Attribute Based Encryption for RAMs from LWE

Prabhanjan Ananth [*] Xiong Fan [†]

Abstract

Attribute based encryption (ABE) is an advanced encryption system with a built-in mechanism to generate keys associated with functions which in turn provide restricted access to encrypted data. Most of the known candidates of attribute based encryption model the functions as circuits. This results in significant efficiency bottlenecks, especially in the setting when the function, associated with the ABE key, admits a RAM program whose runtime is sublinear in the length of the attribute. In this work we study the notion of attribute based encryption for random access machines (RAMs), introduced in the work of Goldwasser, Kalai, Popa, Vaikuntanathan and Zeldovich (Crypto 2013). We present a construction of attribute based encryption for RAMs satisfying sublinear decryption complexity assuming learning with errors. This improves upon the work of Goldwasser et al., who achieved this result based on SNARKs and extractable witness encryption.

En route to constructing this primitive, we introduce the notion of controlled homomorphic recoding (CHR) schemes. We present a generic transformation from controlled homomorphic recoding schemes to attribute-based encryption for RAMs and then we show how to instantiate controlled homomorphic recoding schemes based on learning with errors.

1 Introduction

Attribute based encryption [SW$^+$05] is a powerful paradigm that provides a controlled access mechanism to encrypted data. Unlike a traditional encryption scheme, in an attribute based encryption scheme, an authority can generate a constrained key sk_P for a program P such that it can decrypt an encryption of message μ, associated with attribute x, only if the condition $P(x) = 0$ is satisfied. The last decade of research in this area [SW$^+$05, GPSW06, OSW07, GJPS08, W$^+$09, LW11, Wat12, GVW15a, GGH$^+$13, GKP$^+$13b, BGG$^+$14, GGHZ14, Wee14, GVW15b, BV16] has led to several useful applications including verifiable computation [PRV12] and reusable garbled circuits [GKP$^+$13a]. Special cases of ABE, such as identity based encryption [BF01, Wat05, DG17, BLSV17], and generalizations of ABE, such as FE [BSW11, O'N10, GGH$^+$16], have also been extensively studied.

Current known constructions of ABE offer different flavors of efficiency guarantees and can be based on various cryptographic assumptions. Barring few expections, all these constructions [GPSW06, W$^+$09, LOS$^+$10, GVW15a, BGG$^+$14, GVW15b] model the random access programs, associated with the constrained keys, as circuits. However, transforming random access programs into circuits is associated with significant efficiency costs. If the execution time of these programs were sub-linear

*CSAIL, MIT, Email: prabhanjan@csail.mit.edu. Part of the work done while interning at IBM T.J. Watson Center.

†Cornell University, Email: xfan@cs.cornell.edu. Part of the work done while interning at IBM T.J. Watson Center. This material is based upon work supported by IBM under Agreement 4915013672. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the sponsors.

in the input length to begin with (for instance, binary search), modeling them as circuits destroys the sub-linearity property. As a consequence, the decryption complexity could be exponential in the running time of the programs. This is quite unsatisfactory as we often encounter scenarios where sublinear computations have to be performed on massive data sets. Even if the programs do not have sublinear complexity in the input length, another issue with modeling programs as circuits is that the decryption algorithm could be drastically slower than the running time of the original programs, not to mention the additional overhead involved in transforming programs into circuits.

To circumvent these issues, Goldwasser et al. [GKP+13b] introduced the notion of ABE for RAMs (A RAM program is associated with a memory (initialized with the input to the RAM program) and step circuit. In every step of the RAM computation, the step circuit outputs the next index to be read and additionally, it also writes to a location in the memory. It differs from a Turing machine, in that a RAM program does not have to read all the locations in the memory.). In this setting, the program P associated with attribute keys are represented as RAM programs, which are a more natural model of computation than boolean circuits. They presented the first construction of ABE for RAMs assuming extractable witness encryption and SNARKs. They achieved decryption complexity polynomial only in the running time of the program. Recent works [GGHW14, BP14, BSW16] have brought into question the veracity of the assumptions of extractable witness encryption and SNARKs. While the existence of these assumptions have been ruled out only in specific scenarios, they certainly guide us to be more careful about using them for cryptographic applications.

1.1 Our Contributions

The goal of this work is to base the primitive of ABE for RAMs on well studied cryptographic assumptions. Before stating our result, we explain the model of ABE for RAMs below.

As defined in an ABE for circuits scheme, an ABE for RAMs scheme consists of setup, key generation, encryption and decryption algorithms. The encryption algorithm takes as input an attribute database D, a message μ and produces the ciphertext ct. The key generation takes as input a RAM program P and produces attribute key sk_P associated with P. The decryption algorithm, modeled as a RAM program, takes as input sk_P, a ciphertext ct and produces the decrypted message μ only if $P^D = 0$. The key efficiency requirement on the scheme is that the decryption of sk_P on encryption of μ should take time $p(\lambda, T)$, where T is an upper bound on the running time of P, for a fixed polynomial $p(\cdot)$. In particular, if T is polylogarithmic in length $|D|$ of the attribute then the decryption complexity is also polylogarithmic in $|D|$. We term this *sublinear decryption* property. Barring the work of Goldwasser et al. [GKP+13b], none of the ABE constructions achieve sublinear decryption complexity property.

We show the following result:

Theorem 1.1 (Informal)**.** *Assuming learning with errors (with sub-exponential modulus[1]), there is a construction of public key attribute-based encryption scheme for random access machines satisfying sub-linear decryption property.*

Our construction satisfies selective security (Using the work of [GKW16], we can boost our security to semi-adaptive security. However, this transformation would not preserve the sub-linear decryption property.). Even for programs that do not run in time sublinear in the input length, our work beats previous works in terms of decryption complexity. The decryption complexity in

[1]The same assumption for lattice-based ABE for general circuits [BGG+14].

our work is a polynomial $p(\lambda, T)$, while previous works [GVW15a, BGG$^+$14, GVW15b] achieved decryption complexity $p(\lambda, T^3)$ (polynomial p being the same), where T^3 is the depth of the circuit obtained by transforming a RAM program of runtime at most T [CR72, PF79].

To prove the above theorem, we introduce a novel primitive that we call controlled homomorphic recoding schemes. This primitive generalizes the concepts of fully key homomorphic encryption, introduced in the work of [BGG$^+$14]. Using this tool, we build ABE for RAMs and then we conclude by instantiating the tool from lattice assumptions.

In our scheme, in addition to the decryption complexity, the rest of the parameters in our system also depend on the upper bound on the running time. The decryption complexity itself can be made input-dependent, and hence independent of the upper time bound, using powers-of-two technique introduced by [GKP$^+$13b]. However, it is unclear how to make the encryption and the key generation complexity independent of the time bound in our scheme. In contrast, the scheme of Goldwasser et al. [GKP$^+$13b] achieve *succinctness* property, meaning that the encryption complexity and the key generation complexity is independent of the time bound. A natural question to ask here is whether we can achieve succinctness property without resorting to stronger assumptions. It turns out that an attribute based encryption satisfying succinctness property would imply succinct randomized encodings. This is because, attribute based encryption for RAMs satisfying succinctness, additionally assuming learning with errors, imply succinct randomized encodings for Turing machines[2] [BGJ$^+$16, AJS15]. Current constructions of succinct randomized encodings are based on indistinguishability obfuscation [CHJV15, BGL$^+$15, KLW15] for circuits.

1.2 Technical Overview

We first discuss the hurdles involved in extending the current known attribute based encryption for circuits schemes to the RAM setting. In the ABE for circuits scheme of Boneh et al. [BGG$^+$14], the public key consists of matrices $\mathbf{A}, \mathbf{A}_1, \ldots, \mathbf{A}_N \in \mathbb{Z}_q^{n\times m}$. The encryption of an attribute $D = (x_1, \ldots, x_N)$ and message μ produces the ciphertext consisting of,

$$\boldsymbol{s}^\mathsf{T}\mathbf{A} + \boldsymbol{e}^\mathsf{T},\ \boldsymbol{s}^\mathsf{T}(\mathbf{A}_1 + x_1\mathbf{G}) + \boldsymbol{e}_1^\mathsf{T},\ \ldots,\ \boldsymbol{s}^\mathsf{T}(\mathbf{A}_N + x_N\mathbf{G}) + \boldsymbol{e}_N^\mathsf{T},\ \mathsf{Enc}(\mathsf{sk}, \mu)$$

where $\boldsymbol{s} \in \mathbb{Z}_q^n$ is a randomly chosen secret vector, $\mathbf{G}$ is the gadget matrix [MP12], $\boldsymbol{e}, \{\boldsymbol{e}_i\}$ are error vectors (chosen from an appropriate Gaussian distribution) and Enc is a symmetric encryption scheme[3] that allows for decrypting using "noisy" keys. In particular, given $\mathsf{sk} + \mathsf{err}$, where err has small norm, we can distinguish $\mathsf{Enc}(\mathsf{sk}, 0)$ and $\mathsf{Enc}(\mathsf{sk}, 1)$. An attribute key corresponding to a RAM program P is computed as follows: first transform the program P into a circuit C. Next, homomorphically evaluate C on the matrices $\mathbf{A}_1, \ldots, \mathbf{A}_\ell$ to obtain the matrix $\mathbf{A}_\mathsf{C}$. Finally, the attribute key consists of the trapdoor $\mathbf{T}_\mathsf{C}$ such that the following holds: $[\mathbf{A}|\mathbf{A}_\mathsf{C}] \cdot \mathbf{T}_\mathsf{C} = \mathsf{sk}$. The decryption consists of two steps: (i) *homomorphism step*: in this step, evaluate the ciphertexts $\{\boldsymbol{s}^\mathsf{T}(\mathbf{A}_i + x_i\mathbf{G}) + \boldsymbol{e}_i^\mathsf{T}\}$ to obtain the ciphertext that is approximately $\boldsymbol{s}^\mathsf{T}(\mathbf{A}_\mathsf{C} + \mathsf{C}(D)\mathbf{G})$, (ii) *authentication step*: in this step, we use the homomorphically computed ciphertext and the trapdoor $\mathbf{T}_\mathsf{C}$ to obtain a noisy secret key sk only if $\mathsf{C}(D) = 0$. The noisy key then allows us to obtain the message μ.

Notice that the attribute key T_C is generated as a function of the matrices $\{\mathbf{A}_i\}$ and circuit C. This means key generation algorithm knows all the operations, specified by the circuit C, that is to be performed on the data and thus can authenticate only those operations that are legal. As

[2]The works [BGJ$^+$16, AJS15] show implication of ABE for Turing machines (as defined in [AJS15]) to succinct randomized encodings (Appendix A.5 in [BGJ$^+$16]. However, ABE for RAMs satisfying succinctness property implies ABE for Turing machines.

[3]Boneh et al. use a specific lattice based symmetric encryption scheme.

an example, consider a circuit that consists of applying OR gates to its input and then applying a giant AND gate at the top. At the time of generating the key for this circuit, the authority knows that first applying OR and then AND is the only legal computation path that can be taken and it can thus generate a trapdoor that only authenticates this computation. However, if we were to generate attribute keys for RAM programs directly then we would run into trouble. The operations performed during RAM computation can be highly *data-dependent* (unlike circuits, which consist of data-oblivious operations) and hence it is unclear which set of operations to authenticate during the key generation process. For instance, a RAM program P could read the first bit of the database and if its value is 0 it executes a sequence of OR gates and then applies a giant AND gate, otherwise if its value is 1 then it could simply output the second bit of the database. This means that the computation path, i.e., a sequence of operations to be performed on the data, is ill-defined during the key generation phase and hence its unclear how to execute the authentication mechanism.

A first attempt to solve the above issue is enumerate all possible computation paths and then generate a trapdoor for every computation path. In more detail, let T be an upper bound on the running time of the program and for now, think of T as being a constant. This means that all possible T-sized subsets of the memory locations can be accessed by the program during decryption. For every possible T-sized set $I \subseteq [N]$, we first perform homomorphic evaluation on the matrices $\{\mathbf{A}_i\}_{i \in I}$ to obtain the matrix $\mathbf{A}_I$. The next step is to generate a trapdoor $\mathbf{T}_I$ such that $[\mathbf{A}|\mathbf{A}_I] \cdot \mathbf{T}_I = \mathsf{sk}$. Since T is a constant, the size of the attribute key is polynomial sized, as desired. On input an encryption of attribute D and message μ, first determine the set of locations I^* accessed by the program. Then use the trapdoor $\mathbf{T}_{I^*}$ to obtain the noisy key and decrypt the message as before. This scheme achieves sublinear decryption complexity: the decryption algorithm only needs to touch ciphertext encodings computed with respect to $\{\mathbf{A}_i\}_{i \in I^*}$ and trapdoor $\mathbf{T}_{I^*}$. However, in terms of security, this scheme fails. There is no mechanism in place that prevents a malicious evaluator from illegally using a trapdoor $\mathbf{T}_{I'}$, for $I' \neq I^*$. This suggests that we need a *controlled* authentication mechanism that lets us evaluate only "legal" trapdoors depending on the data. Moreover, even if we tweak the scheme to incorporate this mechanism, a bigger problem is that this does not scale for the case when T is not a constant since the attribute key would no longer be polynomial sized. We introduce the notion of controlled homomorphic recoding schemes that overcomes the above barriers.

Our Approach: Controlled Homomorphic Recoding Scheme. The main insight in our approach is to divide the computation into several tiny modules of computation and then apply authentication mechanism after the execution of every module. A RAM program presents a natural way to achieve such a modularization: a module corresponds to the associated step circuit of the RAM program. As in the case of [BGG+14], the encryption will consist of encodings of the database. We design the decryption process to proceed by homomorphically evaluating the step circuit followed by authenticating its output which then is followed by homomorphic evaluation of the step circuit for next time step and so on. In order to perform authentication after every time step, we provide T auxiliary keys as part of the attribute key, where T is the maximum running time of the associated RAM program. The main challenge we face when we try to nail down this approach is that we need a mechanism to 'stitch' the intermediate homomorphism and the authentication steps together. Specifically the authentication phase should not only verify the correctness of the computation of the step circuit but it should also pass along the valid encoded information to the next homomorphism phase. We term this phase, that performs the job of both authentication and translation of encodings, as controlled recoding phase. Incorporating both the homomorphism phase and the controlled recoding phase, we introduce the notion of controlled homomorphic recoding scheme.

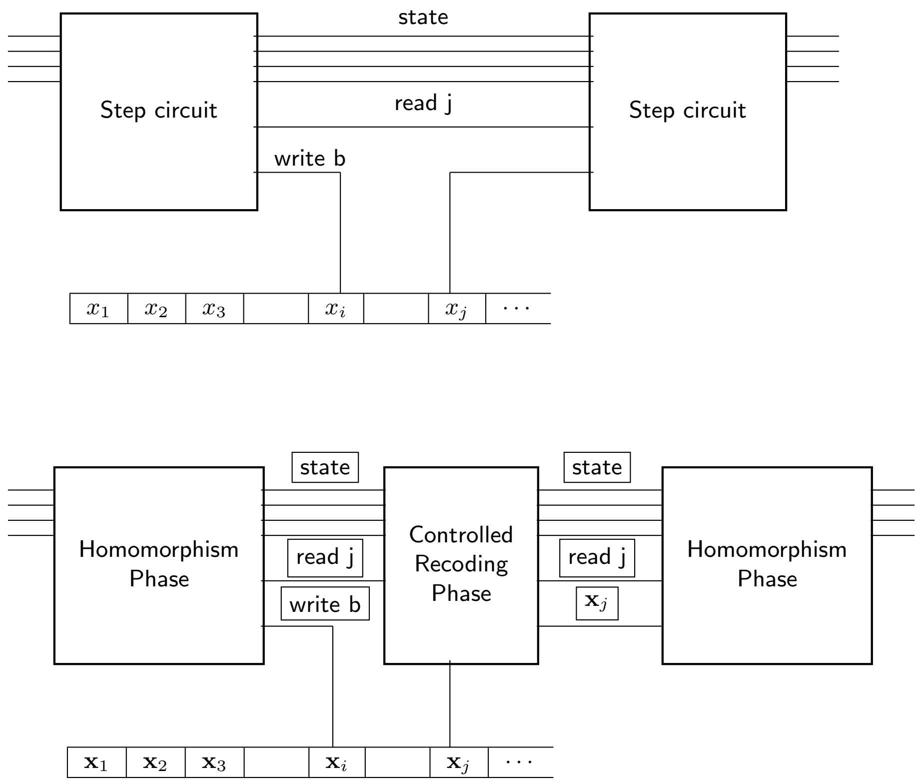

Figure 1: A high level description of how the two phases (homomorphism and controlled recoding) mirror the execution of the RAM program. $\mathbf{x}_i$ denotes the encoding of x_i. The controlled recoding phase translates the encodings of state and "read j" instruction from the previous time step to the next time step. It also translates the j^{th} database encoding into an encoding for the next time step.

A controlled homomorphic recoding scheme allows for encoding messages x along with secret randomness $\mathbf{s}$ with respect to public key pk. There are two main phases associated with a controlled homomorphic recoding scheme.

- Public Homomorphism: Given encodings of messages $\{x_i\}_{i\in[\ell]}$ along with secret randomness $\mathbf{s}$ computed with respect to public keys $\{\mathsf{pk}_i\}_{i\in[\ell]}$ and a boolean circuit C, it outputs the encoding of $C(x_1,\ldots,x_\ell)$ along with $\boldsymbol{s}$ with respect to the public key $C(\mathsf{pk}_1,\ldots,\mathsf{pk}_\ell)$. In particular, the homomorphism phase can only be applied on encodings computed with respect to the same secret randomness $\mathbf{s}$.

- Controlled Recoding Phase: The main goal of this phase is two-fold: first verify the computation in the previous time step and if the verification phase succeeds then produce encodings for the next homomorphism phase. The verification step implicitly captures the controlled authentication mechanism that we touched upon earlier. We describe the inner workings of this phase below.

  Given encodings of messages $\{x_i\}_{i\in[\ell]}$ along with secret randomness $\boldsymbol{s}$ computed with respect to public keys $\{\mathsf{pk}_i\}_{i\in[\ell]}$ and a recoding key (The term recoding key is inspired from the work of [GVW15a]. As in [GVW15a], the recoding key in our work serves the purpose of re-encryption.)

rk, it outputs the encoding of $f(x_1, \ldots, x_\ell)$ with respect to the target public key pk^*, as long as $f(x_1, \ldots, x_\ell) \neq \perp$. That is, if the output of f being $\perp$ signals then the recoding process fails. The function f, target public key pk^* and a secret key associated with one of the public keys in $\{\mathsf{pk}_i\}_{i \in [\ell]}$ are used to compute the recoding key rk.

Looking ahead, in the construction of ABE from CHR, the control functions will be critical in controlling the information to be passed on from the output of step circuit in the i^{th} step to $(i+1)^{th}$ step.

In more detail, we describe the algorithms associated with a controlled homomorphic recoding scheme. Setup generates the public key pk and secret key sk. Enc is a mechanism to transform attribute y and secret message s into a ciphertext ct. Equality test EqTest allows for checking if two different ciphertexts ct_1 and ct_2 encode the same attribute bit, given the condition that they both are computed with respect to the same public key pk and the same secret message s. The rest of the algorithms are classified into public homomorphism and controlled recoding phases.

Public Homomorphism: There are two algorithms associated with this phase. The first algorithm KeyEval takes as input many public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, circuit C and outputs a homomorphically evaluated public key pk_C. The second algorithm, takes as input ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$ with ct_i computed under pk_i, circuit C and it computes the ciphertext ct^* under the resulting public key $C(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$. Looking ahead, C will essentially represent the step circuit of a RAM program. We present a pictorial representation of both these algorithms in Figure 2.

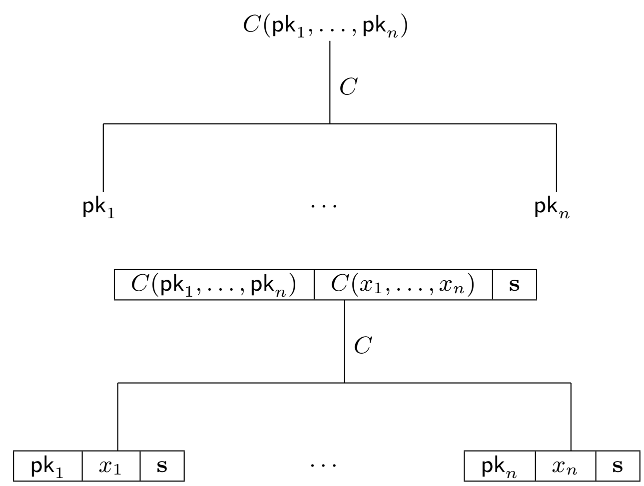

Figure 2: Description of homomorphism algorithms. The topmost figure denotes the execution of KeyEval and the next figure denotes the execution of CtEval.

Controlled Recoding: There are two algorithms associated with the controlled recoding phase. The ciphertext recoding procedure ReEnc allows for recoding ciphertexts of $\{x_i\}_{i \in [n]}$ under public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ into a ciphertext of $f(x_1, \ldots, x_n)$ under the public key pk^* as long as $f(x_1, \ldots, x_n) \neq \perp$. This recoding process is carried out with the help of a recoding key rk_f, which is associated with a functon f. The recoding key generation algorithm ReEncKG allows for generating such a recoding key, rk_f. Looking ahead, the function f will play a crucial role in deciding which of the ciphertexts to recode.

Consider the following example. Let rk_f be a recoding key that recodes ciphertexts under public keys $(\mathsf{pk}_1, \mathsf{pk}_{100})$, where f is a function that takes as input (x, y) and outputs y if $x = 100$, otherwise it outputs $\perp$. This is useful for the reading operation in the ABE application. We can think of the public key pk_1 being used to encode the read address 100 and pk_{100} used to encode the value in the 100^{th} memory location.

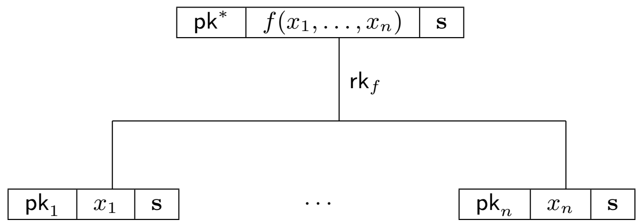

Figure 3: Description of ciphertext recoding, ReEnc.

We explain the correctness requirement by considering a toy example. Consider three input bits (x_1, x_2, x_3), circuits C_1, C_2, and control function f such that $f\big(C_1(x_1, x_2, x_3), C_2(x_1, x_2, x_3)\big) \in \{0, 1\}$.

