96] SIS function as a hash function, together with the 'binary decomposition' gadget matrix $\boldsymbol{G}$ [GPV08] to map elements to the right of the hash.

We recall the the LWE assumption introduced by Regev [Reg05]. LWE is parametrized by the dimension n a modulus q and error distribution χ, typically a discrete Gaussian.

Assumption 2 (LWE assumption). *Let $\boldsymbol{s} \leftarrow\!\!\$\ \mathbb{Z}_q^n$, $\boldsymbol{e} \leftarrow\!\!\$\ \chi^m$, $\boldsymbol{A} \leftarrow\!\!\$\ \mathbb{Z}_q^{n\times m}$, $\boldsymbol{b} \leftarrow\!\!\$\ \mathbb{Z}_q^m$ Then for any PPT adversary $\mathcal{A}$ the following is negligible in the security parameter:*

$$\left|\Pr[\mathsf{Adv}\left(\boldsymbol{A}, \boldsymbol{s}^\top\boldsymbol{A} + \boldsymbol{e} \mod q\right) = 1] - \Pr[\mathsf{Adv}\left(\boldsymbol{A}, \boldsymbol{b}\right) = 1]\right| = \mathsf{negl}(\lambda)$$

VCWE from LWE. We recall the [LLNW16] Vector Commitment (SIS-based Merkle tree). Below we use the SIS hash function, $h : \mathbb{Z}_q^n \times \mathbb{Z}_q^n \to \mathbb{Z}_q^n$ as $h(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{A}_0(-\boldsymbol{G}^{-1}(\boldsymbol{x})) + \boldsymbol{A}_1(-\boldsymbol{G}^{-1}(\boldsymbol{y}))$. We also assume wlog that the size of the vector is $N = 2^b$.

- $\mathsf{Setup}(1^\lambda, N) \to \mathsf{crs}$: $\boldsymbol{A}_0, \boldsymbol{A}_1 \leftarrow\!\!\$\ \mathbb{Z}_q^{n\times m}$, $\mathsf{crs} = \boldsymbol{A}$.
- $\mathsf{Com}(\mathsf{crs}, \boldsymbol{v}) \to C$:
  1. Parses $\boldsymbol{v} := (\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{N-1})$, where $\boldsymbol{v}_i \in \mathbb{Z}_q^n$.
  2. Define $\boldsymbol{u}_i^b := \boldsymbol{v}_i$.
  3. Applies recursively (in a Merkle-tree fashion) for each $\ell = b-1, b-2, \ldots, 1$: $\boldsymbol{u}_i^{\ell-1} \leftarrow h(\boldsymbol{u}_{2i}^\ell, \boldsymbol{u}_{2i+1}^\ell)$ for each $i \in \{0, \ldots, 2^\ell/2 - 1\}$.
  4. Define $\boldsymbol{z}_i^\ell := -\boldsymbol{G}^{-1}(\boldsymbol{u}_i^\ell)$.

  Outputs:
  $$C = \boldsymbol{u}_1^0, \qquad \mathsf{aux} = \{\boldsymbol{z}_0^\ell, \ldots, \boldsymbol{z}_{2^\ell-1}^\ell\}_{\ell\in[b]}$$

- $\mathsf{Open}(\mathsf{crs}, \mathsf{aux}, \boldsymbol{v}_i, i) \to \Lambda$: Let $i = (i_{b-1} \dots i_0)_2$ be i's binary representation. Outputs

$$\Lambda_i = \left(\boldsymbol{z}^b_{(i_{b-1},\dots i_1(1-i_0))_2}, \boldsymbol{z}^{b-1}_{(i_{b-1}\dots i_2 1)_2}, \boldsymbol{z}^{b-1}_{(i_{b-1}\dots i_2 0)_2}, \dots, \boldsymbol{z}^1_{(0)_2}, \boldsymbol{z}^1_{(1)_2}\right)$$

- $\mathsf{Ver}(\mathsf{crs}, C, \Lambda, \boldsymbol{v}_i, i) \to b$:
  1. Let $i = (i_{b-1} \dots i_0)_2$ be i's binary representation.
  2. Parses $\Lambda = (\boldsymbol{z}^b_{1-i_0}, \boldsymbol{z}^{b-1}_0, \boldsymbol{z}^{b-1}_1 \dots, \boldsymbol{z}^1_0, \boldsymbol{z}^1_1)$ and sets $\boldsymbol{z}^b_{i_0} = \boldsymbol{v}_i$
  3. Parses $C := \boldsymbol{u}^0_0$
  4. Outuputs 1 iff $\boldsymbol{A}_i \cdot \boldsymbol{z} = \boldsymbol{c}$ holds, where $\boldsymbol{z} = \left(\boldsymbol{z}^1_0, \boldsymbol{z}^1_1, \dots, \boldsymbol{z}^b_0, \boldsymbol{z}^b_1\right)^\top$, $\boldsymbol{c} = \left(\boldsymbol{u}^0_0, \boldsymbol{0} \dots, \boldsymbol{0}\right)^\top$ and

$$\boldsymbol{A}_i = \begin{pmatrix} \boldsymbol{A}_0 & \boldsymbol{A}_1 & & & & & \\ (1-i_{b-1})\boldsymbol{G} & i_{b-1}\boldsymbol{G} & \boldsymbol{A}_0 & \boldsymbol{A}_1 & & & \\ & & (1-i_{b-2})\boldsymbol{G} & i_{b-2}\boldsymbol{G} & & & \\ & & & & \ddots\ \ddots & & \\ & & & & & \boldsymbol{A}_0 & \boldsymbol{A}_1 \\ & & & & & (1-i_0)\boldsymbol{G} & i_0\boldsymbol{G} \end{pmatrix}$$

Döttling et al. [DKL+23] introduced a powerful technique, using the above vector commitment, in which if $\mathsf{sk} = \boldsymbol{v}_i$ is secret then one can encrypt a binary message M w.r.t. this secret with a Dual-Regev-like method as:

$$\mathsf{ct} = \left(\boldsymbol{r}^\top \cdot \boldsymbol{A}_i + \boldsymbol{e}^\top,\ \boldsymbol{r}^\top \boldsymbol{c} + e + \left\lfloor \frac{q}{2} \right\rfloor \cdot M\right)$$

for random $\boldsymbol{r} \leftarrow\!\!\$\ \mathbb{Z}_q^{n(b+1)}$ and noise $\boldsymbol{e} \leftarrow\!\!\$\ \chi^{2m(b+1)}$, $e \leftarrow\!\!\$\ \chi$. Then the party holding $\mathsf{sk} = \boldsymbol{v}_i$ and $\Lambda_i = (\boldsymbol{z}^b_{1-i_0}, \boldsymbol{z}^{b-1}_0, \boldsymbol{z}^{b-1}_1 \dots, \boldsymbol{z}^1_0, \boldsymbol{z}^1_1)$ can decrypt as usual in the Dual-Regev Encryption (assume wlgo that $i_0 = 1$):

$$M^* = 0 \text{ iff } \left|\mathsf{ct}_2 - \mathbf{ct}_1^\top \cdot (\boldsymbol{z}^1_0, \boldsymbol{z}^1_1, \dots, \boldsymbol{z}^b_0, \boldsymbol{v}_i)^\top \mod q\right| < q/4 \text{ otherwise } M^* = 1$$