- Suppose $\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3$ are encodings of (x_1, x_2, x_3) (under the same randomness) respectively under the public keys $(\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3)$.
- Homomorphically evaluating $(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3)$ using the circuit C_1 (resp., C_2) yields ciphertext of $C_1(x_1, x_2, x_3)$ (resp., $C_2(x_1, x_2, x_3)$) under the public key $C_1(\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3)$ (resp., $C_2(\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3)$). Call these two encodings ct'_1 and ct'_2.
- Suppose rk_f is a recoding key that translates ciphertexts encoded under the public keys $\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3$ into a ciphertext under public key pk^*. Upon executing $\mathsf{ReEncKG}$ on input $\mathsf{ct}'_1, \mathsf{ct}'_2$ and recoding key rk_f, let ct^* be the resulting ciphertext.

We require the following condition to hold: EqTest should declare ct^* and ct equal, as long as ct is a ciphertext of $f(C_1(x_1, x_2, x_3), C_2(x_1, x_2, x_3))$ and secret randomness s under the public key pk^*.

Toy Example: ABE for Circuits. Before we show how to construct ABE for RAMs from CHR, it is instructive to look at how ABE for circuits can be constructed from CHR. We introduce the security properties of CHR that will be useful to prove the security of the resulting ABE for circuits scheme. These security properties will be reused later in the proof of security of the final ABE for RAMs scheme.

We describe the high level construction of ABE for circuits.

- **Setup**: Let N be the length of the attribute. There are three types of CHR public keys to be generated: (i) public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$ corresponding to the attribute, (ii) anchor public key-secret key pair $(\mathsf{pk}_0, \mathsf{sk}_0)$, (iii) target public key $\mathsf{pk}_{\mathsf{out}}$. The anchor secret key sk_0 will be set as the master secret key of the ABE scheme. The public key of the ABE scheme will be set to $(\mathsf{pk}_0, \mathsf{pk}_{\mathsf{out}}, \mathsf{pk}_1, \ldots, \mathsf{pk}_N)$.
- **Key Generation**: Let C be the circuit for which we need to generate the attribute key. Execute the following two steps:
  - Homomorphism keys: Execute the key evaluation algorithm $\mathsf{KeyEval}$ of CHR on the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ and circuit C to obtain the public key pk_C.

- Controlled Recoding keys: Execute the key recoding algorithm $\mathsf{ReEncKG}$ of CHR on input public keys $(\mathsf{pk}, \mathsf{pk}_C)$, target public key $\mathsf{pk}_{\mathsf{out}}$ and control function f (defined next) to obtain the recoding key rk_f. The function f is defined as $f(x, y) = 0$ if it holds that $x = 0$ and $y = 0$, otherwise $f(x, y) = \perp$.

Set the ABE key of C to be rk_f.

- **Encryption**: It takes as input attribute x of size N and message μ. It first samples secret randomness $\mathbf{s}$ from a distribution[4]. It computes the following encodings: (i) encoding of x_i and $\mathbf{s}$ under the public key pk_i, (ii) encoding of 0 and $\mathbf{s}$ under the public key pk_0 and finally, (iii) encoding ct^* of μ and $\mathbf{s}$ under the public key $\mathsf{pk}_{\mathsf{out}}$.

- **Decryption**: This proceeds in two phases:

  - Homomorphism: The circuit C is homomorphically evaluated on the encodings of x_i (and $\mathbf{s}$) using the algorithm CtEval of the CHR scheme. The result is an encoding of $C(x)$ (and $\mathbf{s}$) under pk_C.
  - Controlled Recoding: Using the recoding key rk_f, the encodings of $C(x)$ under pk_C and 0 under pk_0 (with respect to same randomness $\mathbf{s}$) can be translated into an encoding of 0 and $\mathbf{s}$ under $\mathsf{pk}_{\mathsf{out}}$ *as long as* $C(x) = 0$. Recall that this is because the recoding process only succeeds if the output of the function is not $\perp$ and this in turn only happens if $C(x) = 0$.

  Once we have an encoding of 0 and $\mathbf{s}$ under $\mathsf{pk}_{\mathsf{out}}$, we then run the equality test on this encoding and ct^* (as computed in encryption). If they are equal, this means that the secret message μ has to be 0, otherwise it has to be 1. This concludes the scheme.

We now describe the security properties of CHR required to argue the security of this scheme. These security properties will in fact be reused later on in the proof of security of ABE for RAMs from CHR. We draw parallels of the security properties stated below with the security proof of ABE for circuits construction of Boneh et al. [BGG+14].

I. Indistinguishability of Setup: To define this property, we first define a simulator $\mathsf{Sim.CHRSetup}$ that takes as input a value v to be programmed and outputs a public key $\mathsf{Sim.pk}$ and a secret trapdoor τ[5]. We require that the distribution of simulated public keys $\{\mathsf{Sim.pk}\}$ is indistinguishable to the distribution of real public keys $\{\mathsf{pk}\}$.

This is analogous to the hybrid in the security proof of Boneh et al. [BGG+14], where the (honestly generated) public keys correspond to matrices of the form $\{\mathbf{A}\}$ and the simulated public keys are of the form $\{\mathbf{AR} - v\mathbf{G}\}$ and $\mathbf{R}$ is the trapdoor. In the case of Boneh et al., the indistinguishability follows from an application of leftover hash lemma.

II. Indistinguishability of Recoding Keys: The simulator $\mathsf{Sim.CHR}_{\mathsf{rk}}$ is defined as follows: let the public keys used to encode the database be simulated as $(\mathsf{Sim.pk}_1, \ldots, \mathsf{Sim.pk}_n)$, where the i^{th} attribute bit x_i is programmed in $\mathsf{Sim.pk}_i$. As long as the output of C on x is *not 0*, the recoding key rk_f associated with the attribute key of C, can be generated *without the secret key* sk_0.

The indistinguishability of recoding keys property states that the distribution of honestly generated recoding keys is indistinguishable from the distribution of simulated recoding keys.

[4]The security of the scheme will depend on which distribution we choose. Looking ahead, this corresponds to the distribution used to sample the secret in the learning with errors assumption.

[5]This is not the same as lattice trapdoors which we use in the instantiation.

Recall that in [BGG$^+$14], the trapdoor T_C was generated (in the construction) such that $[\mathbf{A}|\mathbf{A}_C] \cdot \mathbf{A} = \mathbf{P}$. In our language, $\mathbf{A}$ corresponds to pk_0, $\mathbf{P}$ corresponds to $\mathsf{pk}_{\mathsf{out}}$ and $\mathbf{A}_C$ corresponds to pk_C. Furthermore, T_C can be viewed as a recoding key. In the construction of [BGG$^+$14], the trapdoor $\mathsf{T}_{\mathbf{A}}$ (a.k.a sk_0) for $\mathbf{A}$ is used to generate T_C. However, in the security proof, when the output of C on the challenge attribute x^* is 1, the matrix $[\mathbf{A}|\mathbf{A}_C]$ can be written as $[\mathbf{A}|\mathbf{A}R_C - \mathbf{G}]$ and the trapdoor of $\mathbf{G}$ (along with knowledge of R_C) helps in simulating the trapdoor R_C *without the trapdoor* $\mathsf{T}_{\mathbf{A}}$.

III. Pseudorandomness of Ciphertexts: This property states that the encoding of b and secret randomness $\mathbf{s}$, under a public key pk, is indistinguishable from uniform distribution on the space of encodings.

This is analogous to the hybrid in [BGG$^+$14], where the assumption of learning with errors is invoked. In more detail, they replace the encoding $\mathbf{s}^{\mathsf{T}}\mathbf{A} + e^{\mathsf{T}}$ (encoding of 0 and $\mathbf{s}$ under pk_0) with uniform random distribution. This can be performed only if the trapdoor $\mathsf{T}_{\mathbf{A}}$ is no longer used to generate the attribute keys.

Using the above security properties, we give the overview of the proof of security of ABE for circuits from CHR.

- Suppose the adversary has submitted the challenge attribute x^* along with attribute key queries $C_1, \ldots, C_q$ such that $C(x^*) = 1$. Using $\mathsf{Sim.CHRSetup}$, the first step is to simulate the public keys $(\mathsf{Sim.pk}_1, \ldots, \mathsf{Sim.pk}_n)$ with the i-th attribute bit x_i^* programmed in $\mathsf{Sim.pk}_i$.
- Next, all the attribute keys are simulated using the algorithm $\mathsf{Sim}_{\mathsf{rk}}$. Here, the fact that $C_i(x^*) = 1$, for every $i \in [q]$, is crucially used.
- At this point, the secret key sk_0 is not used and therefore, the pseudorandomness of ciphertexts property is invoked to compute all the encodings using uniform distribution.

Main Construction: ABE for RAMs from CHR. We now show how to construct ABE for RAMs starting from a controlled recoding scheme. We only provide a high level template below and this suffices to understand the main ideas in our construction. We also later identify some technical challenges that arise when we try to implement this template and how to handle them.

- **Setup**: Let N be the length of the attribute. There are three main types of CHR public keys to be generated: (i) public keys $(\mathsf{Step}[0].\mathsf{pk}_1^{\mathsf{db}}, \ldots, \mathsf{Step}[0].\mathsf{pk}_N^{\mathsf{db}})$ corresponding to the attribute database, (ii) anchor public key-secret key pair $(\mathsf{pk}_0, \mathsf{sk}_0)$, (iii) target public key $\mathsf{pk}_{\mathsf{out}}$. In addition, CHR public key used to encode the initial read address, namely $\mathsf{Step}[0].\mathsf{pk}^{\mathsf{ra}}$ and the CHR public key used to encode the initial state information $\mathsf{Step}[0].\mathsf{pk}^{\mathsf{st}}$. All these public keys, denoted by $\mathsf{Step}[0].PK$, will be part of the ABE public key.

  The anchor secret key sk_0 will be set as the master secret key of the ABE scheme.

- **Key Generation**: Let P be the RAM program for which we need to generate the ABE key with run time upper bounded by T and let C be the step circuit associated with P.

  Sample public keys for every step in $[T-1]$ and the number of such public keys for every step is proportional to the output length of C. That is, generate $\mathsf{Step}[1].PK, \ldots, \mathsf{Step}[T-1].PK$, where $\mathsf{Step}[i].PK$ denotes the set of public keys associated with the i-th step.

The next step is to generate recoding keys $\mathsf{Step}[1].RK, \ldots, \mathsf{Step}[T-1].RK$, where the recoding keys in the set $\mathsf{Step}[i]$ recodes the encodings w.r.t the $\mathsf{Step}[i-1]$ public keys to encodings w.r.t the $\mathsf{Step}[i]$ public keys.

Execute the following two steps for every time step $t \in [T]$:

– Homomorphism keys: Execute the key evaluation algorithm $\mathsf{KeyEval}$ of CHR on the set of public keys on $\mathsf{Step}[t-1].PK$ to obtain the set of public keys $\mathsf{Step}[t-1].PK^{hom}$. The public keys in $\mathsf{Step}[t-1].PK^{hom}$ is used to encode the output of C in the $(t-1)$-th step.

– Controlled Recoding keys: Execute the key recoding algorithm $\mathsf{ReEncKG}$ of CHR on the public keys $\mathsf{Step}[t-1].PK^{hom}$ and control functions in the class $\mathcal{F}$ to obtain the set of recoding keys $\mathsf{Step}[t].RK$, for every time step $t \in [T]$. The class $\mathcal{F}$ is used to translate the output of the $(t-1)$-th step circuit to the input of t-th step circuit.

  To give a glimpse of what $\mathcal{F}$ contains, we give two examples:

  * Ind: this is an identity function. This is useful in converting an encoding of state output by the previous step into an encoding input to the next step. This is also useful in transferring the read address output by previous step to the next one.

  * $f_i(i', b)$: this outputs b only if $i = i'$. This is useful for writing operation: suppose the step circuit at some time t outputs a location i' and value b to be written to. In this case, a recoding key associated with f_i will transform encoding of location i' into encoding of b in the i-th database location only if $i = i'$.

Set the ABE key of the program P to be $(\mathsf{Step}[1].RK, \ldots, \mathsf{Step}[T].RK)$.

- **Encryption**: It takes as input attribute x of size N and message μ. As before, it first samples secret randomness $\mathbf{s}$ from a distribution. It computes the following encodings: (i) encoding of x_i and $\mathbf{s}$ under the public key pk_i, (ii) encoding of 0 and $\mathbf{s}$ under the public key pk_0 and finally, (iii) encoding ct^* of μ and $\mathbf{s}$ under the public key $\mathsf{pk}_{\mathsf{out}}$. Additionally, it also computes encoding of initial read address (set to 1) under $\mathsf{Step}[0].\mathsf{pk}^{\mathsf{ra}}$ and the encoding of initial state (also set to 1s). All the encodings computed will be part of the ABE ciphertext.

- **Decryption**: This proceeds in T steps, where T is the runtime of the RAM program and in each step, it executes homomorphism and controlled recoding phases. In more detail, in the t^{th} step, it executes:

  – Homomorphism: The step circuit C is homomorphically evaluated on the encodings output by $(t-1)$-th step to obtain encodings of output of t-th step under the public keys in $\mathsf{Step}[t].PK^{hom}$.

  – Controlled Recoding: Using the recoding keys in $\mathsf{Step}[t].RK$, the encodings computed under the public keys in $\mathsf{Step}[t].PK^{hom}$ can be recoded to encodings computed under the public keys in $\mathsf{Step}[t].PK$. As described earlier, the recoding keys determine what value to be fed to the t-th time step as a function of the $(t-1)$-th time step.

  In the last step, once we have an encoding of 0 and $\mathbf{s}$ under $\mathsf{pk}_{\mathsf{out}}$, (as before) we then run the equality test on this encoding and ct^* (as computed in encryption). If they are equal, this means that the secret message μ has to be 0, otherwise it has to be 1.

Security Overview. We describe the main steps in the security proof.

- As in the case of ABE for circuits scheme, the first step is to simulate the public keys produced by $\mathsf{Sim.CHRSetup}$. In particular, the challenge attribute x^* is programmed in $\mathsf{Sim.CHRSetup}$.

- The next goal is to simulate the intermediate recoding keys (i.e, $\mathsf{Step}[1].RK, \ldots, \mathsf{Step}[T-1].RK$) in every attribute key. In particular, these recoding keys need to be generated without the help of the anchor secret key. Recall that in the case of ABE for circuits scheme, we could simulate the recoding key rk_f since the output of f on the challenge attribute was guaranteed to be 1. However, in the setting of ABE for RAMs, we have no such guarantee for the intermediate steps of the computation. In particular, there could two programs P_0 and P_1 that output 1 on x^* but differ on every intermediate step of the computation. Thus, we can no longer invoke the indistinguishability of the recoding keys property.

To handle this case, we introduce the final security property associated with the CHR scheme.

<u>IV. Indistinguishability of Simulated Keys</u>: We first define an associated simulator $\mathsf{Sim.CHR_{key}}$. In its basic form, it takes as input anchor public key pk, simulated public keys ($\mathsf{Sim.CHRpk}_1, \ldots, \mathsf{Sim.CHRpk}_n$), associated trapdoors $(\tau_1, \ldots, \tau_n)$, homomorphism circuit C, control function f and it produces simulated recoding keys associated with (C, f) along with simulated target public keys.

- Lets see how to use the above security property to simulate the intermediate recoding keys in the attribute keys. For simplicity, consider the case when the adversary only makes a single attribute key query for RAM program P. Using a standard hybrid argument, we can apply the argument for the case of multiple key queries as well. As a first step, we switch the recoding keys $\mathsf{Step}[1].RK$ in the attribute key of P to simulated recoding keys using the above security property. Note that even the intermediate public keys $\mathsf{Step}[1].PK$ are simulated[6]. In particular, we use the fact that the public keys in $\mathsf{Step}[0].PK$ are simulated using $\mathsf{Sim.CHRSetup}$. Next, we simulate the recoding keys in $\mathsf{Step}[2].RK$. Recall that $\mathsf{Step}[2].RK$ was computed as a function of $\mathsf{Step}[1].PK$ and the step circuit associated with P. Hence, in order to simulate $\mathsf{Step}[2].RK$ we first need to simulate $\mathsf{Step}[1].PK$. But note that we already simulated $\mathsf{Step}[1].PK$ by $\mathsf{Sim.CHR_{key}}$ in the previous step itself! This allows for carry out successful simulation of $\mathsf{Step}[2].RK$.

- Continuing this way, we can simulate all the recoding keys in $\mathsf{Step}[1].RK, \ldots, \mathsf{Step}[T-1].RK$. We cannot, however, use the indistinguishability of simulated keys property to simulate $\mathsf{Step}[T].RK$. This is because, the simulator $\mathsf{Sim.CHR_{key}}$ would end up simulating the target key, which in our construction is $\mathsf{pk^{out}}$. In turn this means that we cannot apply the hybrid argument (for the multiple key queries case) as $\mathsf{pk^{out}}$ is reused across different attribute keys. However, we can still use the indistinguishability of recoding keys property to argue this, since the simulator $\mathsf{Sim.CHR_{rk}}$ does not simulate the target public key. To invoke this, we require that the output of P on x^* is not 0 and this in turn is guaranteed by the ABE security experiment.

- Once the anchor secret key is not used in the generation of the recoding keys for any of the attribute keys, we can now invoke the pseudorandomness of ciphertexts property to argue that the secret message in the ABE encryption is hidden. This completes the security proof.

A remark about the definition of indistinguishability of simulated keys: there are two ways to generate the simulated public keys ($\mathsf{Sim.CHRpk}_1, \ldots, \mathsf{Sim.CHRpk}_n$). We can use $\mathsf{Sim.CHRSetup}$ to

[6]We emphasize that the intermediate public keys are generated afresh for every attribute key. This enables us to apply the hybrid argument for the case of multiple key queries.

generate these keys. Indeed, to argue the security of the recoding keys in $\mathsf{Step}[1].RK$, the public keys in $\mathsf{Step}[0].PK$ is simulated using $\mathsf{Sim.CHRSetup}$. Another option is to invoke $\mathsf{Sim.CHR}_{key}$ to generate the $\{\mathsf{Sim.CHRpk}_i\}_i$. This is not circular since the simulated public keys produced by $\mathsf{Sim.CHR}_{key}$ in the first step is used in the second step by $\mathsf{Sim.CHR}_{key}$ to produce the recoding keys in $\mathsf{Step}[2].RK$. In the technical sections, we formalize this by associating a distribution $\mathcal{E}_{aux}$ which produces $\{\mathsf{Sim.CHRpk}_i\}$.

Instantiation of CHR. It remains to show that the controlled homomorphic recoding schemes can be based on learning with errors. The template for encoding and the key generation is inspired by the schemes of Gorbunov et al. [GVW15a] and Boneh et al. [BGG+14].

To encode a message b with secret randomness $\mathbf{s}$ under the public key pk, our encoding is of the form $\mathbf{s}^{\mathsf{T}}(\mathbf{A}+b\mathbf{G})+e^{\mathsf{T}}$, where $\mathbf{s}^{\mathsf{T}},\mathbf{A}$ and e are sampled according to the parameters associated with the learning with errors assumption. Suppose we have many encodings $\mathbf{s}^{\mathsf{T}}(\mathbf{A}_1+b_1\mathbf{G})+e_1^{\mathsf{T}}, \ldots, \mathbf{s}^{\mathsf{T}}(\mathbf{A}_n+b_n\mathbf{G})+e_n^{\mathsf{T}}$ then we can compute an encoding of the form $\mathbf{s}^{\mathsf{T}}(\mathbf{A}_C+C(b_1,\ldots,b_n)\mathbf{G})+e'^{\mathsf{T}}$, where $\mathbf{A}_C$ is homomorphically computed on public keys $\mathbf{A}_1,\ldots,\mathbf{A}_n$.

To handle the recoding process, we need to generate recoding keys individually for every control function. The recoding keys are set to be lattice trapdoors. As an illustration, we show how to generate lattice trapdoor for the case of control function Ind.

- Ind: Suppose the input to the recoding key generation is anchor public key pk_0, secret key sk_0, public key pk_1, target public key $\mathsf{pk}_{\mathsf{tgt}}$ and control function Ind. We set $\mathsf{pk}_0=\mathbf{A}_0$, $\mathsf{sk}_0=\mathsf{T}_{\mathbf{A}_0}$ (a trapdoor for $\mathbf{A}_0$), public key $\mathsf{pk}_1=\mathbf{A}_1$, $\mathsf{pk}_{\mathsf{tgt}}=\mathbf{A}_{\mathsf{tgt}}$. The recoding key is of the form $[\mathbf{R}_0|\mathbf{I}]^{\mathsf{T}}$ such that $[\mathbf{A}_0|\mathbf{A}_1]\cdot[\mathbf{R}_0|\mathbf{I}]^{\mathsf{T}}=\mathbf{A}_{\mathsf{tgt}}$. Using this recoding key, we can translate encoding of any message b under $\mathbf{A}_1$ into an encoding of b under $\mathbf{A}_{\mathsf{tgt}}$.

We use similar ideas to generate recoding keys for the control functions that are relevant to th construction of ABE for RAMs. We summarise this class of functions in Equation 3 (Section 3.2).

Additional Challenges. The template described above captures the main ideas in our construction. However, while implementing this high level template, we encounter additional difficulties and we highlight a couple of them below.

Repeated Writing Issue. Yet another issue is that of malicious execution of the computation. Suppose the 100^{th} location was updated in the 11^{th} step and also in the 25^{th} step. Lets consider what happens when the RAM program in the 30^{th} step is supposed to read the 100^{th} location. A malicious evaluator could use the encryption computed in the 11^{th} step to be input to the 30^{th} step, instead of 25^{th} step. We need to implement suitable checks in place that prevents him from performing these types of attacks.

In the technical sections, we introduce circuits C^{up} (Figure 4) and C^{ck} (Figure 5) that keeps track of all the addresses written so far along with the along with the most recent time stamps associated with them. We also introduce the control function f_{ij} (Figure 1) is used to ensure that only the correct encoding is recoded.

Early Termination. What if the program terminates much earlier than the upper time bound T? The template described so far, as is, would have the decryption algorithm run in T steps even if the program terminated early. A naive approach to solve this problem would be to give out multiple keys for programs upper bounded by runtime $2, 2^2, \ldots, T$. This would introduce an additional overhead of $\log(T)$ in the size of the original key. Instead we show that we can tweak the original scheme such that the decryption time can be made to be input-dependent.