Our observation is that if we fix $\boldsymbol{v}_i = \boldsymbol{x}$ then this directly yields a witness encryption for $\boldsymbol{v}[i] = \boldsymbol{x}$ (the flavor that we formalize in Section 4.1). In particular VCWE scheme will work as:

- $\mathsf{VCWE.Enc}(\mathsf{crs}, C, i, \boldsymbol{x}, M) \to \mathsf{ct}$:
  1. Parses $C = \boldsymbol{u}^0_0$, computes the binary decomposition $i = (i_{b-1} \dots i_0)_2$.
  2. Samples $\boldsymbol{r} \leftarrow\!\!\$\ \mathbb{Z}_q^{n(b+1)}$, $\boldsymbol{e} \leftarrow\!\!\$\ \chi^{2m(b+1)-1}$, $e \leftarrow\!\!\$\ \chi$.
  3. Computes $\boldsymbol{c} = (\boldsymbol{u}^0_0, 0, \dots, 0, -\boldsymbol{A}_{i_0}\boldsymbol{x}, -\boldsymbol{G}\boldsymbol{x})^\top$ and:

$$\boldsymbol{A}_{i \setminus i_0} = \begin{pmatrix} \boldsymbol{A}_0 & \boldsymbol{A}_1 & & & & \\ (1-i_{b-1})\boldsymbol{G} & i_{b-1}\boldsymbol{G} & \boldsymbol{A}_0 & \boldsymbol{A}_1 & & \\ & & (1-i_{b-2})\boldsymbol{G} & i_{b-2}\boldsymbol{G} & & \\ & & & & \ddots\ \ddots & \\ & & & & & \boldsymbol{A}_{1-i_0} \\ & & & & & \boldsymbol{0} \end{pmatrix}$$

which is essentially $\boldsymbol{A}_i$ without the $(2(b+1) - 1 + i_0)$-th column.

$$\mathsf{ct} = \left(\boldsymbol{r}^\top \cdot \boldsymbol{A}_{i \setminus i_0} + \boldsymbol{e}^\top,\ \boldsymbol{r}^\top \boldsymbol{c} + e + \left\lfloor \frac{q}{2} \right\rfloor \cdot M\right)$$

- $\mathsf{VCWE.Dec}(\mathsf{crs}, C, i, \boldsymbol{x}, \Lambda, \mathsf{ct}) \rightarrow M^*$:
  1. Parses $\Lambda = (\boldsymbol{z}^b_{1-i_0}, \boldsymbol{z}^{b-1}_0, \boldsymbol{z}^{b-1}_1 \ldots, \boldsymbol{z}^1_0, \boldsymbol{z}^1_1)$.
  2. Sets $\boldsymbol{z}_{\setminus i_0} = \left(\boldsymbol{z}^1_0, \boldsymbol{z}^1_1, \ldots, \boldsymbol{z}^{b-1}_0, \boldsymbol{z}^{b-1}_1, \boldsymbol{z}^b_{1-i_0}\right)^\top$,
  3. Outputs $M^* = 0$ iff
  $$\left|\mathsf{ct}_2 - \mathbf{ct}_1^\top \cdot \boldsymbol{z}_{\setminus i_0} \mod q\right| < q/4$$
  otherwise $M^* = 1$.

Correctness. Correctness follows from the typical correctness of the Dual-Regev cryptosystem and from the fact that:

$$\begin{aligned}
&\boldsymbol{A}_i \cdot \left(\boldsymbol{z}^1_0, \boldsymbol{z}^1_1, \ldots, \boldsymbol{z}^{b-1}_0, \boldsymbol{z}^{b-1}_1, \boldsymbol{z}^b_0, \boldsymbol{z}^b_1\right)^\top = \left(\boldsymbol{u}^0_0, \ldots, \mathbf{0}, \mathbf{0}\right)^\top \Leftrightarrow \\
&\boldsymbol{A}_{i\setminus i_0} \cdot \left(\boldsymbol{z}^1_0, \boldsymbol{z}^1_1, \ldots, \boldsymbol{z}^{b-1}_0, \boldsymbol{z}^{b-1}_1, \boldsymbol{z}^b_{1-i_0}\right)^\top + (\mathbf{0}, \ldots, \boldsymbol{A}_{i_0}, \boldsymbol{G})^\top \boldsymbol{z}^b_{i_0} = \left(\boldsymbol{u}^0_0, \ldots, \mathbf{0}, \mathbf{0}\right)^\top \Leftrightarrow \\
&\boldsymbol{A}_{i\setminus i_0} \cdot \left(\boldsymbol{z}^1_0, \boldsymbol{z}^1_1, \ldots, \boldsymbol{z}^{b-1}_0, \boldsymbol{z}^{b-1}_1, \boldsymbol{z}^b_{1-i_0}\right)^\top = \left(\boldsymbol{u}^0_0, \ldots, -\boldsymbol{A}_{i_0}\boldsymbol{z}^b_{i_0}, -\boldsymbol{G}\boldsymbol{z}^b_{i_0}\right)^\top
\end{aligned}$$

where in our case $\boldsymbol{x} = \boldsymbol{z}^b_{i_0}$

Security. The security of VCWE comes in a straightforward way from the security of the [DKL$^+$23] LE scheme. In fact the security proof would be almost identical to [DKL$^+$23, theorem 5], with slight modifications. We formally state the security in the theorem below.

Theorem 8. *The Vector Commitment-Witness Encryption described above satisfies* VCWE *semantic security, under the LWE assumption.*

5.3 Efficiency and Comparison

Putting everything together, we compare the efficiency of the schemes obtained via our compiler, considering both our Robust and Efficient transformation (see Section 4.4), with the existing black-box RBE schemes. We summarize the comparison in Table 1 of the introduction Section 1.1.

In conclusion, our central RBE scheme from pairings with unbounded identity space and adaptive Compactness has the same efficiency properties as the one from [GKMR22], except for a 1.5λ overhead in ciphertext size. On the other hand, if we apply our efficient compiler we get only a 3× overhead in ciphertext size, albeit at the cost of having selective compactness.

For the Lattice-based construction, compared to [DKL$^+$23]'s, if one applies the selective compactness compiler, one gets a ciphertext that is $(2\lambda + 1)/4 \log n$ smaller. For example, for 1billion users this yields an $\approx 2\times$ improvement, while for moderate size number of $100,000$ users the improvement is $\approx 4\times$.