1.3 Related Work

The constructions of ABE systems has a rich literature. The seminal result of Goyal, Pandey, Sahai and Waters [GPSW06] presented the first construction of ABE for boolean formulas from bilinear DDH assumption. Since then, several prominent works achieved stronger security guarantees [LOS+10], better efficiency or design guarantees [Wee14, Att14, AC16] and achieving stronger models of ABE for a restricted class of functions [KSW08]. The breakthrough work of Gorbunov, Vaikuntanathan and Wee [GVW15a] presented the first construction of ABE for all polynomial-sized circuits assuming learning with errors. Following this, several works [BGG+14, BV16] improved this result in terms of efficiency and also considering stronger security models [GVW15a]. There are a few works that consider ABE in other models of computation. Waters [Wat12] proposed a construction of functional encryption for regular languages. As mentioned earlier, Goldwasser et al. [GKP+13b] considered the problem of constructing attribute based encryption for RAMs. Ananth and Sahai [AS16] construct functional encryption for Turing machines assuming sub-exponentially secure functional encryption for circuits. Deshpande et al. [DKW16] present an alternate construction of attribute based encryption for Turing machines under the same assumptions.

2 Preliminaries

Notation. Let λ denote the security parameter, and PPT denote probabilistic polynomial time. Bold uppercase letters are used to denote matrices $\mathbf{M}$, and bold lowercase letters for vectors $\boldsymbol{v}$. We use $[n]$ to denote the set $\{1, ..., n\}$. We say a function $\mathsf{negl}(\cdot) : \mathbb{N} \to (0, 1)$ is negligible, if for every constant $c \in \mathbb{N}$, $\mathsf{negl}(n) < n^{-c}$ for sufficiently large n. Let X and Y be two random variables taking values in Ω. Define the statistical distance, denoted as $\Delta(X, Y)$ as

$$\Delta(X, Y) := \frac{1}{2} \sum_{s \in \Omega} |\mathsf{Pr}[X = s] - \mathsf{Pr}[Y = s]|$$

Let $X(\lambda)$ and $Y(\lambda)$ be distributions of random variables. We say that X and Y are statistically close, denoted as $X \overset{s}{\approx} Y$, if $d(\lambda) := \Delta(X(\lambda), Y(\lambda))$ is a negligible function of λ. We say two distributions $X(\lambda)$ and $Y(\lambda)$ are computationally indistinguishable, denoted as $X \overset{c}{\approx} Y$ if for any PPT distinguisher D, it holds that $|\mathsf{Pr}[D(X(\lambda)) = 1] - \mathsf{Pr}[D(Y(\lambda)) = 1]| = \mathsf{negl}(\lambda)$.

2.1 Random Access Machines

We recall the definition of RAM program in [GHL+14]. A RAM computation consists of a RAM program P and a database D. The representation size of P is independent of the length of the database D. P has random access to the database D and we represent this as P^D. On input x, $P^D(x)$ outputs the answer y. In more detail, the computation proceeds as follows.

The RAM program P is represented as a step-circuit C. It takes as input internal state from the previous step, location to be read, value at that location and it outputs the new state, location to be written into, value to be written and the next location to be read. More formally, for every $i \in T$, where T is the upper running time bound

$$(\mathsf{st}_i, \mathsf{loc}_i^{\mathsf{w}}, b_i^{\mathsf{w}}, \mathsf{loc}_i^{\mathsf{r}}) \leftarrow C(\mathsf{st}_{i-1}, \mathsf{loc}_{i-1}^{\mathsf{r}}, b_{i-1}^{\mathsf{r}}),$$

where we have the following:

- st_{i-1} denotes the state from the $(i-1)$-th step and st_i denotes the state in the i-th step. Initial state st_0 is set to be x, which is the input to $P^D(\cdot)$.
- $\mathsf{loc}^{\mathsf{r}}_{i-1}$ denotes the location to be read from, as output by the $(i-1)$-th step.
- b^{r}_{i-1} denotes the bit at the location $\mathsf{loc}^{\mathsf{r}}_{i-1}$.
- $\mathsf{loc}^{\mathsf{r}}_{i}$ denotes the location to be read from, in the next step.
- $\mathsf{loc}^{\mathsf{w}}_{i}$ denotes the location to be written into.
- b^{w}_{i} denotes the value to be written at the location $\mathsf{loc}^{\mathsf{w}}_{i}$.

At the end of the computation, denote the final state to be $\mathsf{st}_{\mathsf{end}}$. If the computation has been performed correctly, $\mathsf{st}_{\mathsf{end}} = y$. In this work, we consider a simpler case, where the RAM program P does not take additional input x and the output of P^D is in space $\{0, 1\}$.

2.2 Attribute-Based Encryption for RAMs

In this part, we recall the syntax and security definition of (key-policy) attribute-based encryption (ABE). An ABE scheme for a RAM program P and a database D consists a tuple of PPT algorithms $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with details as follows:

- **Setup,** $\mathsf{Setup}(1^\lambda, 1^T)$: On input security parameter λ and upper time bound T, setup algorithm outputs public parameters pp and master secret key msk.
- **Key Generation,** $\mathsf{KeyGen}(\mathsf{msk}, P)$: On input a master secret key msk and a RAM program P, it outputs a secret key sk_P.
- **Encryption,** $\mathsf{Enc}(\mathsf{pp}, D, \mu)$: On input public parameters pp, a database D and a message μ, it outputs a ciphertext ct_D.
- **Decryption,** $\mathsf{Dec}(\mathsf{sk}_P, \mathsf{ct}_D)$: This is modeled as a RAM program. In particular, this algorithm will have random access to the binary representations of the key sk_P and the ciphertext ct_D. It outputs the corresponding plaintext μ if the decryption is successful; otherwise, it outputs $\perp$.

Definition 2.1 (ABE Correctness)**.** *We say the ABE described above is correct, if for any* $(\mathsf{msk}, \mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda, 1^T)$*, any message* μ*, any RAM program* P*, and any database* D *where* P^D *outputs* 0*, we have* $\mathsf{Dec}(\mathsf{sk}_P, \mathsf{ct}_D) = \mu$*, where* $\mathsf{sk}_P \leftarrow \mathsf{KeyGen}(\mathsf{msk}, P)$ *and* $\mathsf{ct}_D \leftarrow \mathsf{Enc}(\mathsf{pp}, D, \mu)$*.*

Sub-linear Efficiency. We require that the decryption complexity should be polynomial in the runtime of the programs. In particular, if a program takes time sublinear in the input length, even the decryption algorithm should take time sublinear in the input length.

Definition 2.2 (Sublinear Decryption)**.** *An ABE for RAMs scheme* ABE *is said to satisfy sublinear decryption property if the following holds: (i)* $(\mathsf{msk}, \mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda)$*, (ii)* $\mathsf{sk}_P \leftarrow \mathsf{KeyGen}(\mathsf{msk}, P)$ *for some RAM program* P*, (iii)* $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, D, x)$ *and, (iv) the decryption* Dec *of the functional key* sk_P *on input the ciphertext* ct *takes time poly*(t, λ)*, where* t *is the running time of* P^D*.*

In particular, if P *takes time sublinear in* $|D|$ *then the decryption time of* sk_P *on the ciphertext* ct *is also sublinear.*

Security Definition. We present the simulation-based definition of selective security of attribute-based encryption as follows

Definition 2.3. *An ABE scheme* Π *for RAMs is simulation-based selectively secure if there exist* PPT *simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ *such that for any* PPT *admissible adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, the two distributions* $\{\mathbf{Expt}_{\mathcal{A}}^{\mathsf{real}}(1^\lambda)\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \{\mathbf{Expt}_{\mathcal{S}}^{\mathsf{ideal}}(1^\lambda)\}_{\lambda\in\mathbb{N}}$ *are computationally indistinguishable*

---

1. $D^* \leftarrow \mathcal{A}_1(1^\lambda)$
2. $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^T, D^*)$
3. $\mu \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pp})$
4. $\mathsf{ct}_{D^*} \leftarrow \mathsf{Enc}(\mathsf{pp}, D^*, \mu)$
5. $\alpha \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pp}, \mathsf{ct}_{D^*})$
6. Output $(\mathsf{pp}, \mu, \alpha)$

(a) $\mathbf{Expt}_{\mathcal{A}}^{\mathsf{real}}(1^\lambda)$

1. $D^* \leftarrow \mathcal{A}_1(1^\lambda)$
2. $\mathsf{pp} \leftarrow \mathcal{S}_1(1^\lambda, 1^T, D^*)$
3. $\mu \leftarrow \mathcal{A}_2^{\mathcal{S}_3(D^*,\cdot)}(\mathsf{pp})$
4. $\mathsf{ct}_{D^*} \leftarrow \mathcal{S}_2(\mathsf{pp}, D^*, 1^{|\mu|})$
5. $\alpha \leftarrow \mathcal{A}_2^{\mathcal{S}_3(D^*,\cdot)}(\mathsf{pp}, \mathsf{ct}_{D^*})$
6. Output $(\mathsf{pp}, \mu, \alpha)$

(b) $\mathbf{Expt}_{\mathcal{S}}^{\mathsf{ideal}}(1^\lambda)$

We call adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ admissible*, if the queries* P_i *made by* $(\mathcal{A}_2, \mathcal{A}_3)$ *satisfies* $P_i(D^*)) \neq 0$*.*

Remark 2.1. We note that we can generalize the ABE syntax, by allowing RAM program P to take in auxiliary input x, denoted as $P^D(x)$. The encryption algorithm $\mathsf{Enc}(\mathsf{pp}, D, x, \mu)$ outputs ciphertext $\mathsf{ct}_{D,x}$ associated with database D and auxiliary input x. Correctness and security can be defined similarly by replacing database D with (D, x).

2.3 Lattice Background

A full-rank m-dimensional integer lattice $\Lambda \subset \mathbb{Z}^m$ is a discrete additive subgroup whose linear span is $\mathbb{R}^m$. The basis of Λ is a linearly independent set of vectors whose linear combinations are exactly Λ. Every integer lattice is generated as the $\mathbb{Z}$-linear combination of linearly independent vectors $\mathbf{B} = \{\boldsymbol{b}_1, ..., \boldsymbol{b}_m\} \subset \mathbb{Z}^m$. For a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, we define the "$q$-ary" integer lattices:

$$\Lambda_q^\perp = \{\boldsymbol{e} \in \mathbb{Z}^m | \mathbf{A}\boldsymbol{e} = \mathbf{0} \bmod q\}, \qquad \Lambda_q^{\mathbf{u}} = \{\boldsymbol{e} \in \mathbb{Z}^m | \mathbf{A}\boldsymbol{e} = \boldsymbol{u} \bmod q\}$$

It is obvious that $\Lambda_q^{\boldsymbol{u}}$ is a coset of $\Lambda_q^\perp$.

Let Λ be a discrete subset of $\mathbb{Z}^m$. For any vector $\boldsymbol{c} \in \mathbb{R}^m$, and any positive parameter $\sigma \in \mathbb{R}$, let $\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x}) = \exp(-\pi||\boldsymbol{x} - \boldsymbol{c}||^2/\sigma^2)$ be the Gaussian function on $\mathbb{R}^m$ with center $\boldsymbol{c}$ and parameter σ. Next, we let $\rho_{\sigma,\boldsymbol{c}}(\Lambda) = \sum_{\boldsymbol{x}\in\Lambda} \rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x})$ be the discrete integral of $\rho_{\sigma,\boldsymbol{x}}$ over Λ, and let $\mathcal{D}_{\Lambda,\sigma,\boldsymbol{c}}(\boldsymbol{y}) := \frac{\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{y})}{\rho_{\sigma,\boldsymbol{c}}(\Lambda)}$. We abbreviate this as $\mathcal{D}_{\Lambda,\sigma}$ when $\boldsymbol{c} = \mathbf{0}$. We note that $\mathcal{D}_{Z^m,\sigma}$ is $\sqrt{m}\sigma$-bounded.

Let S^m denote the set of vectors in $\mathbb{R}^m$ whose length is 1. The norm of a matrix $\mathbf{R} \in \mathbb{R}^{m\times m}$ is defined to be $\mathsf{sup}_{\boldsymbol{x}\in S^m}||\mathbf{R}\boldsymbol{x}||$. The LWE problem was introduced by Regev [Reg05], who showed that solving it *on average* is as hard as (quantumly) solving several standard lattice problems *in the worst case*.

Definition 2.4 (LWE)**.** *For an integer* $q = q(n) \geq 2$*, and an error distribution* $\chi = \chi(n)$ *over* $\mathbb{Z}_q$*, the* Learning With Errors problem $\mathsf{LWE}_{n,m,q,\chi}$ *is to distinguish between the following pairs of distributions (e.g. as given by a sampling oracle* $\mathcal{O} \in \{\mathcal{O}_{\boldsymbol{s}}, \mathcal{O}_\$\}$*):*

$$\{\mathbf{A}, \boldsymbol{s}^\mathsf{T}\mathbf{A} + \boldsymbol{x}^\mathsf{T}\} \text{ and } \{\mathbf{A}, \boldsymbol{u}\}$$

where $\mathbf{A} \overset{\$}{\leftarrow} \mathbb{Z}_q^{n\times m}$*,* $\boldsymbol{s} \overset{\$}{\leftarrow} \mathbb{Z}_q^n$*,* $\boldsymbol{u} \overset{\$}{\leftarrow} \mathbb{Z}_q^m$*, and* $\boldsymbol{x} \leftarrow \chi^m$*.*

Gadget matrix. The gadget matrix described below is proposed in [MP12, AP14].

Definition 2.5. *Let* $m = n \cdot \lceil \log q \rceil$*, and define the gadget matrix* $\mathbf{G} = \boldsymbol{g}_2 \otimes \mathbf{I}_n \in \mathbb{Z}_q^{n \times m}$*, where the vector* $\boldsymbol{g}_2 = (1, 2, 4, ..., 2^{\lfloor \log q \rfloor}) \in \mathbb{Z}_q^{\lceil \log q \rceil}$*. We will also refer to this gadget matrix as "powers-of-two" matrix. We define the inverse function* $\mathbf{G}^{-1} : \mathbb{Z}_q^{n \times m} \to \{0,1\}^{m \times m}$ *which expands each entry* $a \in \mathbb{Z}_q$ *of the input matrix into a column of size* $\lceil \log q \rceil$ *consisting of the bits of binary representations. We have the property that for any matrix* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$*, it holds that* $\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A}$*.*

Sampling Algorithms. We will use the following algorithms to sample short vectors from specified lattices.

Lemma 2.2 ([GPV08, AP10])**.** *Let* q, n, m *be positive integers with* $q \geq 2$ *and sufficiently large* $m = \Omega(n \log q)$*. There exists a* PPT *algorithm* $\mathsf{TrapGen}(q, n, m)$ *that with overwhelming probability outputs a pair* $(\mathbf{A} \in \mathbb{Z}_q^{n \times m}, \mathbf{T_A} \in \mathbb{Z}^{m \times m})$ *such that the distribution of* $\mathbf{A}$ *is statistically close to uniform distribution over* $\mathbb{Z}_q^{n \times m}$ *and* $\mathbf{T_A}$ *is a basis for* $\Lambda_q^{\perp}(\mathbf{A})$ *satisfying*

$$||\mathbf{T_A}|| \leq O(n \log q) \quad \text{and} \quad ||\widetilde{\mathbf{T_A}}|| \leq O(\sqrt{n \log q})$$

except with $\mathsf{negl}(n)$ *probability.*

Lemma 2.3 ([GPV08, CHKP10, ABB10])**.** *Let* $q > 2, m > n$*. There are three sampling algorithms as follows:*

- *There is a* PPT *algorithm* $\mathsf{SamplePre}(\mathbf{A}, \mathbf{T_A}, \boldsymbol{u}, s)$*, that takes as input: (1) a rank-n matrix* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$*, (2) a "short" basis* $\mathbf{T_A}$ *for lattice* $\Lambda_q^{\perp}(\mathbf{A})$*, a vector* $\boldsymbol{u} \in \mathbb{Z}_q^n$*, (3) a Gaussian parameter* $s > ||\widetilde{\mathbf{T_A}}|| \cdot \omega(\sqrt{\log(m)})$*; then outputs a vector* $\boldsymbol{r} \in \mathbb{Z}^{m+m_1}$ *distributed statistically close to* $\mathcal{D}_{\Lambda_q^{\boldsymbol{u}}(\mathbf{A}),s}$*.*
- *There is a* PPT *algorithm* $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{T_A}, \boldsymbol{u}, s)$*, that takes as input: (1) a rank-n matrix* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$*, and any matrix* $\mathbf{B} \in \mathbb{Z}_q^{n \times m_1}$*, (2) a "short" basis* $\mathbf{T_A}$ *for lattice* $\Lambda_q^{\perp}(\mathbf{A})$*, a vector* $\boldsymbol{u} \in \mathbb{Z}_q^n$*, (3) a Gaussian parameter* $s > ||\widetilde{\mathbf{T_A}}|| \cdot \omega(\sqrt{\log(m+m_1)})$*; then outputs a vector* $\boldsymbol{r} \in \mathbb{Z}^{m+m_1}$ *distributed statistically close to* $\mathcal{D}_{\Lambda_q^{\boldsymbol{u}}(\mathbf{F}),s}$ *where* $\mathbf{F} := (\mathbf{A}|\mathbf{B})$*.*
- *There is a* PPT *algorithm* $\mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{T_B}, \boldsymbol{u}, s)$*, that takes as input: (1) a matrix* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$*, and a rank-n matrix* $\mathbf{B} \in \mathbb{Z}_q^{n \times m}$*, a matrix* $\mathbf{R} \in \mathbb{Z}_q^{m \times m}$*, where* $s_{\mathbf{R}} := ||\mathbf{R}|| = \sup_{\boldsymbol{x}:||\boldsymbol{x}||=1} ||\mathbf{R}\boldsymbol{x}||$*, (2) a "short" basis* $\mathbf{T_B}$ *for lattice* $\Lambda_q^{\perp}(\mathbf{B})$*, a vector* $\boldsymbol{u} \in \mathbb{Z}_q^n$*, (3) a Gaussian parameter* $s > ||\widetilde{\mathbf{T_B}}|| \cdot s_{\mathbf{R}} \cdot \omega(\sqrt{\log m})$*; then outputs a vector* $\boldsymbol{r} \in \mathbb{Z}^{2m}$ *distributed statistically close to* $\mathcal{D}_{\Lambda_q^{\boldsymbol{u}}(\mathbf{F}),s}$ *where* $\mathbf{F} := (\mathbf{A}|\mathbf{AR} + \mathbf{B})$*.*

Based on the above sampling algorithms, we have the following lemma:

Lemma 2.4 ([GVW15c])**.** *Given integers* $n \geq 1, q \geq 2$ *there exists some* $m^* = m^*(n,q) = O(n \log q)$*,* $\beta = \beta(n,q) = O(n\sqrt{\log q})$ *and* $s > ||\widetilde{\mathbf{T_A}}|| \cdot \omega(\sqrt{\log(m)})$ *such that for all* $m \geq m^*$ *and all k, we have*

$$\mathbf{A} \overset{s}{\approx} \mathbf{A}', \qquad (\mathbf{A}, \mathbf{T_A}, \mathbf{U}, \mathbf{V}) \overset{c}{\approx} (\mathbf{A}, \mathbf{T_A}, \mathbf{U}', \mathbf{V}')$$

where $(\mathbf{A}, \mathbf{T_A}) \leftarrow \mathsf{TrapGen}(q, n, m), \mathbf{A}' \overset{\$}{\leftarrow} \mathbb{Z}_q^{n \times m}$ *and* $\mathbf{U} \leftarrow \mathcal{D}_{\mathbb{Z}^{m \times k}}, \mathbf{V} = \mathbf{A} \cdot \mathbf{U}$*,* $\mathbf{V}' \overset{\$}{\leftarrow} \mathbb{Z}_q^{n \times k}$ *and* $\mathbf{U}' \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{T_A}, \mathbf{V}', s)$*.*

2.4 Homomorphic Evaluation Algorithms

In this part, we recall three homomorphic evaluation algorithms ($\mathsf{PubEval}, \mathsf{TrapEval}, \mathsf{CtEval}$). The following definition about homomorphic evaluation respective to some circuits is implicitly used in various constructions, such as attribute-based encryption [BGG$^+$14, GV15] and predicate encryption [GVW15b].

Definition 2.6 (δ-expanding evaluation)**.** *The deterministic algorithms* ($\mathsf{PubEval}$, $\mathsf{TrapEval}, \mathsf{CtEval}$) *are δ-expanding with function (circuit with u inputs) $f : \mathcal{X}^d \to \mathcal{Y}$ if they are efficient and satisfy the following properties:*

- $\mathsf{PubEval}(\{\mathbf{D}_i \in \mathbb{Z}_q^{n\times m}\}_{i\in[d]}, f)$*: On input matrices $\{\mathbf{D}_i\}_{i\in[d]}$ and a function $f \in \mathcal{F}$, the public evaluation algorithm outputs $\mathbf{D}_f \in \mathbb{Z}_q^{n\times m}$ as the result.*

- $\mathsf{TrapEval}(\boldsymbol{x} \in \mathcal{X}^d, \mathbf{A} \in \mathbb{Z}_q^{n\times m}, \{\mathbf{R}_i\}_{i\in[d]}, f)$*: the trapdoor evaluation algorithm outputs $\mathbf{R}_f$, such that*
$$\mathsf{PubEval}(\{\mathbf{A}\mathbf{R}_i + x_i\mathbf{G}\}_{i\in[d]}, f) = \mathbf{A}\mathbf{R}_f + f(\boldsymbol{x})\mathbf{G}$$
Furthermore, we have $||\mathbf{R}_f|| \leq \delta \cdot \max_{i\in[d]} ||\mathbf{R}_i||$.

- $\mathsf{CtEval}(\{\boldsymbol{c}_i\}_{i=1}^d, \boldsymbol{x}, f)$*: On input vectors $\{\boldsymbol{c}_i\}_{i=1}^d \in \mathbb{Z}_q^m$, an attribute $\boldsymbol{x}$ and function f, the ciphertext evaluation algorithm outputs $\boldsymbol{c}_{f(\boldsymbol{x})} \in \mathbb{Z}_q^m$, such that*
$$\mathsf{CtEval}(\{\boldsymbol{s}^\mathsf{T}(\mathbf{D}_i + x_i\mathbf{G}) + \boldsymbol{e}_i\}_{i\in[d]}, \boldsymbol{x}, f) = \boldsymbol{s}^\mathsf{T}(\mathbf{D}_f + f(\boldsymbol{x})\mathbf{G}) + \boldsymbol{e}'$$
where $\boldsymbol{x} = (x_1, ..., x_d)$ and $\mathbf{D}_f = \mathsf{PubEval}(\{\mathbf{D}_i \in \mathbb{Z}_q^{n\times m}\}_{i\in[d]}, f)$. Furthermore, we require $||\boldsymbol{e}'|| \leq \delta \cdot \max_{i\in[d]} ||\boldsymbol{e}_i||$.

The definition can be extended to δ-expanding with a family of functions $\mathcal{F}$. I.e., ($\mathsf{PubEval}, \mathsf{TrapEval}$) are δ-expanding with $\mathcal{F}$ if and only if for all $f \in \mathcal{F}$, the algorithms are δ-expanding with f.

3 Controlled Homomorphic Recoding Scheme

We propose a controlled homomorphic recoding scheme scheme consisting of probabilistic polynomial-time computable algorithms $\mathsf{CHR} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyEval}, \mathsf{CtEval}, \mathsf{ReEncKG}, \mathsf{ReEnc}, \mathsf{EqTest})$. Denote by $\mathcal{S}$ to be the space of secret messages encrypted in the scheme. We first describe the basic algorithms.

- **Setup,** $\mathsf{CHR.Setup}(1^\lambda)$: On input security parameter λ, it outputs a public key pk and secret key sk.

- **Encoding procedure,** $\mathsf{CHR.Enc}(\mathsf{pk}, y, \boldsymbol{s})$: On input public key pk, public attribute y and secret message $\boldsymbol{s}$ from space $\mathcal{S}$, it outputs the ciphertext ct (containing attribute y).

Homomorphic Evaluation algorithms: We describe the homomorphic evaluation algorithms below. The evaluation algorithm allows for homomorphically computing on the public keys and the attribute messages.

- **Homomorphic key evaluation,** $\mathsf{CHR.KeyEval}(\{\mathsf{pk}_i\}_{i=1}^n, C)$: On input public keys $\{\mathsf{pk}_i\}_{i=1}^n$ and circuit C, it homomorphically evaluates C with respect to $\{\mathsf{pk}_i\}_{i\in[n]}$ to obtain the resulting public key pk_C.

- **Ciphertext evaluation,** $\mathsf{CHR.CtEval}(\{\mathsf{ct}_i\}_{i=1}^n, \{y_i\}_{i=1}^n, C)$: On input ciphertexts $\{\mathsf{ct}_i\}_{i\in[n]}$ under the public keys $\{\mathsf{pk}_i\}_{i\in[n]}$, circuit C, it outputs the resulting ciphertext ct^*, which is an encryption of $C(y_1, \ldots, y_n)$ under the public key $\mathsf{pk}_C = \mathsf{CHR.KeyEval}(\{\mathsf{pk}_i\}_{i\in[n]}, C)$.

  Looking ahead, in the correctness definition, we require that all the ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$ are encoded using same secret message $\boldsymbol{s}$.

Controlled Recoding algorithms: We describe the controlled recoding algorithms below. The recoding algorithm allows for translating ciphertexts which encode messages $\{y_i\}_{i\in[n]}$ generated using public keys $\{\mathsf{pk}_i\}_{i\in[n]}$ into a ciphertext of $C(y_1, \ldots, y_n)$ under the target public key pk^*. This translation is performed using a special recoding key rk.

- **Recoding key generation,** $\mathsf{CHR.ReEncKG}\,(\mathsf{pk}_1, \ldots, \mathsf{pk}_n, \mathsf{sk}_i, \mathsf{pk}^*, f)$: On input set of public keys $\{\mathsf{pk}_j\}_{j\in[n]}$, secret key sk_i for the i-th public key, target public key pk^*, control function f, it outputs the recoding key rk.

- **Ciphertext recoding procedure,** $\mathsf{CHR.ReEnc}\left(\mathsf{rk}, \{(\mathsf{pk}_i, \mathsf{ct}_i)\}_{i\in[n]}\right)$[7] : On input recoding key rk, ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$ computed under public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, it outputs the recoded ciphertext ct^*.

Auxiliary algorithm: Equality Test. Finally, we describe an equality test algorithm. This determines if two ciphertexts corresponds to encryptions of the same attribute message and secret message.

- **Equality test,** $\mathsf{CHR.EqTest}(\mathsf{pk}, \mathsf{ct}_1, \mathsf{ct}_2)$: On input public key pk, two ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$, it outputs Equal if both ct_1 and ct_2 encrypt the same attribute using the same secret message and under the same public key pk. Otherwise, it outputs $\mathsf{NotEqual}$.

Remark 3.1. *Looking ahead, in the construction of ABE for RAMs from CHR, we sample an* anchor *public key and secret key pair* $(\mathsf{pk}_0, \mathsf{sk}_0)$ *and this pair is used in the generation of all the recoding keys.*

3.1 Properties of Controlled Homomorphic Recoding Scheme

We describe the properties to be satisfied by a controlled homomorphic recoding scheme. Before that, we describe some auxiliary algorithms that will be useful to describe the correctness and security properties.

Derivation of Recoding Keys, $\mathsf{DerivReKey}\left(\{\mathsf{pk}_i\}_{i\in[\ell]}, \mathsf{sk}_{i^*}, \{C_i\}_{i\in[L]}, \mathsf{pk}^*, f\right)$: It takes as input public keys $\{\mathsf{pk}_i\}_{i\in[\ell]}$, secret key sk_{i^*} for some $i^* \in [\ell]$, circuits $\{C_i\}_{i\in[L]}$, target public key pk^* and controlled function f, it does the following:

1. Evaluate public key, $\mathsf{pk}_{C_i} \leftarrow \mathsf{CHR.KeyEval}(\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell, C_i)$, for $i \in [L]$.
2. Obtain rk by running $\mathsf{CHR.ReEncKG}(\{\mathsf{pk}_i\}_{i\in[k]}, \{\mathsf{pk}_{C_i}\}_{i\in[L]}, \mathsf{sk}_{i^*}, \mathsf{pk}^*, f)$, where $k \leq \ell$. That is, rk recodes ciphertexts encoded under the public keys $\{\mathsf{pk}_i\}_{i\in[k]}$ and $\{\mathsf{pk}_{C_i}\}_{i\in[L]}$.

Derivation of Recoded Ciphertexts, $\mathsf{DerivReEnc}\left(\mathsf{rk}, \{(\mathsf{pk}_i, \mathsf{ct}_i)\}_{i\in[\ell]}, \{C_i\}_{i\in[L]}\right)$: It takes as input recoding key rk, public keys $\{\mathsf{pk}_i\}_{i\in[\ell]}$, original ciphertexts $\{\mathsf{ct}_i\}_{i\in[\ell]}$, circuits $\{C_i\}_{i\in[L]}$, it does the following:

[7] For ease of notation, we omit the public keys in the input to algorithm $\mathsf{CHR.ReEnc}$ when the context is clear.

1. Evaluate public key, $\mathsf{pk}_{C_i} \leftarrow \mathsf{CHR.KeyEval}(\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell, C_i)$, for $i \in [L]$.
2. Evaluate ciphertexts $\mathsf{ct}_{C_i} \leftarrow \mathsf{CHR.CtEval}(\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell, C_i)$ for $i \in [L]$.
3. Compute the recoding algorithm, $\mathsf{CHR.ReEnc}(\mathsf{rk}, \{(\mathsf{pk}_i, \mathsf{ct}_i)\}_{i\in[k]}, \{(\mathsf{pk}_{C_i}, \mathsf{ct}_{C_i})\}_{i\in[L]})$, to obtain the recoded ciphertext ct^*. Then output ct^*.

Put simply, the recoding key rk recodes ciphertexts computed with respect to the public keys $\{\mathsf{pk}_i\}_{i\in[k]}$ and $\{\mathsf{pk}_{C_i}\}_{i\in[L]}$ into a ciphertext encoded with respect to the public key pk^*.

We explain the correctness property below. It incorporates the correctness of both the homomorphic and the recoding phases.

Definition 3.1 (CHR Correctness)**.** *Consider a message $\mathbf{y} \in \{0,1\}^\ell$, secret message $\mathbf{s}$ and ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell$, circuits $C_1, \ldots, C_L$ and a function f. Consider the following process:*

1. *Execute $\mathsf{CHR.Setup}(1^\lambda)$, ℓ number of times to obtain ℓ public/secret key pairs $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[\ell]}$. Also, compute target public key $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{CHR.Setup}(1^\lambda)$.*
2. *Compute $\mathsf{DerivReKey}(\{\mathsf{pk}_i\}_{i\in[\ell]}, \mathsf{sk}_{i^*}, \{C_i\}_{i\in[L]}, \mathsf{pk}^*, f)$ to obtain the recoding key rk, for some $i^* \in [\ell]$.*
3. *Compute $\mathsf{DerivReEnc}(\mathsf{rk}, \{(\mathsf{pk}_i, \mathsf{ct}_i)\}_{i\in[\ell]}, \{C_i\}_{i\in[L]})$ to obtain ciphertext ct^*.*
4. *Finally, compute the ciphertext $\mathsf{ct}^*_{\mathsf{fresh}} \leftarrow \mathsf{CHR.Enc}(\mathsf{pk}^*, f(C_1(\mathbf{y}), \ldots, C_L(\mathbf{y})), \boldsymbol{s})$.*

Suppose it holds that $\mathsf{CHR.EqTest}(\mathsf{ct}_i, \mathsf{Enc}(\mathsf{pk}_i, y_i, \mathbf{s})) = \mathsf{Equal}$ with probability $1 - \mathsf{negl}(\lambda)$, where y_i is the i-th bit of $\mathbf{y}$ and $\mathbf{s}$ is uniformly random picked. Then it should hold that $\mathsf{CHR.EqTest}(\mathsf{ct}^*, \mathsf{ct}^*_{\mathsf{fresh}}) = \mathsf{Equal}$ with probability $1 - \mathsf{negl}(\lambda)$.

3.1.1 Security Definitions

The security definitions of controlled homomorphic recoding scheme Π consists of four parts: indistinguishability of setup, indistinguishability of simulated keys, indistinguishability of recoding keys and pseudorandomness of ciphertexts. We describe them in detail below.

Indistinguishability of Setup. This property intuitively states that the distribution of public keys produced by real setup is statistically close to that produced by simulated setup. We define the following simulated setup algorithm:

$\mathsf{Sim.CHRSetup}(1^\lambda, z)$: It takes as input security parameter λ, input z to be programmed and it outputs the programmed simulated public key $\mathsf{Sim.pk}$ and associated trapdoor τ.

Definition 3.2 (Indistinguishability of Setup)**.** *A controlled homomorphic recoding scheme Π is said to satisfy* **indistinguishability of setup** *if $\{\mathsf{pk}\} \overset{s}{\approx} \{\mathsf{Sim.pk}\}$ holds, where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{CHR.Setup}(1^\lambda)$ and $(\mathsf{Sim.pk}, \tau) \leftarrow \mathsf{Sim.CHRSetup}(1^\lambda, z)$ for some $z \in \mathbb{Z}$.*

Indistinguishability of Simulated Keys. We will first define a simulated key generation algorithm.

$\mathsf{Sim.CHR}_{\mathsf{key}}\left(\mathsf{pk}, \{\mathsf{Sim.pk}_i, \tau_i\}_{i\in[\ell]}, \{C_{ij}\}_{i\in[L], j\in[K]}, \{f_j\}_{j\in[K]}\right)$: On input public keys pk, $\{\mathsf{Sim.pk}_i\}_{i\in[\ell]}$ with associated trapdoors $\{\tau_i\}_{i\in[\ell]}$, circuits $\{C_{ij}\}_{i\in[L], j\in[K]}$, functions $\{f_j\}_{j\in[K]}$, it outputs simulated recoding keys $\{\mathsf{Sim.rk}_i\}_{i\in[K]}$ and simulated target public keys $\{\mathsf{Sim.pk}^*_i\}_{i\in[K]}$.

We now define this property formally. This property states the output distribution of the following two procedures are statistically close:

- Simulating the original public keys (except one of the public keys pk). Generate the recoding keys and target public key honestly, with the help of sk.
- Simulate the original public keys. Execute $\mathsf{Sim.CHR}_{\mathsf{key}}$ to obtain the simulated recoding keys and the target public keys.

Definition 3.3 (Indistinguishability of Simulated Keys)**.** *A controlled homomorphic recoding scheme* Π *satisfies* **indistinguishability of simulated keys** *property with respect to a distribution* $\mathcal{E}_{aux}$ *if the following holds: for any collection of circuits* $\{C_{ij}\}_{i\in[L],j\in[K]}$*, control functions* $\{f_j\}_{j\in[K]}$*,*

$$\{(\mathsf{pk}_j^*, \mathsf{rk}_j)_{j\in[K]}\} \overset{s}{\approx} \{(\mathsf{Sim.pk}_j^*, \mathsf{Sim.rk}_j)_{j\in[K]}\},$$

where:

1. *For* $j \in [K]$*, compute the setup* $\mathsf{pk}_j^* \leftarrow \mathsf{CHR.Setup}(1^\lambda)$*. Then compute normal setup algorithm* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{CHR.Setup}(1^\lambda)$*. Next, for* $j \in [\ell]$*, compute* $(\mathsf{Sim.pk}_j, \tau_j) \leftarrow \mathcal{E}_{\mathsf{aux}}(1^\lambda, j)$*.*
2. *For* $j \in [K]$*, execute* $\mathsf{rk}_j \leftarrow \mathsf{DerivReKey}(\mathsf{pk}, \{\mathsf{Sim.pk}_i\}_{i\in[\ell]}, \mathsf{sk}, \{C_{ij}\}_{i\in[L]}, \mathsf{pk}_j^*, f_j)$*.*
3. *Compute* $\mathsf{Sim.CHR}_{\mathsf{key}}(\mathsf{pk}, \{\mathsf{Sim.pk}_j, \tau_j\}_{j\in[\ell]}, \{C_{ij}\}_{i\in[L],j\in[K]}, \{f_j\}_{j\in[K]})$ *to obtain the simulated recoding keys* $\{\mathsf{Sim.rk}_j\}_{j\in[K]}$ *and simulated public keys* $\{\mathsf{Sim.pk}_j^*\}_{j\in[K]}$ *associated with trapdoors* $\{\tau_j^*\}_{j\in[K]}$*.*

We refer the reader to the technical overview for a brief explanation as to why $\mathcal{E}_{aux}$ is necessary in the above definition.

Indistinguishability of Recoding Keys. This property intuitively says that it is hard to distinguish honestly generated recoding keys from simulated recoding keys. To define this formally, we first describe a simulated recoding key generation algorithm as follows:

$\mathsf{Sim.CHR}_{\mathsf{rk}}(\mathsf{pk}, \{\mathsf{Sim.pk}_i, \tau_i\}_{i\in[\ell]}, \{C_i\}_{i\in[L]}, \mathsf{pk}^*, \mathbf{P}, f, aux)$: On input public key pk, simulated public keys $\{\mathsf{Sim.pk}_i\}_{i\in[\ell]}$ with associated trapdoors $\{\tau_i\}_{i\in[\ell]}$, circuits $\{C_i\}_{i\in[L]}$, target public key pk^*, predicate $\mathbf{P}$, controlled function f and auxiliary information aux, it outputs a simulated recoding key $\mathsf{rk}_{\mathsf{sim}}$ only if the output of $\mathbf{P}(f, aux) = 1$. Otherwise, it outputs $\perp$.

Remark 3.2. *Looking ahead, in the construction of ABE for RAMs,* $\mathbf{P}(f, aux)$ *tests if the computation has terminated and if so, its output depends on the result of the computation. For instance,* $\mathbf{P}(f, aux) \neq 1$ *if the computation has terminated and it outputs 0 (meaning that the message can be decrypted in this case). And thus, we should precisely be able to simulate in the scenario where the output of the computation is not 0 or if the computation has not terminated.*

Definition 3.4 (Indistinguishability of recoding Keys)**.** *A controlled homomorphic recoding scheme* Π *satisfies* **indistinguishability of recoding keys** *property with respect to a distribution* $\mathcal{E}_{aux}$ *and predicate* $\mathbf{P}$*, if* $\{\mathsf{rk}_{\mathsf{sim}}\} \overset{s}{\approx} \{\mathsf{rk}_{\mathsf{real}}\}$*, where circuits* $C_1, \ldots, C_L$*, function* f *such that* $\mathbf{P}(f, aux) = 1$*, and we compute the distribution as:*

- *For* $i \in [\ell]$*, compute the simulated setup* $(\mathsf{Sim.pk}_i, \tau_i) \leftarrow \mathcal{E}_{aux}(1^\lambda, i)$*. Compute the setup algorithm* $\mathsf{CHR.Setup}(1^\lambda)$ *to obtain the public key-secret key pairs* $(\mathsf{pk}, \mathsf{sk})$ *and* $(\mathsf{pk}^*, \mathsf{sk}^*)$*.*
- *Compute* $\mathsf{DerivReKey}(\mathsf{pk}, \{\mathsf{Sim.pk}_i\}_{i\in[\ell]}, \mathsf{sk}, \{C_i\}_{i\in L}, \mathsf{pk}^*, f)$ *to obtain the recoding key* $\mathsf{rk}_{\mathsf{real}}$*.*
- *Compute* $\mathsf{Sim.CHR}_{\mathsf{rk}}(\mathsf{pk}, \{\mathsf{Sim.pk}_i, \tau_i\}_{i\in[\ell]}, \{C_i\}_{i\in[L]}, \mathsf{pk}^*, \mathbf{P}, f, aux)$ *to obtain the recoding key* $\mathsf{rk}_{\mathsf{sim}}$*.*

Pseudorandomness of Ciphertexts. Lastly, the pseudorandomness of ciphertexts requires that the distribution of ciphertexts is computationally close to the uniformly distribution over ciphertext space. We define the property formally below.

Definition 3.5 (Pseudorandomness of Ciphertexts)**.** *A controlled homomorphic recoding scheme* Π *is said to satisfy* **pseudorandomness of ciphertexts** *property if for any message* $y \in \mathbb{Z}$*, it is computationally hard to distinguish* $\{\mathsf{Enc}(\mathsf{pk}, y, \mathbf{s})\}$ *from uniformly random distribution over ciphertext space, where* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$*,* $\boldsymbol{s} \xleftarrow{\$} \mathcal{S}$*.*

3.2 Instantiation of CHR from Lattices

In this part, we show how to instantiate controlled homomorphic recoding scheme from lattices, particularly the LWE assumption (c.f. Definition 2.4). Then we prove the correctness of our instantiation and set the parameters in the following subsection. We describe the algorithms $\mathsf{CHR} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyEval}, \mathsf{CtEval}, \mathsf{ReEncKG}, \mathsf{ReEnc}, \mathsf{EqTest})$ below: first, we start by describing the setup and the encoding algorithms. Then we describe the homomorphism phase, followed by equality test. We postpone the description of the recoding phase to the end. Our instantiation Π is as follows:

Basic algorithms: We describe setup and encoding algorithms below.

- $\mathsf{CHR.Setup}(1^\lambda)$: On input the security parameter λ, the setup algorithm generates a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ along with its trapdoor $\mathbf{T_A}$ using
$$(\mathbf{A}, \mathbf{T_A}) \leftarrow \mathsf{TrapGen}(q, n, m)$$
Output $\mathsf{pk} = \mathbf{A}$ and $\mathsf{sk} = \mathbf{T_A}$.
- $\mathsf{CHR.Enc}(\mathsf{pk}, y, \boldsymbol{s})$: On input a public key $\mathsf{pk} = \mathbf{A} \in \mathbb{Z}_q^{n\times m}$, an attribute bit $y \in \{0, 1\}$ and a secret vector $\boldsymbol{s} \in \mathbb{Z}_q^n$, this encoding procedure outputs the ciphertext $\mathsf{ct} = (\boldsymbol{c} = \boldsymbol{s}^\mathsf{T}(\mathbf{A} + y\mathbf{G}) + \boldsymbol{e}^\mathsf{T}, y)$, where $\boldsymbol{e} \leftarrow \mathcal{D}_{\mathbb{Z}^m,\sigma}$.

Homomorphism Phase: We describe the key evaluation and ciphertext homomorphism phase below.

- $\mathsf{CHR.KeyEval}(\mathsf{pk}_1, ..., \mathsf{pk}_\ell, C)$: On input $\{\mathsf{pk}_i = \mathbf{A}_i\}_{i=1}^{\ell}$ and a circuit C, the algorithm outputs $\mathsf{pk}_C = \mathsf{PubEval}(\{\mathbf{A}_i\}_{i\in[\ell]}, C)$.
- $\mathsf{CHR.CtEval}(\mathsf{ct}_1, ..., \mathsf{ct}_\ell, C)$: On input ciphertexts $\{\mathsf{ct}_i = (\boldsymbol{c}_i, y_i)\}_{i\in[\ell]}$ encrypting $\{y_i\}_{i\in[\ell]}$ under public key $\{\mathsf{pk}_i\}_{i\in[\ell]}$ respectively and a circuit C, the algorithm outputs $\mathsf{ct} = \mathsf{CtEval}(\{\boldsymbol{c}_i\}_{i\in[\ell]}, \{y_i\}_{i\in[\ell]}, C)$.

Equality Test: We describe the equality test below.

- $\mathsf{CHR.EqTest}(\mathsf{pk}, \mathsf{ct}_1, \mathsf{ct}_2)$: On input $\mathsf{pk} = \mathbf{A}$ and $\mathsf{ct}_1, \mathsf{ct}_2$ encrypting message under public key pk, the algorithm outputs Equal if $\mathsf{Round}(\mathsf{ct}_1 - \mathsf{ct}_2) = 0$, and $\mathsf{NotEqual}$ otherwise, where function $\mathsf{Round}(\cdot)$ is defined as
$$\mathsf{Round}(x) = \begin{cases} 0, & \text{if } |x| < q/4 \\ 1, & \text{otherwise} \end{cases}$$

The control functions we support for generating recoding keys are

$$\{f_{ij}(i',j',b)|i,i'\in[N],j,j\in[T],b\in\{0,1\}\},\ \{f_i(i',b)|i,i'\in[N]\},\ \{g_i(\boldsymbol{x})|i\in[N]\},h(\cdot)$$

with descriptions as

$$f_{ij}(i',j',b)=\begin{cases}b, & \text{if } i=i'\wedge j=j'\\ \perp, & \text{otherwise}\end{cases},\quad f_i(i',b)=\begin{cases}b, & \text{if } i=i'\\ \perp, & \text{otherwise}\end{cases} \tag{1}$$

$$g_i(\boldsymbol{x})=\begin{cases}i, & \text{if } \mathcal{C}(\boldsymbol{x})=i-1\\ \perp, & \text{otherwise}\end{cases},\quad h(x)=\begin{cases}1, & \text{if } x=0\\ \perp, & \text{otherwise}\end{cases} \tag{2}$$

where $\mathcal{C}$ is a gadget circuit defined as $\mathcal{C}(\boldsymbol{x})=\sum_{i=1}^{L}x_i 2^i$, and $\boldsymbol{x}=(x_1,\ldots,x_L)$. Our recoding algorithm describe below also support the identity function, $\mathsf{Ind}(x)=x$. We use notation $\mathcal{F}$ to denote the set of control functions supported by our lattice-based instantiation, i.e.

$$\mathcal{F}=\{\mathsf{Ind},h,\{g_i\},\{f_i\},\{f_{ij}\}\} \tag{3}$$

Remark 3.3. *Jumping ahead, the controlled functions defined above are used in different scenarios in the ABE scheme, particularly in the key generation algorithm. We use the following table to illustrate their relations.*

Functions	Usage
Ind	recode current state (reading address) to next step
f_{ij}	recode value of current reading address to next step
f_i	recode writing value to current writing address
g_i	recode $(i-1)$-th time step to i-th time step
h	recode current step to final step if the program terminates at this step

Table 1: Usage of Controlled Functions in ABE Setting

Recoding Phase. We describe the recoding phase next. The details of algorithms ($\mathsf{ReEncKG}$, ReEnc) are as follows. We abuse the notation of algorithm $\mathsf{ReEncKG}$ and ReEnc by allowing they taking into several different forms of input and executing differently with respect to the inputs.