As for the construction from the [HLWW23] R-ABE, the differences are both quantitative and qualitative. First, the common reference string in [HLWW23] is quadratic therefore applying the tradeoff of between $|\mathsf{crs}|$ and $|\mathsf{pp}|$ [CES21, GKMR22] the best one can get is $|\mathsf{crs}| + |\mathsf{pp}| = O(\lambda n^{2/3} \log n)$ while in our case is $\sqrt{n} \log n$. Second, and more importantly, the R-ABE scheme of [HLWW23] uses bilinear groups of composite order where the factorization of the order should be unknown. Quantitatively, given the state of affairs in composite order bilinear groups, this leads to severe inefficiency both in running times of the algorithms and group elements' size. Qualitatively the group should be generated by a trusted third party, who afterwards erases the factors of the order of the group.[13]

6 Key-Value Map Commitments from Cuckoo Hashing and Vector Commitments

Given a key space $\mathcal{K}$ and a value space $\mathcal{V}$, a key-value map $\mathcal{M} \subseteq \mathcal{K} \times \mathcal{V}$ is a collection of pairs $(\mathsf{k}, \mathsf{v}) \in \mathcal{K} \times \mathcal{V}$. Key-value map commitments (KVC) [BBF19, AR20] are a cryptographic primitive that allows one to commit to a key-value map $\mathcal{M}$ in such a way that one can later open the commitment at a specific key, i.e., prove that (k, v) is in the committed map $\mathcal{M}$, and do so in a key-binding way. Namely, it is not possible to open the commitment at two distinct values $\mathsf{v} \neq \mathsf{v}'$ for the same key k. KVCs are a generalization of vector commitments [CF13]: while in VCs the key space is the set of integers $\{1, \ldots, n\}$, in a KVC the key space is usually a set of exponential size.

In this section, we present a construction of KVCs based on a combination of vector commitments and cuckoo hashing. The resulting KVC needs to fix at setup time a bound on the cardinality of the key-value map, but otherwise supports a key space and a value space of arbitrary sizes.

6.1 Definition of Key-Value Map Commitments

Given a key-value map $\mathcal{M}$, we write $(\mathsf{k}, \epsilon) \in \mathcal{M}$ to denote that $\mathcal{M}$ does not contain the key k.

Definition 11 (Key-Value Map Commitment). *A Key-Value Map Commitment* $\mathsf{KVM} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ *consists of the following algorithms:*

- $\mathsf{Setup}(1^\lambda, n, \mathcal{K}, \mathcal{V}) \to \mathsf{crs}$*: on input the security parameter* λ*, an upper bound* n *on the cardinality of the key-value maps to be committed, a key-space* $\mathcal{K}$*, and a value-space* $\mathcal{V}$*, the setup algorithm returns the common reference string* crs*.*
- $\mathsf{Com}(\mathsf{crs}, \mathcal{M}) \to (C, \mathsf{aux})$ *: on input a key-value map* $\mathcal{M} = \{(\mathsf{k}_1, \mathsf{v}_1), \ldots, (\mathsf{k}_m, \mathsf{v}_m)\} \subset \mathcal{K} \times \mathcal{V}$*, computes a commitment* C *and auxiliary information* aux*.*
- $\mathsf{Open}(\mathsf{crs}, \mathsf{aux}, \mathsf{k}) \to \Lambda$*: on input auxiliary information* aux *as produced by* Com*, and a key* $\mathsf{k} \in \mathcal{K}$*, the opening algorithm returns an opening* Λ*.*
- $\mathsf{Ver}(\mathsf{crs}, C, \Lambda, (\mathsf{k}, \mathsf{v})) \to b$ *: accepts (*i.e.*, outputs* $b \leftarrow 1$*) if* Λ *is a valid opening of the commitment* C *to the key* $\mathsf{k} \in \mathcal{K}$ *and value* $\mathsf{v} \in \mathcal{V} \cup \{\epsilon\}$*, else rejects (*i.e.*, outputs* $b \leftarrow 0$*).*

Intuitively, a KVC scheme should be correct in the sense that, for honest execution of the algorithms, an opening to a $(\mathsf{k}, \mathsf{v}) \in \mathcal{M}$ should correctly verify for a commitment to $\mathcal{M}$. While usual definitions for VCs consider perfect correctness, our work aims at also capturing constructions that have a negligible probability of failing correctness. To capture this, we introduce a strong notion called *robust correctness*, which essentially means that the expected correctness condition holds with overwhelming probability even for key-value maps that are adversarially chosen after seeing the public parameters. We note that such definition is strictly stronger than a ‘classical’ correctness definition that measures the probability over any choice of input but over the random and independent choice of the public parameters.

[13] In theoy, this is integrated in the trusted setup of the CRS generation. In practice, though, this type of CRS is highly undesirable, since no efficient MPC ceremony to generate it is currently known, in contrast to the ’powers-of-tau’ CRS.

Definition 12 (Robust Correctness). KVM *is* robust *if for any PPT $\mathcal{A}$ the following probability is overwhelming in λ:*

$$\Pr\left[\mathsf{Ver}(\mathsf{crs}, C, \mathsf{Open}(\mathsf{crs}, \mathsf{aux}, \mathsf{k}), (\mathsf{k}, \mathsf{v})) = 1 : \begin{array}{r} \mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, n, \mathcal{K}, \mathcal{V}) \\ (\mathcal{M}, \mathsf{k}, \mathsf{v}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ |\mathcal{M}| \leq n, (\mathsf{k}, \mathsf{v}) \in \mathcal{K} \times \mathcal{V} \cup \{\epsilon\} \\ (C, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{crs}, \mathcal{M}) \end{array}\right]$$

Definition 13 (Key-binding). KVM *is* key-value binding *if for any PPT $\mathcal{A}$:*

$$\Pr\left[\begin{array}{c} \mathsf{Ver}(\mathsf{crs}, C, \Lambda, (\mathsf{k}, \mathsf{v})) = 1 \\ \wedge\, \mathsf{Ver}(\mathsf{crs}, C, \Lambda, (\mathsf{k}, \mathsf{v}')) = 1 \\ \wedge\, \mathsf{v} \neq \mathsf{v}' \end{array} : \begin{array}{c} \mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, n, \mathcal{K}, \mathcal{V}) \\ (C, \mathsf{k}, \mathsf{v}, \Lambda, \mathsf{v}', \Lambda') \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] = \mathsf{negl}(\lambda)$$

Below we define an efficiency notion for KVCs, which aim to rule out "uninteresting" constructions, e.g., schemes where either commitments or openings have size linear in the size of the map or the key space. More formally,

Definition 14 (Efficient KVC). *A key-value map commitment* KVM *as defined above is* efficient *if for any* $\mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, n, \mathcal{K}, \mathcal{V})$*, any key-value map* $\mathcal{M} \subset \mathcal{K} \times \mathcal{V}$*, any* $(C, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{crs}, \mathcal{M})$*, any* $\mathsf{k} \in \mathcal{K}$ *and* $\Lambda \leftarrow \mathsf{Open}(\mathsf{crs}, \mathsf{aux}, \mathsf{k})$*, the bitsize of C and Λ is polylogarithmic in n, i.e., it is bounded by a fixed polynomial $p(\lambda, \log n)$.*

We give definitions for updatable Key-Value Map Commitments in appendix A.1, along with the corresponding robust correctness and efficiency notions.

6.2 KVM Construction from Cuckoo Hashing and Vector Commitments

We present a construction of a KVC for keys of arbitrary size. Our scheme is obtained by combining a Cuckoo Hashing scheme CH and a Vector Vommitment one VC. We refer to Section 2 for an intuitive description.