- $\mathsf{CHR.ReEncKG}$ (**Inp**): On input **Inp**, consider the following two cases:
  - *If* **Inp** *is of the form* $(\mathsf{pk}_0,\mathsf{pk}_1,\mathsf{sk}_0,\mathsf{pk}^*,\mathsf{Ind})$: Here the controlled function ind denotes the identity function, i.e. $\mathsf{ind}(\mathsf{x})=\mathsf{x}$ for any x. The public keys $\mathsf{pk}_0,\mathsf{pk}_1,\mathsf{pk}^*$, secret key sk_0 and target public public keys are parsed as follows,

    $$\mathsf{pk}_0=\mathbf{A}_0,\quad \mathsf{pk}_1=\mathbf{A}_1,\quad \mathsf{sk}_0=\mathbf{T}_{\mathbf{A}_0},\quad \mathsf{pk}^*=\mathbf{A}^*$$

    the recoding key generation algorithm computes

    $$\mathbf{R}\leftarrow\mathsf{SamplePre}(\mathbf{A}_0,\mathbf{T}_{\mathbf{A}_0},\mathbf{A}^*-\mathbf{A}_1,\sigma)$$

    such that

    $$[\mathbf{A}_0|\mathbf{A}_1]\cdot\begin{bmatrix}\mathbf{R}\\ \mathbf{I}_m\end{bmatrix}=\mathbf{A}^*$$

    Then output $\mathsf{rk}=[\mathbf{R}|\mathbf{I}_m]$.

- *If* **Inp** *is of the form* $(\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{sk}_0, \mathsf{pk}^*, h)$: The public keys $\mathsf{pk}_0, \mathsf{pk}_1$, secret key sk_0, target public key pk^*, controlled function h (c.f. Equation (2)) are parsed as follows,

$$\mathsf{pk}_0 = \mathbf{A}_0, \quad \{\mathsf{pk}_i = \mathbf{A}_i\}, \quad \mathsf{sk}_0 = \mathbf{T}_{\mathbf{A}_0}, \quad \mathsf{pk}^* = \mathbf{A}^*$$

  The recoding key generation algorithm computes

$$(\mathbf{R}_0, \mathbf{R}_1) \leftarrow \mathsf{SampleLeft}(\mathbf{A}_0, \mathbf{A}_1, \mathbf{T}_{\mathbf{A}_0}, \mathbf{A}^*, \sigma)$$

  such that

$$[\mathbf{A}_0|\mathbf{A}_1] \cdot \begin{bmatrix} \mathbf{R}_0 \\ \mathbf{R}_1 \end{bmatrix} = \mathbf{A}^*$$

  Then output $\mathsf{rk} = (\mathbf{R}_0, \mathbf{R}_1)$.
- *if* **Inp** *is of the form* $(\mathsf{pk}_0, \{\mathsf{pk}_i\}_{i=1}^L, \mathsf{sk}_0, \mathsf{pk}^*, g_i)$: The public keys $\mathsf{pk}_0, \{\mathsf{pk}_i\}_{i=1}^L$, secret key sk_0, target public key pk^*, controlled function g_i (c.f. Equation (2)) are parsed as follows,

$$\mathsf{pk}_0 = \mathbf{A}_0, \quad \{\mathsf{pk}_i = \mathbf{A}_i\}, \quad \mathsf{sk}_0 = \mathbf{T}_{\mathbf{A}_0}, \quad \mathsf{pk}^* = \mathbf{A}^*$$

  the recoding key generation algorithm first computes $\mathbf{A}_{\mathcal{C}} = \mathsf{CHR.KeyEval}(\mathsf{pk}_1, ...,$ $\mathsf{pk}_L, \mathcal{C})$, then computes

$$(\mathbf{R}_0, \mathbf{R}_1) \leftarrow \mathsf{SampleLeft}(\mathbf{A}_0, \mathbf{A}_{\mathcal{C}} + (i-1)\mathbf{G}, \mathbf{T}_{\mathbf{A}_0}, \mathbf{A}^* + i\mathbf{G}, \sigma)$$

  such that

$$[\mathbf{A}_0|\mathbf{A}_{\mathcal{C}} + (i-1)\mathbf{G}] \cdot \begin{bmatrix} \mathbf{R}_0 \\ \mathbf{R}_1 \end{bmatrix} = \mathbf{A}^* + i\mathbf{G}$$

  Then output $\mathsf{rk} = (\mathbf{R}_0, \mathbf{R}_1)$.
- *if* **Inp** *is of the form* $(\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{pk}_2, \mathsf{sk}_0, \mathsf{pk}^*, f_i)$: The public keys $\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{pk}_2$, secret key sk_0, target public key pk^*, function f_i (c.f. Equation (1)) are parsed as follows,

$$\mathsf{pk}_0 = \mathbf{A}_0, \quad \mathsf{pk}_1 = \mathbf{A}_1, \quad \mathsf{pk}_2 = \mathbf{A}_2, \quad \mathsf{sk}_0 = \mathbf{T}_{\mathbf{A}_0}, \quad \mathsf{pk}^* = \mathbf{A}^*$$

  the recoding key generation algorithm first samples $\mathbf{R}_1 \leftarrow \mathcal{D}_{\mathbb{Z}^{m\times m},\sigma}$ and then computes

$$\mathbf{R}_0 \leftarrow \mathsf{SampleLeft}(\mathbf{A}_0, \mathbf{T}_{\mathbf{A}_0}, \mathbf{A}^* - \mathbf{A}_2 - (\mathbf{A}_1 + i\mathbf{G})\mathbf{R}_1, \sigma)$$

  such that

$$[\mathbf{A}_0|\mathbf{A}_1 + i\mathbf{G}|\mathbf{A}_2] \cdot \begin{bmatrix} \mathbf{R}_0 \\ \mathbf{R}_1 \\ \mathbf{I}_m \end{bmatrix} = \mathbf{A}^*$$

  Then output $\mathsf{rk}_i = [\mathbf{R}_0|\mathbf{R}_1|\mathbf{I}_m]$.
- *If* **Inp** *is of the form* $(\mathsf{pk}_0, \mathsf{pk}_1,$ $\mathsf{pk}_2, \mathsf{pk}_3, \mathsf{sk}_0, \mathsf{pk}^*, f_{ij})$: The public keys $\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3$, secret key sk_0, target public key pk^*, function f_{ij} (c.f. Equation (1)) are parsed as follows,

$$\mathsf{pk}_0 = \mathbf{A}_0, \quad \mathsf{pk}_1 = \mathbf{A}_1, \quad \mathsf{pk}_2 = \mathbf{A}_2, \quad \mathsf{pk}_3 = \mathbf{A}_3, \quad \mathsf{sk}_0 = \mathbf{T}_{\mathbf{A}_0}, \quad \mathsf{pk}^* = \mathbf{A}^*$$

  the recoding key generation algorithm first samples $\mathbf{R}_2 \leftarrow \mathcal{D}_{\mathbb{Z}^{m\times m},\sigma}$ and then computes

$$[\mathbf{R}_0|\mathbf{R}_1] \leftarrow \mathsf{SampleLeft}(\mathbf{A}_0, \mathbf{A} + i\mathbf{G}, \mathbf{T}_{\mathbf{A}_0}, \mathbf{A}^* - \mathbf{A}_3 - (\mathbf{A}_2 + j\mathbf{G})\mathbf{R}_2, \sigma)$$

such that

$$[\mathbf{A}_0|\mathbf{A}_1 + i\mathbf{G}|\mathbf{A}_2 + j\mathbf{G}|\mathbf{A}_3] \cdot \begin{bmatrix}\mathbf{R}_0\\ \mathbf{R}_1\\ \mathbf{R}_2\\ \mathbf{I}_m\end{bmatrix} = \mathbf{A}^*$$

Then output $\mathsf{rk}_{ij} = [\mathbf{R}_0|\mathbf{R}_1|\mathbf{R}_2|\mathbf{I}_m]$.

- $\mathsf{CHR.ReEnc}(\mathbf{Inp})$: On input $\mathbf{Inp}$, consider the following two cases:
  - *If* $\mathbf{Inp}$ *is of the form* $(\mathsf{rk}, \mathsf{pk}_0, \mathsf{ct}_0, \mathsf{pk}_1, \mathsf{ct}_1)$: The recoding key rk, pairs $(\mathsf{pk}_0, \mathsf{ct}_0)$ and $(\mathsf{pk}_1, \mathsf{ct}_1)$ are parsed as follows,

    $$\mathsf{rk} = [\mathbf{R}|\mathbf{I}_m], \quad (\mathsf{pk}_0 = \mathbf{A}_0, \mathsf{ct}_0 = (\boldsymbol{c}_0, 0)), \quad (\mathsf{pk}_1 = \mathbf{A}_1, \mathsf{ct}_1 = (\boldsymbol{c}_1, y_1))$$

    the recoding algorithm computes

    $$\boldsymbol{c}_2 = (\boldsymbol{c}_0, \boldsymbol{c}_1) \cdot \begin{bmatrix}\mathbf{R}\\ \mathbf{I}_m\end{bmatrix}$$

    Output re-encrypted ciphertext $(\boldsymbol{c}_2, y_1)$.
  - *If* $\mathbf{Inp}$ *is of the form* $(\mathsf{rk}, \mathsf{pk}_0, \mathsf{ct}_0, \mathsf{pk}_1, \mathsf{ct}_1)$: The recoding key rk, pairs $(\mathsf{pk}_0, \mathsf{ct}_0)$ and $(\mathsf{pk}_1, \mathsf{ct}_1)$ are parsed as follows,

    $$\mathsf{rk} = [\mathbf{R}_0|\mathbf{R}_1], \quad (\mathsf{pk}_0 = \mathbf{A}_0, \mathsf{ct}_0 = (\boldsymbol{c}_0, 0)), \quad (\mathsf{pk}_1 = \mathbf{A}_1, \mathsf{ct}_1 = (\boldsymbol{c}_1, 0))$$

    the recoding algorithm computes

    $$\boldsymbol{c}^* = (\boldsymbol{c}_0, \boldsymbol{c}_1) \cdot \begin{bmatrix}\mathbf{R}_0\\ \mathbf{R}_1\end{bmatrix}$$

    Output re-encrypted ciphertext $(\boldsymbol{c}^*, 0)$.
  - *If* $\mathbf{Inp}$ *is of the form* $(\mathsf{rk}, \mathsf{pk}_0, \mathsf{ct}_0, \{\mathsf{pk}_j, \mathsf{ct}_j\}_{j=1}^L)$: The recoding key rk, pairs $(\mathsf{pk}_0, \mathsf{ct}_0)$ and $\{\mathsf{pk}_j, \mathsf{ct}_j\}_{j=1}^L$ are parsed as follows,

    $$\mathsf{rk} = [\mathbf{R}_0|\mathbf{R}_1], \quad (\mathsf{pk}_0 = \mathbf{A}_0, \mathsf{ct}_0 = (\boldsymbol{c}_0, 0)), \quad \{\mathsf{pk}_j = \mathbf{A}_j, \mathsf{ct}_j = (\boldsymbol{c}_j, b_j)\}_{j=1}^L$$

    the recoding algorithm first computes $\boldsymbol{c} = \mathsf{CHR.CtEval}(\boldsymbol{c}_1, \ldots, \mathsf{ct}_L, \mathcal{C})$, and then calculates

    $$\boldsymbol{c}' = (\boldsymbol{c}_0, \boldsymbol{c}) \cdot \begin{bmatrix}\mathbf{R}_0\\ \mathbf{R}_1\end{bmatrix}$$

    For $j \in [L]$, compute $\boldsymbol{c}'_j = \mathsf{CHR.CtEval}(\boldsymbol{c}', \mathcal{C}_j)$, where circuit $\mathcal{C}'_j$ converts integer i to its j-th bit. Output re-encrypted message $\{\boldsymbol{c}'_j, b'_j\}_{j=1}^L$.
  - *If* $\mathbf{Inp}$ *is of the form* $(\mathsf{rk}, \mathsf{pk}_0, \mathsf{ct}_0, \mathsf{pk}_1, \mathsf{ct}_1, \mathsf{pk}_2, \mathsf{ct}_2)$: The recoding key rk and public key/ciphertext pairs $(\mathsf{pk}_i, \mathsf{ct}_i)$ for $i \in \{0, 1, 2\}$ are parsed as follows,

    $$\mathsf{rk} = [\mathbf{R}_0|\mathbf{R}_1|\mathbf{I}_m], \ (\mathsf{pk}_0 = \mathbf{A}_0, \mathsf{ct}_0 = (\boldsymbol{c}_0, 0)), \ (\mathsf{pk}_1 = \mathbf{A}_1, \mathsf{ct}_1 = (\boldsymbol{c}_1, i)), \\ (\mathsf{pk}_2 = \mathbf{A}_3, \mathsf{ct}_2 = (\boldsymbol{c}_2, y))$$

    the recoding algorithm computes

    $$\boldsymbol{c}^* = (\boldsymbol{c}_0, \boldsymbol{c}_1, \boldsymbol{c}_2) \cdot \begin{bmatrix}\mathbf{R}_0\\ \mathbf{R}_1\\ \mathbf{I}_m\end{bmatrix}$$

    Output re-encrypted message $(\boldsymbol{c}^*, y)$.

– *If* **Inp** *is of the form* $(\mathsf{rk}, \mathsf{pk}_0, \mathsf{ct}_0, \mathsf{pk}_1, \mathsf{ct}_1, \mathsf{pk}_2, \mathsf{ct}_2, \mathsf{pk}_3, \mathsf{ct}_3)$: The recoding key rk and public key/ciphertext pairs $(\mathsf{pk}_i, \mathsf{ct}_i)$ for $i \in \{0, 1, 2, 3\}$ are parsed as follows,

$$\mathsf{rk} = [\mathbf{R}_0|\mathbf{R}_1|\mathbf{R}_2],\ (\mathsf{pk}_0 = \mathbf{A}_0, \mathsf{ct}_0 = (\boldsymbol{c}_0, 0)),\ (\mathsf{pk}_1 = \mathbf{A}_1, \mathsf{ct}_1 = (\boldsymbol{c}_1, i)),$$
$$(\mathsf{pk}_2 = \mathbf{A}_2, \mathsf{ct}_2 = (\boldsymbol{c}_2, j)),\ (\mathsf{pk}_3 = \mathbf{A}_3, \mathsf{ct}_3 = (\boldsymbol{c}_3, y))$$

the recoding algorithm computes

$$\boldsymbol{c}^* = (\boldsymbol{c}_0, \boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3) \cdot \begin{bmatrix} \mathbf{R}_0 \\ \mathbf{R}_1 \\ \mathbf{R}_2 \\ \mathbf{I}_m \end{bmatrix}$$

Output re-encrypted message $(\boldsymbol{c}^*, y)$.

3.3 Correctness and Parameters Setting

Next, we show the correctness of the instantiation Π from lattices.

Lemma 3.4. *The above instantiation* Π *of* CHR *for supported controlled function* $\mathcal{F}$ *as defined in Equation (3) from lattices is correct (c.f. Definition 3.1) given the parameters setting below.*

Proof. For $i \in \{0\} \cup [\ell]$, let $y_0 = 0, \boldsymbol{y} = (y_1, ..., y_\ell) \in \{0, 1\}^\ell$ and $(\mathsf{pk}_i = \mathbf{A}_i, \mathsf{sk}_i = \mathbf{T}_{\mathbf{A}_i}) \leftarrow \mathsf{CHR.Setup}(1^\lambda)$. For $i \in \{0\} \cup [\ell]$, encrypt the message as

$$\mathsf{ct}_i = (\boldsymbol{c}_i, y_i) \leftarrow \mathsf{CHR.Enc}(\mathsf{pk}, y_i, \boldsymbol{s})$$

where $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$ and $\boldsymbol{c}_i = \boldsymbol{s}^\mathsf{T}(\mathbf{A}_i + y_i\mathbf{G}) + \boldsymbol{e}_i^\mathsf{T}$ and $\boldsymbol{e}_i \leftarrow \mathcal{D}_{\mathbb{Z}^m,\sigma}$. Then compute auxiliary algorithms $(\mathsf{DerivReKey}, \mathsf{DerivReEnc})$ as

- $\mathsf{DerivReKey}(\mathbf{Inp})$: On input **Inp**, consider the following cases:

– if **Inp** is of the form $(\mathsf{pk}_0, \{\mathsf{pk}_i\}_{i\in[\ell]}, \mathsf{sk}_0, C, \mathsf{pk}^*, \mathsf{Ind})$: first parse

$$\{\mathsf{pk}_i = \mathbf{A}_i\}_{i\in\{0\}\cup[\ell]}, \quad \mathsf{sk}_0 = \mathbf{T}_{\mathbf{A}_0}, \quad \mathsf{pk}^* = \mathbf{A}^*$$

the algorithm evaluates $\mathsf{pk}_C \leftarrow \mathsf{CHR.KeyEval}(\{\mathsf{pk}_i\}_{i\in[\ell]}, C)$ and then computes rk by running $\mathsf{CHR.ReEncKG}\,(\mathsf{pk}_0, \mathsf{pk}_C, \mathsf{pk}^*, \mathsf{sk}_0, \mathsf{Ind})$.

– if **Inp** is of the form $(\mathsf{pk}_0, \{\mathsf{pk}_i\}_{i\in[\ell]}, \mathsf{sk}_0, C, \mathsf{pk}^*, h)$: first parse

$$\{\mathsf{pk}_i = \mathbf{A}_i\}_{i\in\{0\}\cup[\ell]}, \quad \mathsf{sk}_0 = \mathbf{T}_{\mathbf{A}_0}, \quad \mathsf{pk}^* = \mathbf{A}^*$$

the algorithm evaluates $\mathsf{pk}_C \leftarrow \mathsf{CHR.KeyEval}(\{\mathsf{pk}_i\}_{i\in[\ell]}, C)$ and then computes rk by running $\mathsf{CHR.ReEncKG}\,(\mathsf{pk}_0, \mathsf{pk}_C, \mathsf{pk}^*, \mathsf{sk}_0, h)$.

– If **Inp** is of the form $(\mathsf{pk}_0, \{\mathsf{pk}_i\}_{i=1}^L, \mathsf{sk}_0, \mathsf{pk}^*, g_i)$, then compute

$$\mathsf{rk} \leftarrow \mathsf{CHR.ReEncKG}\left(\mathsf{pk}_0, \{\mathsf{pk}_i\}_{i=1}^L, \mathsf{sk}_0, \mathsf{pk}^*, g_i\right)$$

– if **Inp** is of the form $\left(\mathsf{pk}_0, \{\mathsf{pk}_i\}_{i\in[\ell]}, \mathsf{pk}', \mathsf{sk}_0, C, \mathsf{pk}^*, f_j\right)$: On input $\mathsf{pk}_i = \mathbf{A}_i$, for $i \in \{0\} \cup [\ell]$, $\mathsf{pk}' = \mathbf{A}'$, $\mathsf{sk}_0 = \mathbf{T}_{\mathbf{A}_0}$, circuits C, a target public key $\mathsf{pk}^* = \mathbf{A}^*$ and a controlled function f_j as defined in Equation (1), the algorithm first computes $\mathsf{pk}_C \leftarrow \mathsf{CHR.KeyEval}(\{\mathsf{pk}_i\}_{i\in[\ell]}, C)$, and then runs

$$\mathsf{rk} \leftarrow \mathsf{CHR.ReEncKG}\left(\mathsf{pk}_0, \mathsf{pk}_C, \mathsf{pk}', \mathsf{sk}_0, \mathsf{pk}^*, f_j\right)$$

– if **Inp** is of the form $\left(\mathsf{pk}_0, \{\mathsf{pk}_i\}_{i\in[\ell]}, \mathsf{pk}', \mathsf{sk}_0, C_1, C_2, \mathsf{pk}^*, f_{ij}\right)$: On input $\mathsf{pk}_i = \mathbf{A}_i$, for $i \in \{0\} \cup [\ell]$, $\mathsf{pk}' = \mathbf{A}'$, $\mathsf{sk}_0 = \mathbf{T}_{\mathbf{A}_0}$, circuits $\{C_i\}_{i\in[2]}$, a target public key $\mathsf{pk}^* = \mathbf{A}^*$ and a controlled function f_{ij} as defined in Equation (1), the algorithm first computes $\mathsf{pk}_{C_i} \leftarrow \mathsf{CHR.KeyEval}(\{\mathsf{pk}_i\}_{i\in[\ell]}, C_i)$ for $i \in [2]$ and then runs

$$\mathsf{rk} \leftarrow \mathsf{CHR.ReEncKG}\left(\mathsf{pk}_0, \mathsf{pk}_{C_1}, \mathsf{pk}_{C_2}, \mathsf{pk}', \mathsf{sk}_0, \mathsf{pk}^*, f_{ij}\right)$$

- $\mathsf{DerivReEnc}(\mathbf{Inp})$: On input **Inp**, consider the following cases:

– If $\mathsf{rk} \leftarrow \mathsf{DerivReKey}\left(\mathsf{pk}_0, \{\mathsf{pk}_i\}_{i\in[\ell]}, \mathsf{sk}_0, C, \mathsf{pk}^*, \mathsf{Ind}\right)$: First evaluate the public key/ciphertext

$$\mathsf{pk}_C = \mathbf{A}_C = \mathsf{CHR.KeyEval}(\{\mathsf{pk}_i\}_{i\in[\ell]}, C)$$

$$\boldsymbol{c}_{C(\boldsymbol{y})} = \boldsymbol{s}^\top(\mathbf{A}_C + C(\boldsymbol{y})\mathbf{G}) + \boldsymbol{e}^\top = \mathsf{CHR.CtEval}(\{\mathsf{ct}_i\}_{i=1}^{\ell}, C)$$

Next, compute the recoding

$$\begin{aligned}
\boldsymbol{c}^* &= \mathsf{CHR.ReEnc}(\mathsf{rk}, \mathsf{pk}_0, \mathsf{pk}_C, \mathsf{ct}_0, \mathsf{ct}_{C(\boldsymbol{y})}) \\
&= (\boldsymbol{s}^\top[\mathbf{A}_0|\mathbf{A}_C + C(\boldsymbol{y})\mathbf{G}] + (\boldsymbol{e}_0^\top, \boldsymbol{e}^\top)) \begin{bmatrix}\mathbf{R} \\ \mathbf{I}_m\end{bmatrix} \\
&= \boldsymbol{s}^\top(\mathbf{A}^* + C(\boldsymbol{y})\mathbf{G}) + (\boldsymbol{e}_0^\top\mathbf{R} + \boldsymbol{e}^\top)
\end{aligned}$$

Output the recoded ciphertext $\mathsf{ct}^* = (\boldsymbol{c}^*, C(\boldsymbol{y}))$.

– If $\mathsf{rk} \leftarrow \mathsf{DerivReKey}\left(\mathsf{pk}_0, \{\mathsf{pk}_i\}_{i\in[\ell]}, \mathsf{sk}_0, C, \mathsf{pk}^*, h\right)$: First evaluate the public key/ciphertext

$$\mathsf{pk}_C = \mathbf{A}_C = \mathsf{CHR.KeyEval}(\{\mathsf{pk}_i\}_{i\in[\ell]}, C)$$

$$\boldsymbol{c}_{C(\boldsymbol{y})} = \boldsymbol{s}^\top(\mathbf{A}_C + 0\mathbf{G}) + \boldsymbol{e}^\top = \mathsf{CHR.CtEval}(\{\mathsf{ct}_i\}_{i=1}^{\ell}, C)$$

Next, compute the recoding

$$\begin{aligned}
\boldsymbol{c}^* &= \mathsf{CHR.ReEnc}(\mathsf{rk}, \mathsf{pk}_0, \mathsf{pk}_C, \mathsf{ct}_0, \mathsf{ct}_{C(\boldsymbol{y})}) \\
&= (\boldsymbol{s}^\top[\mathbf{A}_0|\mathbf{A}_C + 0\mathbf{G}] + (\boldsymbol{e}_0^\top, \boldsymbol{e}^\top)) \begin{bmatrix}\mathbf{R}_0 \\ \mathbf{R}_!\end{bmatrix} \\
&= \boldsymbol{s}^\top(\mathbf{A}^* + 0\mathbf{G}) + (\boldsymbol{e}_0^\top\mathbf{R} + \boldsymbol{e}^\top)
\end{aligned}$$

Output the recoded ciphertext $\mathsf{ct}^* = (\boldsymbol{c}^*, 0)$.

– If $\mathsf{rk} \leftarrow \mathsf{CHR.ReEncKG}\left(\mathsf{pk}_0, \{\mathsf{pk}_i\}_{i=1}^{L}, \mathsf{sk}_0, \mathsf{pk}^*, g_i\right)$: First evaluate the public key/ciphertext

$$\mathsf{pk}_{\mathcal{C}} = \mathbf{A}_{\mathcal{C}} = \mathsf{CHR.KeyEval}(\{\mathsf{pk}_i\}_{i\in[L]}, \mathcal{C})$$

$$\boldsymbol{c}_{\mathcal{C}\boldsymbol{x})} = \boldsymbol{s}^\top(\mathbf{A}_C + (i-1)\mathbf{G}) + \boldsymbol{e} = \mathsf{CHR.CtEval}(\mathsf{ct}_1, ..., \mathsf{ct}_L, C)$$

Then, compute the recoding

$$\begin{aligned}
\boldsymbol{c}^* &= \mathsf{CHR.ReEnc}(\mathsf{rk}, \mathsf{pk}_0, \mathsf{pk}_{\mathcal{C}}, \mathsf{ct}_0, \mathsf{ct}_{\mathcal{C}(\boldsymbol{x})}) \\
&= (\boldsymbol{s}^\top[\mathbf{A}_0|\mathbf{A}_C + (i-1)\mathbf{G}] + (\boldsymbol{e}_0^\top, \boldsymbol{e}^\top)) \begin{bmatrix}\mathbf{R}_0 \\ \mathbf{R}_1\end{bmatrix} \\
&= \boldsymbol{s}^\top(\mathbf{A}^* + i\mathbf{G}) + (\boldsymbol{e}_0^\top\mathbf{R}_0 + \boldsymbol{e}^\top\mathbf{R}_1)
\end{aligned}$$

Next, for $j \in [L]$, compute $\boldsymbol{c}'_j = \mathsf{CHR.CtEval}(\boldsymbol{c}', \mathcal{C}_j)$, where circuit $\mathcal{C}'_j$ converts integer i to its j-th bit. Output recoding $\{\boldsymbol{c}'_j, b'_j\}_{j=1}^{L}$, where $\{b'_j\}_{j=1}^{L}$ is the bit-representation of i.

– If $\mathsf{rk} \leftarrow \mathsf{DerivReKey}\big(\mathsf{pk}_0, \mathsf{pk}_C, \mathsf{pk}', \mathsf{pk}^*, \mathsf{sk}_0, f_j\big)$: First evaluate public key/ciphertext as

$$\mathsf{pk}_C = \mathbf{A}_{C_i} \leftarrow \mathsf{CHR.KeyEval}(\{\mathsf{pk}_i\}_{i\in[\ell]}, C_i)$$

$$\boldsymbol{c}_{C(\boldsymbol{y})} = \boldsymbol{s}^\mathsf{T}(\mathbf{A}_C + C(\boldsymbol{y})\mathbf{G}) + \boldsymbol{e}^\mathsf{T} \leftarrow \mathsf{CtEval}(\{\mathsf{ct}_i\}_{i=1}^{\ell}, C)$$

where $C(\boldsymbol{y}) = i$. Let $\mathsf{ct}' = (\boldsymbol{c}', b) = \mathsf{Enc}(\mathsf{pk}', b, \boldsymbol{s})$, and $\boldsymbol{c}' = \boldsymbol{s}^\mathsf{T}(\mathbf{A}' + b\mathbf{G}) + \boldsymbol{e}'^\mathsf{T}$. Next, compute

$$\begin{aligned}\boldsymbol{c}^* &= \mathsf{CHR.ReEnc}(\mathsf{rk}, \mathsf{pk}_0, \mathsf{ct}_0, \{\mathsf{pk}_{C_i}, \mathsf{ct}_{C_i(\boldsymbol{y})}\}_{i\in[2]}, \mathsf{pk}', \mathsf{ct}') \\ &= (\boldsymbol{s}^\mathsf{T}[\mathbf{A}_0|\mathbf{A}_{C_1} + i\mathbf{G}|\mathbf{A}_{C_2} + j\mathbf{G}|\mathbf{A}' + b\mathbf{G}] + (\boldsymbol{e}_0^\mathsf{T}, \boldsymbol{e}_1^\mathsf{T}, \boldsymbol{e}'^\mathsf{T}))\begin{bmatrix}\mathbf{R}_0\\ \mathbf{R}_1\\ \mathbf{I}_m\end{bmatrix} \\ &= \boldsymbol{s}^\mathsf{T}(\mathbf{A}^* + b\mathbf{G}) + (\boldsymbol{e}_0^\mathsf{T}\mathbf{R}_0 + \boldsymbol{e}_1^\mathsf{T}\mathbf{R}_1 + \boldsymbol{e}'^\mathsf{T})\end{aligned}$$

Output recoded ciphertext as $\mathsf{ct}^* = (\boldsymbol{c}^*, b)$.

– If $\mathsf{rk} \leftarrow \mathsf{DerivReKey}\big(\mathsf{pk}_0, \{\mathsf{pk}_i\}_{i\in[\ell]}, \mathsf{pk}', \mathsf{sk}_0, C_1, C_2, \mathsf{pk}^*, f_{ij}\big)$: For $i \in [2]$, compute $\mathsf{pk}_{C_i} = \mathbf{A}_{C_i} \leftarrow \mathsf{CHR.KeyEval}(\{\mathsf{pk}_i\}_{i\in[\ell]}, C_i)$. Then for $i \in [2]$, evaluate ciphertexts as

$$\boldsymbol{c}_{C_i(\boldsymbol{y})} = \boldsymbol{s}^\mathsf{T}(\mathbf{A}_C + C_i(\boldsymbol{y})\mathbf{G}) + \boldsymbol{e}_1^\mathsf{T} \leftarrow \mathsf{CtEval}(\{\mathsf{ct}_i\}_{i=1}^{\ell}, C)$$

where $C_1(\boldsymbol{y}) = i$ and $C_2(\boldsymbol{y}) = j$. Let $\mathsf{ct}' = (\boldsymbol{c}', b) = \mathsf{Enc}(\mathsf{pk}', b, \boldsymbol{s})$, and $\boldsymbol{c}' = \boldsymbol{s}^\mathsf{T}(\mathbf{A}' + b\mathbf{G}) + \boldsymbol{e}'^\mathsf{T}$. Next, compute

$$\begin{aligned}\boldsymbol{c}^* &= \mathsf{CHR.ReEnc}\big(\mathsf{rk}, (\mathsf{pk}_0, \mathsf{ct}_0), \{\mathsf{pk}_{C_i}, \mathsf{ct}_{C_i(\boldsymbol{y})}\}_{i\in[2]}, (\mathsf{pk}', \mathsf{ct}')\big) \\ &= (\boldsymbol{s}^\mathsf{T}[\mathbf{A}_0|\mathbf{A}_{C_1} + i\mathbf{G}|\mathbf{A}_{C_2} + j\mathbf{G}|\mathbf{A}' + b\mathbf{G}] + (\boldsymbol{e}_0^\mathsf{T}, \boldsymbol{e}_1^\mathsf{T}, \boldsymbol{e}_2^\mathsf{T}, \boldsymbol{e}'^\mathsf{T}))\begin{bmatrix}\mathbf{R}_0\\ \mathbf{R}_1\\ \mathbf{R}_2\\ \mathbf{I}_m\end{bmatrix} \\ &= \boldsymbol{s}^\mathsf{T}(\mathbf{A}^* + b\mathbf{G}) + (\boldsymbol{e}_0^\mathsf{T}\mathbf{R}_0 + \boldsymbol{e}_1^\mathsf{T}\mathbf{R}_1 + \boldsymbol{e}_2^\mathsf{T}\mathbf{R}_2 + \boldsymbol{e}'^\mathsf{T})\end{aligned}$$

Output recoded ciphertext as $\mathsf{ct}^* = (\boldsymbol{c}^*, b)$.

If we encrypt the message freshly under target public key, i.e. $\mathsf{ct}^*_{\mathsf{fresh}} = (\boldsymbol{c}^*_{\mathsf{fresh}}, b) = \mathsf{Enc}(\mathsf{pk}^*, \boldsymbol{s}, b)$, the vector $\boldsymbol{c}^*_{\mathsf{fresh}} = \boldsymbol{s}^\mathsf{T}(\mathbf{A}^* + b\mathbf{G}) + \boldsymbol{e}^\mathsf{T}$, where $\boldsymbol{e} \leftarrow \mathcal{D}_{\mathbb{Z}^m,\sigma}$. Thus, we have

$$\lfloor \boldsymbol{c}^* - \boldsymbol{c}^*_{\mathsf{fresh}} \rfloor = \lfloor \mathsf{error} \rfloor$$

where error can be various forms in different settings, and the in the most complex setting, $\mathsf{error} = \boldsymbol{e}_0^\mathsf{T}\mathbf{R}_0 + \boldsymbol{e}_1^\mathsf{T}\mathbf{R}_1 + \boldsymbol{e}_2^\mathsf{T}\mathbf{R}_2 + \boldsymbol{e}'^\mathsf{T} - \tilde{\boldsymbol{e}}^\mathsf{T}$. By setting parameters appropriately in the following, we have $\mathsf{CHR.EqTest}(\mathsf{ct}^*, \tilde{\mathsf{ct}}) = \mathsf{Equal}$. □

Parameter Setting. We set the parameters in the instantiation as follows: For decryption (or equal test) to work correctly, the modulus q should be slightly larger than the noise accumulated in the ciphertext. If the circuit being evaluated has depth d, the noise in the ciphertexts grows in the worst case by a factor of $O(m^d)$. Hence, we need q be the order of $\Omega(Bm^d)$, where B is the maximum magnitude of noise added during encryption. The hardness of LWE assumption requires that the ratio q/B is not too large. The LWE problem is believed to be hard even when q/B is 2^{n^ϵ} for some fixed $0 < \epsilon < 1/2$.

To support circuits of depth $d(\lambda)$ for some polynomial $d(\cdot)$, we set $n = \tilde{\Theta}(d^{1/\epsilon})$, modulus $q = 2^{n^\epsilon}$, dimension $m = \Theta(n \log q)$, LWE noise bound $B = O(n)$ and Gaussian parameter $\sigma = O(\sqrt{n \log q})$.

3.4 Security Proof

In this part, we show that our instantiation Π of controlled homomorphic recoding scheme from lattices satisfies the security definitions in Section 3.1.1, namely indistinguishability of setup, indistinguishability of simulated keys, indistinguishability of recoding keys and pseudorandomness of ciphertexts.

Indistinguishability of Setup. First, we describe the simulated setup algorithm $\mathsf{Sim.CHRSetup}(1^\lambda, z)$ as follows:

$\mathsf{Sim.CHRSetup}(1^\lambda, z)$: The simulated setup algorithm randomly chooses matrices $\mathbf{A}' \leftarrow \mathbb{Z}_q^{n\times m}, \mathbf{S} \leftarrow \{-1,1\}^{m\times m}$, and outputs simulated public key $\mathsf{Sim.pk} = \mathbf{A} = \mathbf{A}'\mathbf{S} - z\cdot\mathbf{G}$ and trapdoor $\mathbf{S}$.

We argue the statistical indistinguishability between the distribution of normally generated public keys and simulated public keys in the following lemma.

Lemma 3.5. *The instantiation* Π *of controlled homomorphic recoding scheme satisfies* **indistinguishability of setup** *(c.f. Definition 3.2).*

Proof. The difference between normal setup algorithm $\mathsf{Setup}(1^\lambda)$ and simulated setup algorithm $\mathsf{Sim.CHRSetup}(1^\lambda, z)$ is that in $\mathsf{Setup}(1^\lambda)$, the $\mathsf{pk} = \mathbf{A} \in \mathbb{Z}_q^{n\times m}$ is generated by algorithm $\mathsf{TrapGen}(q, n, m)$ and in $\mathsf{Sim.CHRSetup}(1^\lambda, z)$, we compute $\mathsf{Sim.pk} = \tilde{\mathbf{A}} = \mathbf{A}'\mathbf{S} - z\mathbf{G}$, where matrices $\mathbf{A}' \stackrel{\$}{\leftarrow} \mathbb{Z}_q^{n\times m}, \mathbf{S} \stackrel{\$}{\leftarrow} \{-1,1\}^{m\times m}$. By property of algorithm $\mathsf{TrapGen}$ as stated in Lemma 2.2, the output distribution of $\mathbf{A}$ is statistically close to uniform distribution. By Leftover Hash Lemma 2.4, the distribution of $\mathbf{A}'\mathbf{S}$ is statistically close to uniform distribution given the facts that $\mathbf{A}' \stackrel{\$}{\leftarrow} \mathbb{Z}_q^{n\times m}, \mathbf{S} \stackrel{\$}{\leftarrow} \{-1,1\}^{m\times m}$. Therefore, we have that the distribution $\{\mathsf{pk}\}$ is statistically close to $\{\mathsf{Sim.pk}\}$. □

Generalization of $\mathsf{Sim.GenCHRSetup}$: We can further generalize the simulated setup by augmenting its input as $\mathsf{Sim.CHRSetup}(1^\lambda, \boldsymbol{z}, \ell; \mathbf{A})$, where $\boldsymbol{z} \in \{0,1\}^\ell$, and $\mathbf{A}$ is used in a similar way as $\mathbf{A}'$ in algorithm $\mathsf{Sim.CHRSetup}(1^\lambda, z)$.

$\mathsf{Sim.GenCHRSetup}(\mathbf{Inp})$: On input $\mathbf{Inp}$, consider the following two cases:

- If $\mathbf{Inp}$ is of form $(1^\lambda, z)$, then run $\mathsf{Sim.CHRSetup}(1^\lambda, z)$.
- If $\mathbf{Inp}$ is of form $(1^\lambda, \boldsymbol{z}, \ell; \mathbf{A})$, then for $i \in [\ell]$, choose $\mathbf{S}_i \stackrel{\$}{\leftarrow} \{-1,1\}^{m\times m}$, and set $\mathsf{Sim.pk}_i = \mathbf{A}_i = \mathbf{A}\mathbf{S}_i - z_i\mathbf{G}$, where z_i is i-th bit of vector $\boldsymbol{z}$. Output $\{\mathsf{Sim.pk}_i\}_{i\in[\ell]}$ and trapdoors $\{\mathbf{S}_i\}_{i=1}^{\ell}$.

Similarly, we can also show that the distribution of public keys generalized by $\mathsf{Sim.GenCHRSetup}(\mathbf{Inp})$ is statistically close to the distribution of running normal setup algorithm ℓ times.

Corollary 3.6. *The instantiation of controlled homomorphic recoding scheme satisfies* **indistinguishability of setup**.

The proof of the above corollary is very similar to the proof of Lemma 3.5, thus we omit it here.

Remark 3.7. *Looking ahead, the sequence of integers* $\boldsymbol{z}$ *to be programmed in the generalized case corresponds to challenge database committed by the adversary in the ABE setting.*

Corollary 3.8. *The distribution of (regularly generated or simulated) public keys is indistinguishable from uniformly random distribution over space* $\mathbb{Z}_q^{n\times m}$.

Proof. For regularly generated public keys, they are computed by algorithm $\mathsf{TrapGen}$. By Lemma 2.4, the distribution of regularly generated public keys is statistically close to random distribution. For simulated public keys, they are computed as $\mathbf{A}'\mathbf{S} - z\mathbf{G}$, where $\mathbf{A}'$ is chosen from random, and $\mathbf{S}$ is chosen from distribution $\mathcal{D}_{\mathbb{Z}^{n\times m},\sigma}$. Again by Lemma 2.4, the distribution of simulated keys is statistically close to random. □

Indistinguishability of Simulated Keys. We first describe the simulated key generation algorithm $\mathsf{Sim.CHR_{key}}(\mathsf{pk}, \{\mathsf{Sim.pk}_i, \tau_i\}_{i\in[\ell]}, \{C_i\}_{i\in[L]}, \{y_i\}_{i=1}^{\ell}, f)$. The circuits $\{C_i\}_{i\in[L]}$ are defined as $C_i : \{0,1\}^{\ell} \to \{0,1\}$. To be consistent with our instantiation of controlled homomorphic recoding scheme where we allow algorithm $\mathsf{DerivReKey}$ to take into two different forms of inputs (c.f. proof of Lemma 3.4). For simplicity, we only consider the case where $L = 0, 1, 2$ and variant controlled function f, which is how we define algorithm $\mathsf{DerivReKey}$ as above. This proof can be generalized to the case where L is any arbitrary integers.

Case. $L = 0$ **and** $f = g_i$ **(c.f. Equation (2)).** $\mathsf{Sim.CHR_{key}}(\mathsf{pk}, \{\mathsf{Sim.pk}_i, \tau_i\}_{i\in[\ell]}, \{b_i\}_{i=1}^{\ell}, g_i)$: First parse part of the input as $\mathsf{pk} = \mathbf{A}, \mathsf{Sim.pk}_i = \mathbf{A}_i = \mathbf{A}\mathbf{S}_i - b_i\mathbf{G}, \tau_i = \mathbf{S}_i$, where $\mathcal{C}(\{b_i\}_{i=1}^{\ell}) = i$. Evaluate public key as $\mathbf{A}_{\mathcal{C}} = \mathbf{A}\mathbf{S}_{\mathcal{C}} = \mathsf{KeyEval}(\{\mathsf{Sim.pk}_i\}_{i\in[\ell]}, \mathcal{C})$

- If $i = 1$, then sample matrices $\mathbf{R}_0, \mathbf{R}_1$ from $\mathcal{D}_{\mathbb{Z}^{m\times m},\sigma}$ and output $\mathsf{Sim.rk} = [\mathbf{R}_0|\mathbf{R}_1]$, and $(\mathsf{Sim.pk}_i^*, \tau_i^*) = (\mathsf{KeyEval}(\mathbf{A}^*, \mathcal{C}_i), \mathsf{TrapEval}(\mathbf{R}_0 + \mathbf{S}_{\mathcal{C}}, \mathcal{C}_i))$ for $i \in [\ell]$, where $\mathbf{A}^* = \mathbf{A}(\mathbf{R}_0 + \mathbf{S}_{\mathcal{C}}\mathbf{R}_1)$.
- If $i > 1$, then generate $(\mathsf{Sim.pk}, \tau^*) = (\mathbf{A}^*, \mathbf{S}^*) \leftarrow \mathsf{Sim.GenCHRSetup}(1^{\lambda})$. Then sample $[\mathbf{R}_0|\mathbf{R}_1]$, using
$$[\mathbf{R}_0|\mathbf{R}_1] \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{S}_{\mathcal{C}}, \mathbf{T}_{\mathbf{G}}, \mathbf{A}^*, \sigma)$$
output $\mathsf{Sim.rk} = [\mathbf{R}_0|\mathbf{R}_1]$, and $(\mathsf{Sim.pk}_i^*, \tau_i^*) = (\mathsf{KeyEval}(\mathbf{A}^*, \mathcal{C}_i), \mathsf{KeyEval}(\mathbf{S}^*, \mathcal{C}_i))$, for $i \in [\ell]$.