- $\mathsf{Setup}(1^\lambda, n, \mathcal{K}, \mathcal{V}) \rightarrow \mathsf{crs}$: runs $(\mathsf{pp}_{\mathsf{CH}}, \hat{\boldsymbol{T}}, \hat{S}) \leftarrow \mathsf{CH.Setup}(1^\lambda, n)$, and generates $\mathsf{crs}_{\mathsf{VC}} \leftarrow \mathsf{VC.Setup}(1^\lambda, N)$, then returns $\mathsf{crs} \leftarrow (\mathsf{crs}_{\mathsf{VC}}, \mathsf{pp}_{\mathsf{CH}})$.
- $\mathsf{Com}(\mathsf{crs}, \mathcal{M}) \rightarrow (C, \mathsf{aux})$: on input a key-value map $\mathcal{M} = \{(\mathsf{k}_i, \mathsf{v}_i)\}_{i=1}^m$:
  1. if there exists $i, j \in [m], i \neq j$ such that $\mathsf{k}_i = \mathsf{k}_j$, it aborts;
  2. $(\boldsymbol{T}, S) \leftarrow \mathsf{CH.Insert}(\mathsf{pp}_{\mathsf{CH}}, \hat{\boldsymbol{T}}, \hat{S}, \mathsf{k}_1, \ldots, \mathsf{k}_m)$; if $\boldsymbol{T} = \bot$ it aborts, else sets $\boldsymbol{T}' \leftarrow \mathbf{cat}(\boldsymbol{T})$;
  3. $(C_{\boldsymbol{T}}, \mathsf{aux}_{\boldsymbol{T}}) \leftarrow \mathsf{VC.Com}(\mathsf{crs}_{\mathsf{VC}}, \boldsymbol{T}')$;
  4. For $j = 1$ to m, let $\mathsf{ind}_j \in [N]$ be the index such that $\boldsymbol{T}'[\mathsf{ind}_j] = \mathsf{k}_j$. If ind_j exists, it sets $\boldsymbol{V}[\mathsf{ind}_j] \leftarrow \mathsf{v}_j$, otherwise, if $\mathsf{k}_j \in S$, adds $(\mathsf{k}_j, \mathsf{v}_j)$ to S^*.
  5. $(C_{\boldsymbol{V}}, \mathsf{aux}_{\boldsymbol{V}}) \leftarrow \mathsf{VC.Com}(\mathsf{crs}_{\mathsf{VC}}, \boldsymbol{V})$

  It return $C = (C_{\boldsymbol{T}}, C_{\boldsymbol{V}}, S^*)$ and $\mathsf{aux} = (\mathsf{aux}_{\boldsymbol{T}}, \mathsf{aux}_{\boldsymbol{V}}, S^*, \boldsymbol{T}', S, \mathcal{M})$.
- $\mathsf{Open}(\mathsf{crs}, \mathsf{aux}, \mathsf{k}) \rightarrow \Lambda$:
  1. $(\mathsf{ind}_1, \ldots, \mathsf{ind}_k) \leftarrow \mathsf{CH.Lookup}(\mathsf{pp}_{\mathsf{CH}}, \mathsf{k})$;
  2. if $\mathsf{k} \notin \{\boldsymbol{T}[\mathsf{ind}_1], \ldots, \boldsymbol{T}[\mathsf{ind}_k]\} \cup S$ aborts;
  3. for $j = 1$ to k: $\Lambda_j \leftarrow \mathsf{VC.Open}(\mathsf{crs}_{\mathsf{CH}}, \mathsf{aux}_{\boldsymbol{T}}, \mathsf{ind}_j)$.
  4. Let $\mathsf{ind}^* \in [N]$ be the index such that $\boldsymbol{T}[\mathsf{ind}^*] = \mathsf{k}$. If ind^* exists, it computes $\Lambda^* \leftarrow \mathsf{VC.Open}(\mathsf{crs}_{\mathsf{CH}}, \mathsf{aux}_{\boldsymbol{V}}, \mathsf{ind}^*)$, else sets $\Lambda^* = \bot$.

  Return $\Lambda = (\Lambda_1, \boldsymbol{T}[\mathsf{ind}_1], \ldots, \Lambda_k, \boldsymbol{T}[\mathsf{ind}_k], \Lambda^*)$.

- $\mathsf{Ver}(\mathsf{crs}, C, \varLambda, (\mathsf{k}, \mathsf{v})) \to b$: parses $C = (C_{\boldsymbol{T}}, C_{\boldsymbol{V}}, S^*)$ and $\varLambda = (\varLambda_1, t_1, \ldots, \varLambda_k, t_k, \varLambda^*)$ and proceeds as follows:
  1. $(\mathsf{ind}_1, \ldots, \mathsf{ind}_k) \leftarrow \mathsf{CH.Lookup}(\mathsf{pp}_{\mathsf{CH}}, \mathsf{k})$;
  2. for $j = 1$ to k: $b_j \leftarrow \mathsf{VC.Ver}(\mathsf{crs}_{\mathsf{CH}}, C_{\boldsymbol{T}}, \varLambda_j, \mathsf{ind}_j, t_j)$; if $\bigwedge_{j=1}^{k} b_j = 0$ outputs 0, else continues;
  3. if $\mathsf{k} \notin \{t_1, \ldots, t_k\}$, outputs 1 if and S^* is valid (i.e., it does not contain any repeated key and no entry (k, ϵ) and $(\mathsf{k}, \mathsf{v}) \in S^*$, else outputs 0.
  4. Otherwise, let j^* be the first index such that $\mathsf{k} = t_{j^*}$: it computes $b^* \leftarrow \mathsf{VC.Ver}(\mathsf{crs}_{\mathsf{CH}}, C_{\boldsymbol{V}}, \varLambda^*, \mathsf{ind}_{j^*}, \mathsf{v})$, and returns b^*.

Correctness and succinctness One can see by inspection that the proposed KVC scheme is robust (with overwhelming probability) under the assumption that VC is perfectly correct and that the cuckoo hashing scheme CH is robust. Succinctness of our KVC scheme is also easy to see by inspection, under the assumption that VC is succinct and that we use an instantiation of CH that satisfies $k = O(\log n)$.

Theorem 9 (Key binding). *If* VC *is position binding, then* KVM *is a key-binding KVC.*

Proof. Assume by contradiction that there exists a PPT adversary $\mathcal{A}$ that breaks the position binding of our KVC scheme. Then we show how to build a reduction $\mathcal{B}$ that breaks the position binding of VC. $\mathcal{B}$ takes as input $\mathsf{crs}_{\mathsf{VC}}$, generates the CH public parameters and runs $\mathcal{A}$ on input $\mathsf{crs} \leftarrow (\mathsf{crs}_{\mathsf{VC}}, \mathsf{pp}_{\mathsf{CH}})$.

Assume that $\mathcal{A}$ returns a tuple $(C, \mathsf{k}, \mathsf{v}, \varLambda, \tilde{\mathsf{v}}, \tilde{\varLambda})$ that breaks key binding with non-negligible probability. Let

$$\varLambda = (\varLambda_1, t_1, \ldots, \varLambda_k, t_k, \varLambda^*), \quad \tilde{\varLambda} = (\tilde{\varLambda}_1, \tilde{t}_1, \ldots, \tilde{\varLambda}_k, \tilde{t}_k, \tilde{\varLambda}^*)$$

By the winning condition of key binding we have that $\mathsf{v} \neq \tilde{\mathsf{v}}$ and that both opening proofs are accepted by the Ver algorithm. In particular, since Ver is invoked on the same key k, the first step of verification computes the same indices $\mathsf{ind}_1, \ldots, \mathsf{ind}_k$ in the verification of both $\varLambda$ and $\tilde{\varLambda}$.