Case. $L = 1$ **and** $f = \mathsf{Ind}$. $\mathsf{Sim.CHR_{key}}(\mathsf{pk}, \{\mathsf{Sim.pk}_i, \tau_i\}_{i\in[\ell]}, \boldsymbol{y}, C, \mathsf{Ind})$: First parse part of the input as $\mathsf{pk} = \mathbf{A}, \mathsf{Sim.pk}_i = \mathbf{A}_i = \mathbf{A}\mathbf{S}_i - y_i\mathbf{G}$ and $\tau = \mathbf{S}_i$. Evaluate public key as $\mathbf{A}_C = \mathbf{A}\mathbf{S}_C - C(\boldsymbol{y})\mathbf{G} = \mathsf{KeyEval}(\{\mathsf{Sim.pk}_i\}_{i\in[\ell]}, C)$, and sample a random matrix $\mathbf{R}$ from $\mathcal{D}_{\mathbb{Z}^{m\times m}}$. Output $\mathsf{Sim.rk} = [\mathbf{R}|\mathbf{I}_m]$, $\mathsf{Sim.pk} = \mathbf{A}(\mathbf{R} + \mathbf{S}_C) - C(\boldsymbol{y})\mathbf{G}$ and its trapdoor $\mathbf{R} + \mathbf{S}_C$.

Case. $L = 1$ **and** $f = f_i$ (c.f. Equation (1)). $\mathsf{Sim.CHR_{key}}(\mathsf{pk}, \{\mathsf{Sim.pk}_i, \tau_i\}_{i\in[\ell]}, \mathsf{Sim.pk}', \boldsymbol{y}, C, f_i)$: First parse part of the input as $\mathsf{pk} = \mathbf{A}, \mathsf{Sim.pk}_i = \mathbf{A}_i = \mathbf{A}\mathbf{S}_i - y_i\mathbf{G}, \tau_i = \mathbf{S}_i$, and $\mathsf{Sim.pk}' = \mathbf{A}\mathbf{S}' - b\mathbf{G}$. Evaluate public key as $\mathbf{A}_C = \mathbf{A}\mathbf{S}_C - C(\boldsymbol{y})\mathbf{G} = \mathsf{KeyEval}(\{\mathsf{Sim.pk}_i\}_{i\in[\ell]}, C_i)$.

- If $C(\boldsymbol{y}) = i$, sample matrices $\mathbf{R}_0, \mathbf{R}_1$ from $\mathcal{D}_{\mathbb{Z}^{m\times m},\sigma}$ and output $\mathsf{Sim.pk}^* = \mathbf{A}(\mathbf{R}_0 + \mathbf{S}_C\mathbf{R}_1 + \mathbf{S}') - b\mathbf{G}$, its trapdoor $(\mathbf{R}_0 + \mathbf{S}_C\mathbf{R}_1 + \mathbf{S}')$, and $\mathsf{Sim.rk} = [\mathbf{R}_0|\mathbf{R}_1|\mathbf{I}_m]$.
- If $C(\boldsymbol{y}) \neq i$, generate $\mathsf{Sim.pk} = \mathbf{A}^* = \mathbf{A}\mathbf{S}^* \leftarrow \mathsf{Sim.CHRSetup}(\mathbf{A})$. Then sample $[\mathbf{R}_0|\mathbf{R}_1]$, using
$$[\mathbf{R}_0|\mathbf{R}_1] \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{S}_{C_1}, \mathbf{T}_{\mathbf{G}}, \mathbf{A}^* - \mathbf{A}', \sigma)$$
output $\mathsf{Sim.pk} = \mathbf{A}^*$, its trapdoor $\mathbf{S}^*$, and $\mathsf{Sim.rk} = [\mathbf{R}_0|\mathbf{R}_1|\mathbf{I}_m]$.

Case. $L = 2$ **and** $f = f_{jk}$ **(c.f. Equation (1)).** $\mathsf{Sim.CHR_{key}}(\mathsf{pk}, \{\mathsf{Sim.pk}_i\}_{i\in[\ell]}, \mathsf{Sim.pk}', \boldsymbol{y}, b, C_1, C_2, f_{jk})$: First parse part of the input as $\mathsf{pk} = \mathbf{A}, \mathsf{Sim.pk}_i = \mathbf{A}_i = \mathbf{A}\mathbf{S}_i - y_i\mathbf{G}$, and $\mathsf{Sim.pk}' = \mathbf{A}' = \mathbf{A}\mathbf{S}' - b\mathbf{G}$. Evaluate public key as $\mathbf{A}_{C_i} = \mathbf{A}\mathbf{S}_{C_i} - C_i(\boldsymbol{y})\mathbf{G} = \mathsf{KeyEval}(\{\mathsf{Sim.pk}_i\}_{i\in[\ell]}, C_i)$, for $i = 1, 2$.

- If $C_1(\boldsymbol{y}) = j$ and $C_2(\boldsymbol{y}) = k$, sample matrices $\mathbf{R}_0, \mathbf{R}_1, \mathbf{R}_2$ from $\mathcal{D}_{\mathbb{Z}^{m\times m},\sigma}$ and output $\mathsf{Sim.pk} = \mathbf{A}(\mathbf{R}_0 + \mathbf{S}_{C_1}\mathbf{R}_1 + \mathbf{S}_{C_2}\mathbf{R}_2 + \mathbf{S}') - b\mathbf{G}$, its trapdoor $(\mathbf{R}_0 + \mathbf{S}_{C_1}\mathbf{R}_1 + \mathbf{S}_{C_2}\mathbf{R}_2 + \mathbf{S}')$, and $\mathsf{Sim.rk} = [\mathbf{R}_0|\mathbf{R}_1|\mathbf{R}_2|\mathbf{I}_m]$.
- If $C_1(\boldsymbol{y}) = j$ and $C_2(\boldsymbol{y}) \neq k$ (or the other case), generate $\mathsf{Sim.pk} = \mathbf{A}^* = \mathbf{AS}^* \leftarrow \mathsf{Sim.CHRSetup}(\mathbf{A})$. First sample a matrix $\mathbf{R}_1$ from $\mathcal{D}_{\mathbb{Z}^{m\times m},\sigma}$, and then sample $[\mathbf{R}_0|\mathbf{R}_2]$, using
$$[\mathbf{R}_0|\mathbf{R}_2] \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{S}_{C_2}, \mathbf{T}_\mathbf{G}, \mathbf{A}^* - \mathbf{AS}_{C_1}\mathbf{R}_1, \sigma)$$
output $\mathsf{Sim.pk} = \mathbf{A}^*$, its trapdoor $\mathbf{S}^*$, and $\mathsf{Sim.rk} = [\mathbf{R}_0|\mathbf{R}_1|\mathbf{R}_2|\mathbf{I}_m]$.
- If $C_1(\boldsymbol{y}) \neq j$ and $C_2(\boldsymbol{y}) \neq k$, generate $\mathsf{Sim.pk} = \mathbf{A}^* = \mathbf{AS}^* \leftarrow \mathsf{Sim.CHRSetup}(\mathbf{A})$. First sample a matrix $\mathbf{R}_1$ from $\mathcal{D}_{\mathbb{Z}^{m\times m},\sigma}$, and then sample $[\mathbf{R}_0|\mathbf{R}_2]$, using
$$[\mathbf{R}_0|\mathbf{R}_2] \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{S}_{C_2}, \mathbf{T}_\mathbf{G}, \mathbf{A}^* - \mathbf{A}_{C_1}\mathbf{R}_1, \sigma)$$
output $\mathsf{Sim.pk} = \mathbf{A}^*$, its trapdoor $\mathbf{S}^*$, and $\mathsf{Sim.rk} = [\mathbf{R}_0|\mathbf{R}_1|\mathbf{R}_2|\mathbf{I}_m]$.

The indistinguishability of simulated keys property is proved below:

Lemma 3.9. *The instantiation* Π *of controlled homomorphic recoding scheme satisfies* **indistinguishability of simulated keys** *(c.f. Definition 3.3) with respect to* $\mathcal{E}_{\mathsf{aux}}$*, where* $\mathcal{E}_{\mathsf{aux}}$ *is defined as follows:*

- *Case I:* $\mathcal{E}_{\mathsf{aux}}$ *is the same as* $\mathsf{Sim.CHRSetup}(1^\lambda, z)$*, where* aux *corresponds to value being programmed,* z*.*
- *Case II:* $\mathcal{E}_{\mathsf{aux}}$ *is the same as* $\mathsf{Sim.CHR_{key}}$*.*

Proof. The difference between the normal key generation $\mathsf{DerivReKey}$ and simulated key generation $\mathsf{Sim.CHR_{key}}$ are summarized as below:

- In algorithm $\mathsf{DerivReKey}$, the target public key $\mathbf{A}^*$ is given as random matrix over $\mathbb{Z}_q^{n\times m}$ and recoding keys using the secret key $\mathsf{sk} = \mathbf{T}_\mathbf{A}$ of $\mathsf{pk} = \mathbf{A}$ via algorithm $\mathsf{SamplePre}$ or $\mathsf{SampleLeft}$.
- In algorithm $\mathsf{Sim.CHR_{key}}$, the target public key is not given as input, but generated via variant computing methods as listed above, and recoding keys are sampled from Gaussian distribution $\mathcal{D}_{\mathbb{Z}^{m\times m}}$ (via direct sampling or algorithm $\mathsf{SampleRight}$).

By Leftover Hash Lemma 2.4, the distribution $(\mathbf{A}, \mathbf{A}^*)$ is statistically close to the distribution $(\mathbf{A}, \mathbf{AR}_0)$. And by the properties of algorithms $\mathsf{SamplePre}, \mathsf{SampleLeft}$ and $\mathsf{SampleRight}$ as stated in Lemma 2.3, their outputs are statistically close to discrete Gaussian distribution $\mathcal{D}_{\mathbb{Z}^{m\times m},\sigma}$. This statement holds simulated public keys from algorithm $\mathcal{E}_{aux}$ (the two cases defined above), where the simulated public keys are computed from algorithms $\mathsf{Sim.CHR_{key}}$ or $\mathsf{Sim.CHRSetup}$. Therefore, we have that $(\mathsf{Sim.pk}^*, \mathsf{rk}) \overset{c}{\approx} (\mathsf{Sim.pk}, \mathsf{Sim.rk})$, and thus we prove our instantiation of controlled homomorphic recoding scheme satisfies indistinguishability of simulated keys. □

Generalization of $\mathsf{Sim.CHR_{key}}$: We note that we can generalize algorithm $\mathsf{Sim.CHR_{key}}(\mathsf{pk}, \{\mathsf{Sim.pk}_i\}_{i\in[\ell]}, \mathsf{pk}', \boldsymbol{y}, C_1, C_2, f_{jk})$ to generate a sequence of simulated recoding keys $\{\mathsf{Sim.rk}_{ij}\}_{i\in[N],j\in[L]}$ (for some integers N, L as the range of circuits C_1, C_2 respectively) and one target simulated key:

$\mathsf{Sim.GenCHR_{key}}(\mathsf{pk}, \{\mathsf{Sim.pk}_i\}_{i\in[\ell]}, \mathsf{pk}', \boldsymbol{y}, C_1, C_2, N, L)$: First Evaluate $y_1 = C_1(\boldsymbol{y})$ and $y_2 = C(\boldsymbol{y})$. Then compute $(\mathsf{Sim.pk}^*, \mathsf{Sim.rk}_{y_1y_2}) \leftarrow \mathsf{Sim.CHR_{key}}(\mathsf{pk}, \{\mathsf{Sim.pk}_i\}_{i\in[\ell]}, \mathsf{pk}', \boldsymbol{y}, C_1, C_2, f_{y_1y_2})$, where $\mathsf{Sim.pk}^* = \mathbf{A}^*$. Then for $j \in [L], k \in [N]$,

- If $j = y_1$ and $k \neq y_2$, sample $\mathbf{R}_{jk1}$ from $\mathcal{D}_{\mathbb{Z}^{m\times m},\sigma}$, then compute

$$[\mathbf{R}_{jk0}|\mathbf{R}_{jk2}] \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{S}_{C_2}, \mathbf{T}_{\mathbf{G}}, \mathbf{A}^* - \mathbf{A}\mathbf{S}_{C_1}\mathbf{R}_{jk1}, \sigma)$$

- If $j \neq y_1$ and $k = y_2$, sample $\mathbf{R}_{jk2}$ from $\mathcal{D}_{\mathbb{Z}^{m\times m},\sigma}$, then compute

$$[\mathbf{R}_{jk0}|\mathbf{R}_{jk1}] \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{S}_{C_1}, \mathbf{T}_{\mathbf{G}}, \mathbf{A}^* - \mathbf{A}\mathbf{S}_{C_2}\mathbf{R}_{jk2}, \sigma)$$

- If $j \neq y_1$ and $k \neq y_2$, sample $\mathbf{R}_{jk1}$ from $\mathcal{D}_{\mathbb{Z}^{m\times m},\sigma}$, then compute

$$[\mathbf{R}_{jk0}|\mathbf{R}_{jk2}] \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{S}_{C_2}, \mathbf{T}_{\mathbf{G}}, \mathbf{A}^* - (\mathbf{A}\mathbf{S}_{C_1} - (y_1 - j)\mathbf{G})\mathbf{R}_{jk1}, \sigma)$$

Output simulated recoding keys and target public key as $\{\mathsf{Sim.rk}_{ij}\}_{i\in[N],j\in[L]}$ and $\mathsf{Sim.pk}$.

Corollary 3.10. *The generalized instantiation of controlled homomorphic recoding scheme satisfies* **indistinguishability of simulated keys** *(c.f. Definition 3.3) with respect to $\mathcal{E}_{\mathsf{aux}}$ as defined in Lemma 3.9.*

The proof of indistinguishability of simulated keys from the generalized algorithm is very similar to the proof of Lemma 3.9, thus we omit it here.

Remark 3.11. *Looking ahead, the above generalization of the algorithm $\mathsf{Sim.CHR}_{\mathsf{key}}$ will be used in the ABE scheme to generate the recoding keys for translating the output of the step circuit in the i-th step into the $(i+1)$-th step.*

Indistinguishability of Recoding Keys. We first describe the simulated recoding key generation algorithm $\mathsf{Sim.CHR}_{\mathsf{rk}}(\mathsf{pk}, \{\mathsf{Sim.pk}_i\}_{i\in[\ell]}, \{C_i\}_{i\in[L]}, \boldsymbol{y}, \mathsf{pk}^*, f)$. In the inputs, for $i \in [\ell]$, the circuit $C_i : \{0,1\}^\ell \to \{0,1\}$ and f is defined as

$$f : \{0,1\}^L \to \{0,1\}, \quad f(\{x_i\}_{i\in[L]}) = \begin{cases} 0, & \text{if } \wedge_{i=1}^{L} \bar{x}_i = 1 \\ 1, & \text{otherwise} \end{cases} \tag{4}$$

Remark 3.12. *Looking ahead, in the ABE scheme, the function f will be used to signal whether the output of the computation is all zero. If the output is all zero (earlier than the upper time bound T), then f outputs 0.*

$\mathsf{Sim.CHR}_{\mathsf{rk}}(\mathsf{pk}, \{\mathsf{Sim.pk}_i, \tau_i\}_{i\in[\ell]}, \{C_i\}_{i\in[L]}, \boldsymbol{y}, \mathsf{pk}^*, f)$: If $f(C_1(\boldsymbol{y}), \ldots, C_L(\boldsymbol{y})) = 0$, then output $\perp$. Otherwise, parse the input as

$$\mathsf{pk} = \mathbf{A}, \quad \{\mathsf{Sim.pk}_i = \mathbf{A}_i\}_{i\in[\ell]} \leftarrow \mathsf{Sim.CHRSetup}(1^\lambda, \boldsymbol{y}, \ell; \mathbf{A}), \quad \mathsf{pk}^* = \mathbf{A}^*$$

where $\mathbf{A}_i = \mathbf{A}\mathbf{S}_i - y_i\mathbf{G}$ and its trapdoor $\tau_i = \mathbf{S}_i$. First for $i \in [L]$, evaluate public key as $\mathbf{A}_{C_i} = \mathbf{A}\mathbf{S}_{C_i} - C_i(\boldsymbol{y}) = \mathsf{KeyEval}(\{\mathsf{Sim.pk}_j\}_{j\in[\ell]}, C_i)$. Since $f(\{C_i(\boldsymbol{y})\}_{i\in[L]}) = 1$, then there exists an index $k \in [L]$, such that $C_k(\boldsymbol{y}) = 1$. For $i \in [L] - \{k\}$, sample matrices $\mathbf{R}_i \leftarrow \mathcal{D}_{\mathbb{Z}^{m\times m},\sigma}$, and sample $[\mathbf{R}_0|\mathbf{R}_k]$, using

$$[\mathbf{R}_0|\mathbf{R}_k] \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{S}_{C_k}, \mathbf{T}_{\mathbf{G}}, \mathbf{A}^* - \sum_{i\in[L]-\{k\}} \mathbf{A}_{C_i}\mathbf{R}_i)$$

Output simulated recoding key as $\mathsf{Sim.rk}_{\mathsf{sim}} = \{\mathbf{R}\}_{i\in[L]}$.

The indistinguishability of recoding keys property is proved using the properties of sampling algorithms used in the (simulated) recoding keys generation process as follows:

Lemma 3.13. *The instantiation* Π *of controlled homomorphic recoding scheme satisfies* **indistinguishability of recoding keys** *(c.f. Definition 3.4) with respect to* $\mathcal{E}_{\mathsf{aux}}$*, where* $\mathcal{E}_{\mathsf{aux}}$*is defined in Lemma 3.9.*

Proof. The only difference between generating normal recoding keys through $\mathsf{DerivReKey}$ and simulated recoding keys through $\mathsf{Sim.CHR}_{\mathsf{rk}}$ is

- The normal recoding keys are generated using algorithm $\mathsf{SampleLeft}$ with sk.
- The simulated recoding keys are generated using algorithm $\mathsf{SampleRight}$.

By the property of algorithms $\mathsf{SampleLeft}, \mathsf{SampleRight}$ as stated in Lemma 2.3, their outputs are statistically close to discrete Gaussian distribution. In simulated recoding generation, either the recoding keys are generated using $\mathsf{SampleRight}$ or directly sampled from discrete Gaussian distribution. Therefore, we have $\{\mathsf{rk}_{\mathsf{sim}}\} \stackrel{c}{\approx} \{\mathsf{rk}_{\mathsf{real}}\}$, and thus show our instantiation of controlled homomorphic recoding scheme satisfies indistinguishability of recoding keys. □

Generalization of $\mathsf{Sim.CHR}_{\mathsf{rk}}$: We can generalized the above $\mathsf{Sim.CHR}_{\mathsf{rk}}$ algorithm (used in the ABE construction), by evaluating public keys $\{\mathbf{A}_{C_i}\}$ with respect to the gadget circuit $\mathcal{C}$ before generating the simulated recoding keys. We define the algorithm formally as

$\mathsf{Sim.GenCHR}_{\mathsf{rk}}(\mathbf{Inp})$: On input $\mathbf{Inp}$, consider the following two cases:

- If $\mathbf{Inp}$ is of form $(\mathsf{pk}, \{\mathsf{Sim.pk}_i\}_{i\in[\ell]}, \{C_i\}_{i\in[L]}, \boldsymbol{y}, \mathsf{pk}^*, f)$, the run
$$\mathsf{Sim.CHR}_{\mathsf{rk}}(\mathsf{pk}, \{\mathsf{Sim.pk}_i\}_{i\in[\ell]}, \{C_i\}_{i\in[L]}, \boldsymbol{y}, \mathsf{pk}^*, f)$$
- If $\mathbf{Inp}$ is of form $(\mathsf{pk}, \{\mathsf{Sim.pk}_i\}_{i\in[\ell]}, \{C_i\}_{i\in[L]}, \boldsymbol{y}, \mathsf{pk}^*, f, \mathcal{C})$, then If $f(C_1(\boldsymbol{y}), \ldots, C_L(\boldsymbol{y})) = 0$, output $\perp$. Otherwise, first for $i \in [L]$, evaluate public key as $\mathbf{A}_{C_i} = \mathbf{A}\mathbf{S}_{C_i} - C_i(\boldsymbol{y}) = \mathsf{KeyEval}(\{\mathsf{Sim.pk}_j\}_{j\in[\ell]}, C_i)$. Then evaluate $\{\mathbf{A}_{C_i}\}$ with respect to circuit $\mathcal{C}$ to obtain $\mathbf{A}_{\mathsf{C}} = \mathbf{A}\mathbf{S}_{\mathsf{C}} - z\mathbf{G}$, where $z \neq 0$ since $f(C_1(\boldsymbol{y}), \ldots, C_L(\boldsymbol{y})) = 1$. Sample $[\mathbf{R}_0|\mathbf{R}_1]$, using
$$[\mathbf{R}_0|\mathbf{R}_1] \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{S}_{\mathsf{C}}, \mathbf{T}_{\mathbf{G}}, \mathbf{A}^*, \sigma)$$
Output simulated recoding key $\mathsf{Sim.rk}_{\mathsf{sim}} = [\mathbf{R}_0|\mathbf{R}_1]$.

Similarly, we can argue that the recoding key $\mathsf{rk}_{\mathsf{sim}}$ key produced by the generalized algorithm $\mathsf{Sim.GenCHR}_{\mathsf{rk}}(\mathbf{Inp})$ satisfies indistinguishability of recoding keys as

Corollary 3.14. *The generalized instantiation of controlled homomorphic recoding scheme satisfies* **indistinguishability of recoding keys** *(c.f. Definition 3.4) with respect to* $\mathcal{E}_{\mathsf{aux}}$*, where* $\mathcal{E}_{\mathsf{aux}}$ *is defined in Lemma 3.9.*

The proof of indistinguishability of simulated keys from the generalized algorithm is very similar to the proof of Lemma 3.13, thus we omit it here.

Pseudorandomness of Ciphertexts. We show pseudorandomness of ciphertexts (under simulated public keys or regularly generated public keys) based on the hardness of LWE assumption. We first describe the simulated encryption algorithm $\mathsf{Sim.CHR_{ct}}$:

$\mathsf{Sim.CHR_{ct}}(\mathsf{Sim.pk}, \tau, y, \boldsymbol{s})$: On input the simulated public key $\mathsf{Sim.pk} = \mathbf{AS} - y\mathbf{G}$, the trapdoor $\mathbf{S}$, the attribute y and secret message $\boldsymbol{s}$, the simulated encryption algorithm computes and outputs the simulated ciphertext

$$\mathsf{Sim.ct} = \boldsymbol{s}^\mathsf{T}(\mathbf{AS} - y\mathbf{G} + y\mathbf{G}) + \boldsymbol{e}^\mathsf{T}\mathbf{S}$$

where veceot $\boldsymbol{e} \leftarrow \mathcal{D}_{\mathbb{Z}^m,\sigma}$.