First, notice that it must be the case that $\forall j \in [k]$, $t_j = \tilde{t}_j$. Otherwise, one can immediately break the VC position binding with the tuple $(C_{\boldsymbol{T}}, \mathsf{ind}_j, \varLambda_j, t_j, \tilde{\varLambda}_j, \tilde{t}_j)$.

Second, if $\mathsf{k} \notin \{t_1, \ldots, t_k\}$ then by construction of Ver (step 3), $\mathcal{A}$ cannot break position binding.

Finally, let j^* be the index such that $\mathsf{k} = t_{j^*}$. Then one can break the VC position binding with the tuple $(C_{\boldsymbol{V}}, \mathsf{ind}_{j^*}, \varLambda^*, \mathsf{v}, \tilde{\varLambda}^*, \tilde{\mathsf{v}})$. □

In Appendix A.2 we show that this KVC is updatable.

6.3 Key-Value Map Instantiations

We can instantiate the generic construction of the previous section with the CH scheme CH_λ from Theorem 1 (which is robust in the random oracle model) and any of the existing vector commitment schemes. If the VC has constant-size openings, say $O(\lambda)$, then the resulting KVC constructions have openings of size $O(\lambda^2)$. The most interesting implication of this KVC instantiation is that we obtain the first KVCs for unbounded key space based on pairings in a black-box manner. More in detail, we can obtain a variety of updatable KVCs according to which updatable VC we start from, e.g., we can use [CF13] to obtain a KVC based on CDH, [LY10] for one based on q-DHE. Prior to this work, an updatable KVC under these assumptions could only be obtained by instantiating the Merkle tree scheme with one of [LY10, CF13] VCs. However, Merkle trees with algebraic VCs need to make a non-black-box use of the underlying groups in order to map commitments back to the message space. In contrast, all our KVCs are black-box, if so are the underlying VC (as it is the case for virtually all existing schemes).

6.4 Accumulators from Vector Commitments with Cuckoo Hashing

It is easy to see that a KVC for a key space $\mathcal{K}$ immediately implies a universal accumulator [BP97, LLX07] for universe $\mathcal{K}$. The idea is simple: to accumulate $\mathsf{k}_1, \ldots, \mathsf{k}_n$ one commits to the key-value map $\{(\mathsf{k}_1, 1), \ldots, (\mathsf{k}_n, 1)\}$; a membership proof for k is an opening to $(\mathsf{k}, 1)$, and a non-membership proof is a KVC opening to (k, ϵ). The security of this construction (i.e., undeniability [Lip12] – the hardness of finding a membership and a non-membership proof for the same element) follows straightforwardly from key binding. Furthermore, if the KVC is updatable, the accumulator is updatable (aka dynamic, in accumulators lingo).

From this, we obtain new accumulators for large universe enjoying properties not known in prior work. For instance, we obtain the first dynamic accumulators for a large universe that are based on pairings in a black-box manner. To the best of our knowledge, prior black-box pairing-based accumulators either support a small universe [CKS09, CF13], or are not dynamic [Ngu05, KZG10]. Notably, using the CDH-based VC of [CF13] we obtain the first universal accumulator for a large universe that is dynamic, based on the CDH problem over bilinear groups, and black-box.

Acknowledgements We would like to thank Kevin Yeo for helpful feedback on the robustness of Cuckoo Hashing. The first two authors received funding from projects from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under project PICOCRYPT (grant agreement No. 101001283), from the Spanish Government under projects PRODIGY (TED2021-132464B-I00) and ESPADA (PID2022-142290OB-I00). The last two projects are co-funded by European Union EIE, and NextGenerationEU/PRTR funds.

References

ACLS18. Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy*, pages 962–979. IEEE Computer Society Press, May 2018.

ADW14. Martin Aumüller, Martin Dietzfelbinger, and Philipp Woelfel. Explicit and efficient hash families suffice for cuckoo hashing with a stash. *Algorithmica*, 70(3):428–456, 2014.

Ajt96. Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108. ACM Press, May 1996.

AR20. Shashank Agrawal and Srinivasan Raghuraman. KVaC: Key-Value Commitments for blockchains and beyond. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 839–869. Springer, Heidelberg, December 2020.

BBF19. Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 561–586. Springer, Heidelberg, August 2019.

Bd94. Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 274–285. Springer, Heidelberg, May 1994.

BGW05. Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 258–275. Springer, Heidelberg, August 2005.

BL20. Fabrice Benhamouda and Huijia Lin. Mr NISC: Multiparty reusable non-interactive secure computation. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 349–378. Springer, Heidelberg, November 2020.

BP97. Niko Bari and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 480–494. Springer, Heidelberg, May 1997.

CDG$^+$17. Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 33–65. Springer, Heidelberg, August 2017.

CDK$^+$22. Matteo Campanelli, Bernardo David, Hamidreza Khoshakhlagh, Anders Konring, and Jesper Buus Nielsen. Encryption to the future - A paradigm for sending secret messages to future (anonymous) committees. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part III*, volume 13793 of *LNCS*, pages 151–180. Springer, Heidelberg, December 2022.

CES21. Kelong Cong, Karim Eldefrawy, and Nigel P. Smart. Optimizing registration based encryption. In Maura B. Paterson, editor, *18th IMA International Conference on Cryptography and Coding*, volume 13129 of *LNCS*, pages 129–157. Springer, Heidelberg, December 2021.

CF13. Dario Catalano and Dario Fiore. Vector commitments and their applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 55–72. Springer, Heidelberg, February / March 2013.

CFG$^+$20. Matteo Campanelli, Dario Fiore, Nicola Greco, Dimitris Kolonelos, and Luca Nizzardo. Incrementally aggregatable vector commitments and applications to verifiable decentralized storage. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 3–35. Springer, Heidelberg, December 2020.

CFK22. Matteo Campanelli, Dario Fiore, and Hamidreza Khoshakhlagh. Witness encryption for succinct functional commitments and applications. Cryptology ePrint Archive, Report 2022/1510, 2022. `https://eprint.iacr.org/2022/1510`.

CFM08. Dario Catalano, Dario Fiore, and Mariagrazia Messina. Zero-knowledge sets with short proofs. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 433–450. Springer, Heidelberg, April 2008.

CKS09. Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 481–500. Springer, Heidelberg, March 2009.

CL02. Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76. Springer, Heidelberg, August 2002.

dCP23. Leo de Castro and Chris Peikert. Functional commitments for all functions, with transparent setup and from SIS. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 287–320. Springer, Heidelberg, April 2023.

DG17. Nico Döttling and Sanjam Garg. Identity-based encryption from the Diffie-Hellman assumption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 537–569. Springer, Heidelberg, August 2017.

DH76. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

DHMW22. Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wohnig. McFly: Verifiable encryption to the future made practical. Cryptology ePrint Archive, Report 2022/433, 2022. `https://eprint.iacr.org/2022/433`.

DKL$^+$23. Nico Döttling, Dimitris Kolonelos, Russell W. F. Lai, Chuanwei Lin, Giulio Malavolta, and Ahmadreza Rahimi. Efficient laconic cryptography from learning with errors. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 417–446. Springer, Heidelberg, April 2023.

DP23. Pratish Datta and Tapas Pal. Registration-based functional encryption. *Cryptology ePrint Archive*, 2023.

DW07. Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theoretical Computer Science*, 380(1-2):47–68, 2007.

ElG85. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

FFM$^+$23. Danilo Francati, Daniele Friolo, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Daniele Venturi. Registered (inner-product) functional encryption. *Cryptology ePrint Archive*, 2023.

FJ16. Alan M. Frieze and Tony Johansson. On the insertion time of random walk cuckoo hashing. *CoRR*, abs/1602.04652, 2016.

FMM09. Alan M. Frieze, Páll Melsted, and Michael Mitzenmacher. An analysis of random-walk cuckoo hashing. In *International Workshop and International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 2009.

FPS13. Nikolaos Fountoulakis, Konstantinos Panagiotou, and Angelika Steger. On the insertion time of cuckoo hashing, 2013.

FPSS03. Dimitris Fotakis, R. Pagh, Peter Sanders, and Paul G. Spirakis. Space efficient hash tables with worst case constant access time. *Theory of Computing Systems*, 38:229–248, 2003.

GGH$^+$13. Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

GGSW13. Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013.

GHM$^+$19. Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 63–93. Springer, Heidelberg, April 2019.

GHMR18. Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 689–718. Springer, Heidelberg, November 2018.

GKMR22. Noemi Glaeser, Dimitris Kolonelos, Giulio Malavolta, and Ahmadreza Rahimi. Efficient registration-based encryption. Cryptology ePrint Archive, Report 2022/1505, 2022. `https://eprint.iacr.org/2022/1505`.

GLW14. Craig Gentry, Allison B. Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 426–443. Springer, Heidelberg, August 2014.

GPV08. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

GV20. Rishab Goyal and Satyanarayana Vusirikala. Verifiable registration-based encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 621–651. Springer, Heidelberg, August 2020.

HLWW23. Susan Hohenberger, George Lu, Brent Waters, and David J Wu. Registered attribute-based encryption. In *Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part III*, pages 511–542. Springer, 2023.

JLS21. Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, pages 60–73. ACM Press, June 2021.

Kho13. Megha Khosla. Balls into bins made faster. In *Embedded Systems and Applications*, 2013.

KMW10. Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. *SIAM Journal on Computing*, 39(4):1543–1561, 2010.

Kus18. John Kuszmaul. Verkle trees: V(ery short m)erkle trees, 2018.

KZG10. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.

Lip12. Helger Lipmaa. Secure accumulators from euclidean rings without trusted setup. In Feng Bao, Pierangela Samarati, and Jianying Zhou, editors, *ACNS 12*, volume 7341 of *LNCS*, pages 224–240. Springer, Heidelberg, June 2012.

LLNW16. Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 1–31. Springer, Heidelberg, May 2016.

LLX07. Jiangtao Li, Ninghui Li, and Rui Xue. Universal accumulators with efficient nonmembership proofs. In Jonathan Katz and Moti Yung, editors, *ACNS 07*, volume 4521 of *LNCS*, pages 253–269. Springer, Heidelberg, June 2007.

LY10. Benoît Libert and Moti Yung. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 499–517. Springer, Heidelberg, February 2010.

Ngu05. Lan Nguyen. Accumulators from bilinear pairings and applications. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 275–292. Springer, Heidelberg, February 2005.

PPYY19. Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 79–93. ACM Press, November 2019.

PR04. Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

PR10. Benny Pinkas and Tzachy Reinman. Oblivious RAM revisited. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 502–519. Springer, Heidelberg, August 2010.

PSSZ15. Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In Jaeyeon Jung and Thorsten Holz, editors, *USENIX Security 2015*, pages 515–530. USENIX Association, August 2015.

Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

RSA78. Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, February 1978.

Sha84. Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 47–53. Springer, Heidelberg, August 1984.

Tsa22. Rotem Tsabary. Candidate witness encryption from lattice techniques. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 535–559. Springer, Heidelberg, August 2022.

VWW22. Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Witness encryption and null-IO from evasive LWE. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 195–221. Springer, Heidelberg, December 2022.

W$^+$17. Udi Wieder et al. Hashing, load balancing and multiple choice. *Foundations and Trends® in Theoretical Computer Science*, 12(3–4):275–379, 2017.

Wal22. Stefan Walzer. Insertion time of random walk cuckoo hashing below the peeling threshold, 2022.

Yeo23. Kevin Yeo. Cuckoo hashing in cryptography: Optimal parameters, robustness and applications. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 197–230, Cham, 2023. Springer Nature Switzerland.

A Updatable Key-Value Map Commitments

A.1 Definitions

Updatable KVCs We define *updatable* key-value map commitments as an extension of KVCs in which, akin to updatable VCs, one can efficiently update the commitment and the openings with respect to changes in the committed key-value map.

We model an update in a key-value map $\mathcal{M}$ as a pair (k, δ) where $\mathsf{k} \in \mathcal{K}$ is a key and δ is an update information which can be:

- $\delta = (\mathsf{insert}, \mathsf{v})$ to denote the insertion of the pair (k, v), i.e., $\mathcal{M}' \leftarrow \mathcal{M} \cup (\mathsf{k}, \mathsf{v})$;
- $\delta = (\mathsf{delete}, \mathsf{v})$ to denote the deletion of the pair (k, v), i.e., $\mathcal{M}' \leftarrow \mathcal{M} \setminus (\mathsf{k}, \mathsf{v})$;
- $\delta = (\mathsf{update}, \mathsf{v}, \mathsf{v}')$ to denote the change of value from v to v' associated to the key k, i.e., $\mathcal{M}' \leftarrow (\mathcal{M} \setminus (\mathsf{k}, \mathsf{v})) \cup (\mathsf{k}, \mathsf{v}')$.

Also, we say that (k, δ) is valid for $\mathcal{M}$ if the operation is well defined, namely: it inserts a key that is not present in the map, it deletes or changes the value of an already existing key.

Definition 15 (Updatable KVC). *A key-value map has* updatable commitments *if there exist two additional algorithms* $\mathsf{ComUpdate}$ *and* $\mathsf{ProofUpdate}$ *that work as follows.*

- $\mathsf{ComUpdate}(\mathsf{crs}, C, (\mathsf{k}, \delta), \mathsf{aux}) \to (C', \mathsf{aux}', \pi_\delta)$*: Given a commitment* C*, key* k*, update information* δ *and an auxiliary information* aux*, the algorithm outputs a new commitment* C'*, auxiliary information* aux'*, and update hint* π_δ*.*
- $\mathsf{ProofUpdate}(\mathsf{crs}, \Lambda_{\mathsf{k}}, (\hat{\mathsf{k}}, \delta), \pi_\delta) \to \Lambda'_{\mathsf{k}})$*: Given an opening* Λ_{k} *for some key* $\mathsf{k} \in \mathcal{K}$*, a key* $\hat{\mathsf{k}}$ *(possibly different from* k*), update information* δ *associated to* $\hat{\mathsf{k}}$*, and an update hint* π_δ*, the algorithm outputs a new opening* Λ'_{k}*.*

Robust Correctness of Updatable KVCs. *An updatable* KVM *is* robust *if for any public parameters* $\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, n, \mathcal{K}, \mathcal{V})$*, any adversarial choice of a key-value map* $\mathcal{M} \subseteq \mathcal{K} \times \mathcal{V}$ *of size* $\leq n$*,* $\mathsf{v} \in (\mathcal{V} \cup \{\epsilon\})$*,* $(\mathsf{k}, \mathsf{v}') \in \mathcal{K} \times (\mathcal{V} \cup \{\epsilon\})$*, and sequence of valid updates* $\{(\hat{\mathsf{k}}_j, \delta_j)\}_{j \in [m]}$ *that eventually yields a* $\mathcal{M}'$ *of size* $\leq n$ *such that* $(\mathsf{k}, \mathsf{v}') \in \mathcal{M}'$*, any initial commitment* $(C_0, \mathsf{aux}_0) \leftarrow$

$\mathsf{Com}(\mathsf{crs}, \mathcal{M})$ *and opening* $\Lambda_{\mathsf{k}}^{(0)} \leftarrow \mathsf{Open}(\mathsf{crs}, \mathsf{aux}_0, \mathsf{k})$*, and, sequentially, for* $j \in [m]$*, updated commitments and openings* $(C_j, \mathsf{aux}_j, \pi_{\delta j}) \leftarrow \mathsf{ComUpdate}(\mathsf{crs}, C_{j-1}, (\hat{\mathsf{k}}_j, \delta_j), \mathsf{aux}_{j-1})$ *and* $\Lambda_{\mathsf{k}}^{(j)} \leftarrow \mathsf{Proof Update}(\mathsf{crs}, \Lambda_{\mathsf{k}}^{(j-1)}, (\hat{\mathsf{k}}_j, \delta_j), \pi_{\delta j})$*, we have that* $\mathsf{Ver}(\mathsf{crs}, C_m, \Lambda_{\mathsf{k}}^{(m)}, (\mathsf{k}, \mathsf{v}')) = 1$ *holds with overwhelming probability in* λ.

Efficiency of Updatable KVCs. *An updatable KVC is said efficient if:*

(i) the efficiency definition of Definition 14 holds also for commitments and openings produced by $\mathsf{ComUpdate}$ *and* $\mathsf{ProofUpdate}$ *respectively. Additionally, the update hints* π_δ *produced by* $\mathsf{ComUpdate}$ *should also have polylogarithmic size.*

(ii) the runtimes of $\mathsf{ComUpdate}$ *and* $\mathsf{ProofUpdate}$ *are polylogarithmic in* n.

Our updateability notion for KVCs corresponds to what is known as *stateful updates* in the VC literature. This is due to the fact that $\mathsf{ComUpdate}$ requires knowledge of the auxiliary information aux related to the commitment C (which possibly contains the committed vector), and produces an update hint π_δ that can be used by anyone to update local proofs. This model is useful in applications where a central party can perform an update and enables everyone else to update proofs by publishing succinct information.

A.2 An Updatable Key-Value Map Construction from Cuckoo Hashing and Vector Commitments

Updatable KVC Here we show that the scheme described above is also updatable, defining the following algorithms:

- $\mathsf{ComUpdate}(\mathsf{crs}, C, (\mathsf{k}, \delta), \mathsf{aux}) \rightarrow (C', \mathsf{aux}', \pi_\delta)$: this algorithm initializes the following vectors: $\hat{\boldsymbol{T}}' \leftarrow \hat{\boldsymbol{T}}, \hat{\boldsymbol{V}}' \leftarrow \hat{\boldsymbol{V}}$, and computes:

– $(\mathsf{ind}_1, \ldots, \mathsf{ind}_k) \leftarrow \mathsf{CH.Lookup}(\mathsf{pp}_{\mathsf{CH}}, \mathsf{k})$.

1. if $\delta[1] \in \{\mathsf{delete}, \mathsf{update}\}$, then:
   (a) if looks for $\mathsf{ind} \in \{\mathsf{ind}_1, \ldots, \mathsf{ind}_k\}$ such that $\hat{\boldsymbol{T}}[\mathsf{ind}] = \mathsf{k}$. If there are several, it aborts, if there is none, then it looks for an element in $\hat{S}$ with first component k: if there are several or none it aborts; if there is one such element but its second component is not $\mathsf{v} = \delta[2]$, then it also aborts;
   (b) if $\delta[1] = \mathsf{delete}$, it removes the value in $\hat{\boldsymbol{T}}'[\mathsf{ind}]$ if ind existed, and else it removes (k, v) from $\hat{S}$;
   (c) else if $\delta[1] = \mathsf{update}$, it sets $\hat{\boldsymbol{V}}'[\mathsf{ind}] \leftarrow \mathsf{v}'$ (where $\mathsf{v}' = \delta[3]$) if ind existed, and else it removes (k, v) from S^* and adds $(\mathsf{k}, \mathsf{v}')$ in S^*;
2. if $\delta[1] = \mathsf{insert}$, then:
   (a) $(\boldsymbol{T}, S) \leftarrow \mathsf{CH.Insert}(\mathsf{pp}_{\mathsf{CH}}, \hat{\boldsymbol{T}}, \hat{S}, \mathsf{k})$. If $\boldsymbol{T} = \perp$, it aborts. Else it updates $\hat{\boldsymbol{T}}' \leftarrow \mathbf{cat}(\boldsymbol{T})$;
   (b) it finds $\mathsf{ind} \in \{\mathsf{ind}_1, \ldots, \mathsf{ind}_k\}$ such that $\hat{\boldsymbol{T}}'[\mathsf{ind}] = \mathsf{k}$. If there are several, it aborts, if there is none, if looks for k in S; if $\mathsf{k} \notin S$, then it aborts;
   (c) if ind existed, it sets $\hat{\boldsymbol{V}}'[\mathsf{ind}] \leftarrow \mathsf{v}$ (where $\mathsf{v} = \delta[2]$), else it adds (k, v) to S^*.
3. $\pi_{\boldsymbol{T}}$ is initialized as $\emptyset$. For each index i such that $\hat{\boldsymbol{T}}'[i] \neq \hat{\boldsymbol{T}}[i]$, it sequentially updates $(C_{\boldsymbol{T}}, \mathsf{aux}_{\boldsymbol{T}}) \leftarrow \mathsf{VC.ComUpdate}(\mathsf{crs}_{\mathsf{VC}}, C_{\boldsymbol{T}}, i, \hat{\boldsymbol{T}}[i], \hat{\boldsymbol{T}}'[i])$, and adds $(i, \hat{\boldsymbol{T}}[i], \hat{\boldsymbol{T}}'[i])$ to $\pi_{\boldsymbol{T}}$.
4. it initializes $\pi_{\boldsymbol{V}} \leftarrow \emptyset$. For each index i such that $\hat{\boldsymbol{V}}'[i] \neq \hat{\boldsymbol{V}}[i]$, it sequentially updates $(C_{\boldsymbol{V}}, \mathsf{aux}_{\boldsymbol{V}}) \leftarrow \mathsf{VC.ComUpdate}(\mathsf{crs}_{\mathsf{VC}}, C_{\boldsymbol{V}}, i, \hat{\boldsymbol{V}}[i], \hat{\boldsymbol{V}}'[i])$, and adds $(i, \hat{\boldsymbol{V}}[i], \hat{\boldsymbol{V}}'[i])$ to $\pi_{\boldsymbol{V}}$.
5. finally it returns: $C' \leftarrow (C_{\boldsymbol{T}}, C_{\boldsymbol{V}}, S^*)$, $\mathsf{aux}' \leftarrow (\mathsf{aux}_{\boldsymbol{T}}, \mathsf{aux}_{\boldsymbol{V}}, S^*, \hat{\boldsymbol{T}}, S, \mathcal{M})$, and $\pi_\delta \leftarrow (\pi_{\boldsymbol{T}}, \pi_{\boldsymbol{V}}, \hat{\boldsymbol{T}}[\mathsf{ind}_1], \ldots, \hat{\boldsymbol{T}}[\mathsf{ind}_k], \hat{S})$.

- $\mathsf{ProofUpdate}(\mathsf{crs}, \Lambda_{\mathsf{k}} = (\Lambda_1, t_1, \dots, \Lambda_k, t_k, \Lambda^*), (\hat{\mathsf{k}}, \delta), \pi_\delta = (\pi_{\boldsymbol{T}}, \pi_{\boldsymbol{V}}, t'_1, \dots, t'_k, S)) \to \Lambda'_{\mathsf{k}}$: this algorithm computes:
  1. $(\mathsf{ind}_1, \dots, \mathsf{ind}_k) \leftarrow \mathsf{CH.Lookup}(\mathsf{pp}_{\mathsf{CH}}, \hat{\mathsf{k}})$.
  2. if $\hat{\mathsf{k}} \notin \{t'_1, \dots, t'_k\} \cup S$, then it aborts;
  3. for each $j \in [k]$, for each $(i, t, t') \in \pi_{\boldsymbol{T}}$, it updates: $\Lambda_j \leftarrow \mathsf{VC.ProofUpdate}(\mathsf{crs}_{\mathsf{VC}}, \Lambda_j, i, t, t')$;
  4. it looks for $i^* \in [k]$ such that $t'_{i^*} = \hat{\mathsf{k}}$. If there are several such indices, it aborts. Else if i^* does not exist, it updates $\Lambda^* \leftarrow \perp$. Else if i^* exists and is unique, for each $(i, v, v') \in \pi_{\boldsymbol{V}}$, it sequentially updates $\Lambda^* \leftarrow \mathsf{VC.ProofUpdate}(\mathsf{crs}_{\mathsf{VC}}, \Lambda^*, i, v, v')$.
  5. finally it returns $\Lambda'_{\mathsf{k}} = (\Lambda_1, t'_1, \dots, \Lambda_k, t'_k, \Lambda^*)$.

One can remark that in the $\mathsf{ProofUpdate}$ algorithm, if the input $\hat{\mathsf{k}}$ is equal to the k of the input Λ_{k}, then $t'_1, \dots, t'_k$ will not be required in δ as they will be equal to the $t_1, \dots, t_k$ in Λ_{k}.

Robust Correctness We show robust correctness of the above updatable KVC, $\mathsf{KVM} = (\mathsf{Setup}$, Com, Open, Ver, $\mathsf{ComUpdate}$, $\mathsf{ProofUpdate})$, built from the Cuckoo Hashing scheme CH and the Vector Commitment scheme VC, if CH is robust and VC is correct.

Let n be an integer, λ a security parameter, $\mathcal{K}$ a key space and $\mathcal{V}$ a value space which are subspaces of the input space of VC's vectors, and: $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, n, \mathcal{K}, \mathcal{V})$. Let $\mathcal{A}$ be a PPT adversary, who chooses a key-value map $\mathcal{M} \subset \mathcal{K} \times \mathcal{V}$ of size n or less, $(\mathsf{k}, \mathsf{v}) \in \mathcal{K} \times \mathcal{V} \cup \{\epsilon\}$, $\mathsf{v}' \in \mathcal{V} \times \{\epsilon\}$, and a sequence of valid updates $\{(\hat{\mathsf{k}}_j, \delta_j)\}_{j \in [m]}$ that eventually yields the udpated key-value map $\mathcal{M}'$ of size n or less such that $(\mathsf{k}, \mathsf{v}') \in \mathcal{M}'$.

Let $(C_0, \mathsf{aux}_0) \leftarrow \mathsf{Com}(\mathsf{crs}, \mathcal{M})$, $\Lambda_k^{(0)} \leftarrow \mathsf{Open}(\mathsf{crs}, \mathsf{aux}, \mathsf{k})$, $(C_j, \mathsf{aux}_j, \pi_{\delta j}) \leftarrow \mathsf{ComUpdate}(\mathsf{crs}, C_{j-1}, (\hat{\mathsf{k}}_j, \delta_j), \mathsf{aux}_{j-1})$ and $\Lambda_{\mathsf{k}}^{(j)} \leftarrow \mathsf{ProofUpdate}(\mathsf{crs}, \Lambda_{\mathsf{k}}^{(j-1)}, (\hat{\mathsf{k}}_j, \delta_j), \pi_{\delta j})$.

We write this demonstration by induction. If there are no updates ($m = 0$), then $\mathcal{M}' = \mathcal{M}$, and since we required uniqueness of the keys in our key-value map construction, $\mathsf{v}' = \mathsf{v}$ is the value associated to k and $\mathsf{Ver}(\mathsf{crs}, C_0, \Lambda_{\mathsf{k}}^{(0)}, (\mathsf{k}, \mathsf{v})) = 1$ from the correctness of VC. Now, for the end of the induction, let us suppose that for some $i \in [0; m-1]$, and for $(\mathsf{k}, \mathsf{v}_i)$ the key-value pair of k in the $\mathcal{M}$ after update i, $\mathsf{Ver}(\mathsf{crs}, C_i, \Lambda_{\mathsf{k}}^{(i)}, (\mathsf{k}, \mathsf{v}_i)) = 1$.

Let $(\mathsf{k}, \mathsf{v}_{i+1})$ be the key-value pair for k in the map after update $i+1$. Then, the update hint $\pi_{\delta i+1}$, returned by $\mathsf{ComUpdate}$ under the robustness of CH, provides the changes from the key-value map after the i-th update, $\mathcal{M}_i$, to the one after the $(i+1)$-th update $(\hat{\mathsf{k}}_{i+1}, \delta_{i+1})$, $\mathcal{M}_{i+1}$. Running on this $\pi_{\delta i+1}$ with the same $(\hat{\mathsf{k}}_{i+1}, \delta_{i+1})$, $\mathsf{ProofUpdate}$ will also make Λ_{i+1} an opening for the map $\mathcal{M}_{i+1}$ in k. As $(\mathsf{k}, \mathsf{v}_{i+1})$ is the key-value pair for k after the $(i+1)$-th update, $\mathsf{Ver}(\mathsf{crs}, C_{i+1}, \Lambda_{\mathsf{k}}^{(i+1)}, (\mathsf{k}, \mathsf{v}_{i+1}))$ will thus be equal to 1 from the correctness of VC.

Efficiency We show that if VC and CH are efficient, and CH is robust, then KVM also is. Indeed, if they are, then crs is of polylogarithmic size, as well as commitments and openings under the efficiency of the VC. The auxiliary information input to $\mathsf{ComUpdate}$ is of polylogarithmic size, and the key and update information of constant size. Under the efficiency of CH, the output π_δ is polylogarithmic except with probability negligible in λ, as its size is at most in the number of elements moved by a $\mathsf{CH.Insert}$ operation, and in the stash size, which is also polylogarithmic if CH is robust. Under the efficiency of VC $\mathsf{ComUpdate}$ and $\mathsf{ProofUpdate}$ then both run in polylogarithmic time.