Lemma 3.15. *Assuming the hardness of sub-exponential LWE assumption (c.f. Definition 2.4), the instantiation* Π *of controlled homomorphic recoding scheme satisfies* **pseudorandomness of ciphertexts** *(c.f. Definition 3.5).*

Proof. For regularly generated public keys, the ciphertext is of the form $\boldsymbol{c} = \boldsymbol{s}^\mathsf{T}(\mathbf{A} - y\mathbf{G}) + \boldsymbol{e}^\mathsf{T}$, where matrix $\mathbf{A}$ is chosen randomly from $\mathbb{Z}_q^{n\times m}$, vector $\boldsymbol{s}$ is secret and chosen randomly from $\mathbb{Z}_q^n$ and $\boldsymbol{e} \leftarrow \mathcal{D}_{\mathbb{Z}^m,\sigma}$. By hardness of LWE assumption, we have $\boldsymbol{s}^\mathsf{T}\mathbf{A} + \boldsymbol{e}^\mathsf{T}$ is computationally close to uniformly random distribution over $\mathbb{Z}_q^m$. For simulated public keys, the ciphertext is of the form $\boldsymbol{c} = (\boldsymbol{s}^\mathsf{T}\mathbf{A} + \boldsymbol{e}^\mathsf{T})\mathbf{R}$. By the LWE assumption, we have $\boldsymbol{s}^\mathsf{T}\mathbf{A} + \boldsymbol{e}^\mathsf{T}$ is computationally close to uniform distribution over $\mathbb{Z}_q^m$, and since $\mathbf{R} \leftarrow \mathcal{D}_{\mathbb{Z}^{m\times m}}$ and by Leftover Hash Lemma, we have $(\boldsymbol{s}^\mathsf{T}\mathbf{A} + \boldsymbol{e}^\mathsf{T})\mathbf{R}$ is computationally close to uniformly random distribution over $\mathbb{Z}_q^m$.

Therefore, we prove that the instantiation satisfies the property of pseudorandomness of ciphertexts. □

4 ABE for RAMs from CHR

In this section, we present the construction of ABE for the class of RAM programs $\mathcal{P}$ from controlled homomorphic encoding scheme. Before we present our construction, we first define auxiliary circuits that will be associated with the step circuit of the RAM program.

Auxiliary Circuits. We define auxiliary circuits $(C^{\mathsf{up}}, C^{\mathsf{ck}})$ that will keep track of all the locations that have been written so far along with the most recent time step they were updated. These circuits will be useful to prevent an adversary from using an "illegal" encoding to recode to the next step. For instance, suppose the step circuit outputs the location 112 to be read in the next step. If the 112-th location has been written multiple times then the adversarial evaluator can use an 'old' encoding of the 112-th location (and hence, illegal) in the recoding step. We refer to this issue as repeated writing issue in the technical overview.

Thus, we have $(C^{\mathsf{up}}, C^{\mathsf{ck}})$ to keep track of the updates made. Moreover, the pair of circuits $(C^{\mathsf{up}}, C^{\mathsf{ck}})$ will be combined with the step circuit at the cost of increasing the size as a function of the upper bound T.

We define auxiliary circuits C^{up} and C^{ck} as

Input: a list L, location i, time j.
Computation: Traverse the list L to check whether there is a pair (i, j') where $j' < j$. If yes, replace the pair (i, j') with (i, j) and add $(0, 0)$ to the list. Otherwise, add (i, j) to the list.

Figure 4: Definition of circuit C^{up}

Input: a list L, location i, time j.
Computation: Traverse the list L to check whether there is a pair (i, j). If yes, then return j, otherwise return 0.

Figure 5: Definition of circuit C^{ck}

We assume that, for every program $P \in \mathcal{P}$, the associated step circuit C always takes as input the first location of memory, and the initial state is all 1. This is without loss of generality since every program P can be modified such that this property holds, with the overhead of an extra time step. We list the RAM parameters that will be used in our construction in the following table:

Parameters	Description	Setting
N	maximum database length	$\mathsf{poly}(\lambda)$
T	maximum running time	$\mathsf{poly}(\lambda)$
τ	state bit-length	$\mathsf{poly}(\lambda)$
θ	address bit-length	$\log N$
η	list unit bit-length	$\log N + \log T$
ϕ	C^{ck} circuit output bit-length	$\log T$

Table 2: ABE Parameters

Every RAM program $P \in \mathcal{P}$ is parameterized by running time t and memory length N and represented as a step-circuit C, which is

$$(\mathsf{st}_i, \mathsf{loc}_i^{\mathsf{w}}, b_i^{\mathsf{w}}, \mathsf{loc}_i^{\mathsf{r}}) \leftarrow C(\mathsf{st}_{i-1}, \mathsf{loc}_{i-1}^{\mathsf{r}}, b_{i-1}^{\mathsf{r}})$$

We incorporate the auxiliary circuits described above in the description of every program $P \in \mathcal{P}$. In more detail, we have a different step circuit for every step of the computation. The step-circuit C_j in the j-th step, decomposed into binary representation can be written as follows, i.e.

$$C_j = \left(\{C_i^{\mathsf{st}}\}_{i=1}^{\tau}, \{C_i^{\mathsf{w}}\}_{i=1}^{\theta}, C^{\mathsf{wb}}, \{C_i^{\mathsf{r}}\}_{i=1}^{\theta}, \{C_k^{up}\}_{k=1}^{(j+1)\eta}, \{C_i^{\mathsf{ck}}\}_{i=1}^{\phi}\right)$$

where C_i^{st} outputs the i-th bit of st for $i \in [\tau]$, $(C_i^{\mathsf{w}}, C_i^{\mathsf{r}})$ output the i-th bit of the write/read address respectively for $i \in [\theta]$, and C^{wb} outputs the bit to be written. Since the list maintained by update circuit C^{up} increases by one component for each step, so for j-th step the number of decomposed outputs in C^{up} is $(j+1)\eta$.

Construction. We construct attribute based encryption for RAMs from CHR scheme $\mathsf{CHR} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyEval}, \mathsf{CtEval}, \mathsf{ReEncKG}, \mathsf{ReEnc}, \mathsf{EqTest})$. In our construction below, we define a gadget circuit $\mathcal{C}$ as $\mathcal{C}(x_1, ..., x_\theta) = \sum_{i=1}^{\theta} x_i 2^i$, where $x_i \in \{0,1\}$ for $i \in [\theta]$. In the execution of RAM program, we assume that the initial state is all 1s, the satisfying state is all 0s, and the program always reads the 1st location of database. As mentioned in Table 1, we use different controlled functions in different ABE settings. We denote the ABE scheme to be $\mathsf{ABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.

Notational convention: As explained in the technical overview, we need a layer of public keys for every step of the computation. The 0-th layer of public keys and the target public key (for the last step of computation) is reused across different encryptions. The intermediate layers of public keys, however, are freshly sampled from one execution of key generation to another.

We denote the public keys of i-th step to be $\mathsf{Step}[i].\mathsf{pk}$. The superscript in the notation of the public key denotes the type of the value being encoded. The subscript denotes the index of the binary representation of the value being encoded.

Notations	Encoding
$\mathsf{Step}[j].\mathsf{pk}_i^{\mathsf{st}}$	i-th bit of state in j-th step
$\mathsf{Step}[j].\mathsf{pk}_i^{\mathsf{ra}}$	i-th bit of to-read address for $(j+1)$-th step
$\mathsf{Step}[j].\mathsf{pk}_i^{\mathsf{lt}}$	i-th bit of update list until j-th step
$\mathsf{Step}[j].\mathsf{pk}_i^{\mathsf{db}}$	i-th bit of database in j-th step
$\mathsf{Step}[j].\mathsf{pk}$	reading value in j-th step
$\mathsf{Step}[i].\mathsf{pk}^t$	i-th time step
$\mathsf{pk}_{\mathsf{out}}$	target public key

Table 3: ABE Construction Notations

We use the notation $\mathsf{Step}[j].\mathsf{hompk}$ to correspond to the public key obtained by homomorphically evaluating on j-th layer public keys. The superscripts and subscripts on hompk hold the same meaning as above.

- $\mathsf{ABE.Setup}(1^\lambda)$: On input security parameter λ,
  - ◇ PUBLIC KEYS ASSOCIATED WITH STATE: Generate the 0-th step public keys that are used to encode the initial state. Compute $\mathsf{CHR.Setup}(1^\lambda)$, τ number of times, to obtain $\{(\mathsf{Step}[0].\mathsf{pk}_i^{\mathsf{st}}, \mathsf{Step}[0].\mathsf{sk}_i^{\mathsf{st}})\}_{i\in[\tau]}$.
  - ◇ PUBLIC KEYS ASSOCIATED WITH READ ADDRESS: Generate the 0-th step public keys that is used to encode the initial read address. Compute $\mathsf{CHR.Setup}(1^\lambda)$, θ number of times, to obtain $\{(\mathsf{Step}[0].\mathsf{pk}_j^{\mathsf{ra}}, \mathsf{Step}[0].\mathsf{sk}_j^{\mathsf{ra}})\}_{j\in[\theta]}$.
  - ◇ PUBLIC KEYS ASSOCIATED WITH ADDRESS LIST: Generate the 0-th step public keys that are used to encode the address list, which is initialized with zeroes. During the evaluation process, the address list is populated with the addresses written so far and the most recent time step they were written. Compute $\mathsf{CHR.Setup}(1^\lambda)$, η number of times, to obtain $\{(\mathsf{Step}[0].\mathsf{pk}_k^{\mathsf{lt}}, \mathsf{Step}[0].\mathsf{sk}_k^{\mathsf{lt}})\}_{k\in[\eta]}$. Generate $\mathsf{Step}[0].\mathsf{pk}^t$ to encode the 0-th time step.
  - ◇ PUBLIC KEYS ASSOCIATED WITH DATABASE: Generate the 0-th step public keys that is used to encode the initial attribute database. Compute $\mathsf{CHR.Setup}(1^\lambda)$, N number of times, to obtain $\{(\mathsf{Step}[0].\mathsf{pk}_i^{\mathsf{lt}}, \mathsf{Step}[0].\mathsf{sk}_i^{\mathsf{lt}})\}_{i\in[N]}$.
  - ◇ ANCHOR PUBLIC KEY: Generate a public key-secret key pair $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{CHR.Setup}(1^\lambda)$. The public key pk_0 will participate in every recoding key process (during key generation), in which the secret key sk_0 will be used. We note that the secret keys generated in the above bullets will be discarded and only the public keys will be used for the rest of the construction.

  Output master secret key $\mathsf{msk} = \mathsf{sk}_0$ and public parameter pp as

  $$\mathsf{pp} = (\{\mathsf{Step}[0].\mathsf{pk}_i^{\mathsf{st}}\}_{i\in[\tau]}, \{\mathsf{Step}[0].\mathsf{pk}_j^{\mathsf{ra}}\}_{j\in[\theta]}, \{\mathsf{Step}[0].\mathsf{pk}_k^{\mathsf{lt}}\}_{k\in[\eta]}, \{\mathsf{Step}[0].\mathsf{pk}_i^{\mathsf{db}}\}_{i\in[N]}, \mathsf{Step}[0].\mathsf{pk}^t, \mathsf{pk}_0, \mathsf{pk}_{\mathsf{out}})$$

- $\mathsf{ABE.KeyGen}(\mathsf{msk}, P)$: On input master secret key msk and program P with upper time bound T and database length N, the key generation algorithm parse the step circuit of program P as $(\{C_j^{\mathsf{st}}\}_{j=1}^{\tau}, \{C_i^{\mathsf{w}}\}_{j=1}^{\theta}, C^{\mathsf{wb}}, \{C_j^{\mathsf{r}}\}_{j=1}^{\theta})$ Then generate public keys along for each step as:

◇ Public Keys Associated with Read Value: For $i \in [T]$, generate the public key associated with read value for each step. Compute $\mathsf{CHR.Setup}(1^\lambda)$, T number of times, to obtain $\{(\mathsf{Step}[i].\mathsf{pk}, \mathsf{Step}[i].\mathsf{pk}\}_{i\in[T]}$.

◇ Public keys associated with time step: For $i \in [T]$, generate the public key associated with time step for each step. Compute $\mathsf{CHR.Setup}(1^\lambda)$, T number of times, to obtain $\{(\mathsf{Step}[i].\mathsf{pk}^t, \mathsf{Step}[i].\mathsf{pk}^t\}_{i\in[T]}$.

◇ Public keys associated with state: For $i \in [T]$, generate the i-th public keys that are used to encode the state. Compute $\mathsf{CHR.Setup}(1^\lambda)$, $T\tau$ number of times, to obtain $\{\mathsf{Step}[i].\mathsf{pk}^{\mathsf{st}}_j, \mathsf{Step}[i].\mathsf{sk}^{\mathsf{st}}_j\}_{i\in[T],j\in[\tau]}$.

◇ public keys associated with read address: For $i \in [T]$, generate the i-th public keys that are used to encode the read address for each step. Compute $\mathsf{CHR.Setup}(1^\lambda)$, $T\theta$ number of times, to obtain $\{\mathsf{Step}[i].\mathsf{pk}^{\mathsf{ra}}_j, \mathsf{Step}[i].\mathsf{sk}^{\mathsf{ra}}_j\}_{i\in[T],j\in[\theta]}$.

◇ public keys associated with database: For $i \in [T]$, generate the i-th public keys that are used to encode the database for each step. Compute $\mathsf{CHR.Setup}(1^\lambda)$, TN number of times, to obtain $\{\mathsf{Step}[i].\mathsf{pk}^{\mathsf{db}}_\ell, \mathsf{Step}[i].\mathsf{sk}^{\mathsf{db}}_\ell\}_{i\in[T],\ell\in[N]}$.

◇ public keys associated with address list: For $i \in [T]$, generate the public keys that are used to encode the updated address list. In each step, the list grows by η entries as specified in the Definition of circuit C^{up} (c.f. Figure 4). For $i \in [T]$, compute $\mathsf{CHR.Setup}(1^\lambda)$, $(i+1)\eta$ number of times, to obtain $\{\mathsf{Step}[i].\mathsf{pk}^{\mathsf{lt}}_j, \mathsf{Step}[i].\mathsf{sk}^{\mathsf{lt}}_j\}_{j\in(i+1)\eta}$.

For $i \in [T]$, generate the recoding key for time-step as

$$\mathsf{Step}[i].\mathsf{rk}^t \leftarrow \mathsf{CHR.ReEncKG}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{pk}^t, \mathsf{sk}_0, \mathsf{Step}[i+1].\mathsf{pk}^t, g_i)$$

where control function g_i is defined in Equation (2). Next, for $i \in \{0\} \cup [T]$, do the following:

1. State circuit $\{C^{\mathsf{st}}_j\}_{j\in[\tau]}$: First for $j \in [\tau]$, homomorphically compute public key $\mathsf{Step}[i].\mathsf{hompk}^{\mathsf{st}}_j$ with respect to C^{st}_j, then evaluate the gadget circuit $\mathcal{C}$ on input the homomorphic public keys, and provide a terminating recoding key $\mathsf{Step}[i].\mathsf{rk}^{\mathsf{out}}$ from current i-th step to the output step. Next, provide a recoding key $\mathsf{Step}[i].\mathsf{rk}^{\mathsf{st}}_j$ which recode the state information of i-th step to the $(i+1)$-th step. The detail follows: For $j \in [\tau]$, evaluate C^{st}_j

$$\mathsf{Step}[i].\mathsf{hompk}^{\mathsf{st}}_j \leftarrow \mathsf{CHR.KeyEval}(\{\mathsf{Step}[i].\mathsf{pk}^{\mathsf{st}}_k\}_{k\in[\tau]}, \{\mathsf{Step}[i].\mathsf{pk}^{\mathsf{ra}}_k\}_{k\in[\theta]}, \mathsf{Step}[i].\mathsf{pk}, C^{\mathsf{st}}_j)$$

   And then compute $\mathsf{Step}[i].\mathsf{hompk}^{\mathsf{st}} = \mathsf{CHR.KeyEval}(\{\mathsf{Step}[i].\mathsf{hompk}^{\mathsf{st}}_j\}_{j\in[\tau]}, \mathcal{C})$, and use the secret key sk_0 to compute the recoding key

$$\mathsf{Step}[i].\mathsf{rk}^{\mathsf{out}} \leftarrow \mathsf{CHR.ReEncKG}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{hompk}^{\mathsf{st}}, \mathsf{sk}_0, \mathsf{pk}_{\mathsf{out}}, h)$$

   where control function h is defined in Equation (2). Next generate the recoding key as

$$\mathsf{Step}[i].\mathsf{rk}^{\mathsf{st}}_j \leftarrow \mathsf{CHR.ReEncKG}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{hompk}^{\mathsf{st}}_j, \mathsf{sk}_0, \mathsf{Step}[i+1].\mathsf{pk}^{\mathsf{st}}_j, \mathsf{Ind})$$

2. Reading address circuit $\{C^{\mathsf{r}}_j\}_{j\in[\theta]}$: First for $j \in [\theta]$, homomorphically compute public key $\mathsf{Step}[i].\mathsf{hompk}^{\mathsf{ra}}_j$ with respect to C^{r}_j, then provide a recoding key $\mathsf{Step}[i].\mathsf{rk}^{\mathsf{ra}}_j$ which recode the read address information of i-th step to the $(i+1)$-th step. Next, evaluate the gadget circuit $\mathcal{C}$ on input the homomorphic public keys $\{\mathsf{Step}[i].\mathsf{hompk}^{\mathsf{ra}}_j\}$ to obtain $\mathsf{Step}[i].\mathsf{pk}^{\mathsf{ra}}$, and evaluate the check circuit C^{ck} on address list $\{\mathsf{Step}[i].\mathsf{pk}^{\mathsf{lt}}_k\}_{k\in[i\eta]}$ and $\mathsf{Step}[i].\mathsf{pk}^{\mathsf{ra}}$. Provide recoding keys,

which recode the specific database location to read value, according to read address $\mathsf{Step}[i].\mathsf{pk}^{\mathsf{ra}}$ and result $\mathsf{Step}[i].\mathsf{pk}^{\mathsf{ck}}$ of C^{ck}. The detail follows: For $j \in [\theta]$, evaluate C^{r}_j as

$$\mathsf{Step}[i].\mathsf{hompk}^{\mathsf{ra}}_j \leftarrow \mathsf{CHR.KeyEval}(\{\mathsf{Step}[i].\mathsf{pk}^{\mathsf{st}}_k\}_{k\in[\tau]}, \{\mathsf{Step}[i].\mathsf{pk}^{\mathsf{ra}}_k\}_{k\in[\theta]}, \mathsf{Step}[i].\mathsf{pk}, C^{\mathsf{r}}_j)$$

Then compute the following

$$\mathsf{Step}[i].\mathsf{rk}^{\mathsf{ra}}_j \leftarrow \mathsf{CHR.ReEncKG}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{hompk}^{\mathsf{ra}}_j, \mathsf{sk}_0, \mathsf{Step}[i+1].\mathsf{pk}^{\mathsf{ra}}_k, \mathsf{Ind})$$

Then evaluate gadget circuit $\mathcal{C}$ and the check circuit C^{ck} (c.f. Figure 5) as

$$\mathsf{Step}[i].\mathsf{pk}^{\mathsf{ra}} = \mathsf{CHR.KeyEval}(\{\mathsf{Step}[i].\mathsf{hompk}^{\mathsf{ra}}_j\}_{j\in[\theta]}, \mathcal{C})$$

$$\mathsf{Step}[i].\mathsf{pk}^{\mathsf{ck}} \leftarrow \mathsf{CHR.KeyEval}(\mathsf{Step}[i].\mathsf{pk}^{\mathsf{r}}, \{\mathsf{Step}[i].\mathsf{pk}^{\mathsf{lt}}_k\}_{k\in[i\eta]}, \mathsf{pk}^t_i, C^{\mathsf{ck}})$$

Next, for $k \in [N], \ell \in [i-1]$, compute the following

$$\mathsf{Step}[i].\mathsf{rk}^{\mathsf{r}}_{k\ell} \leftarrow \mathsf{CHR.ReEncKG}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{pk}^{\mathsf{ra}}, \mathsf{Step}[i].\mathsf{pk}^{\mathsf{ck}}, \mathsf{Step}[\ell].\mathsf{pk}^{\mathsf{db}}_k, \mathsf{sk}_0, \mathsf{Step}[i+1].\mathsf{pk}, f_{k\ell})$$

where $\{\mathsf{Step}[\ell].\mathsf{pk}^{\mathsf{db}}_k\}_{k\in[N],\ell\in[i-1]}$ are freshly generated public keys in Writing address/value part as described below, and control function $f_{k\ell}$ is defined in Equation (1).

3. Writing address/value circuits ($\{C^{\mathsf{w}}_j\}_{j\in[\theta]}, C^{\mathsf{wb}}$): First for $j \in [\theta]$, homomorphically compute public key $\mathsf{Step}[i].\mathsf{hompk}^{\mathsf{w}}_j$ with respect to C^{w}_j and $\mathsf{Step}[i].\mathsf{pk}^{\mathsf{wb}}$ with respect to C^{wb}). Then, evaluate the gadget circuit $\mathcal{C}$ on input the homomorphic public keys $\{\mathsf{Step}[i].\mathsf{hompk}^{\mathsf{w}}_j\}$ to obtain $\mathsf{Step}[i].\mathsf{pk}^{\mathsf{w}}$. Next, for each entry of the database, provide a recoding key, which recodes the writing value to the freshly generated database public key $\mathsf{Step}[i+1].\mathsf{pk}^{\mathsf{db}}_\ell$. The detail follows: For $j \in [\theta]$, evaluate C^{w}_j and C^{wb} as

$$\mathsf{Step}[i].\mathsf{hompk}^{\mathsf{w}}_j \leftarrow \mathsf{CHR.KeyEval}(\{\mathsf{Step}[i].\mathsf{pk}^{\mathsf{st}}_k\}_{k\in[\tau]}, \{\mathsf{Step}[i].\mathsf{pk}^{\mathsf{ra}}_k\}_{k\in[\theta]}, \mathsf{Step}[i].\mathsf{pk}, C^{\mathsf{w}}_j)$$

$$\mathsf{Step}[i].\mathsf{pk}^{\mathsf{wb}} \leftarrow \mathsf{CHR.KeyEval}(\{\mathsf{Step}[i].\mathsf{pk}^{\mathsf{st}}_k\}_{k\in[\tau]}, \{\mathsf{Step}[i].\mathsf{pk}^{\mathsf{ra}}_k\}_{k\in[\theta]}, \mathsf{Step}[i].\mathsf{pk}, C^{\mathsf{wb}})$$

Then compute $\mathsf{Step}[i].\mathsf{pk}^{\mathsf{w}} = \mathsf{CHR.KeyEval}(\{\mathsf{Step}[i].\mathsf{hompk}_j\}_{j\in[\theta]}, \mathcal{C})$, where circuit $\mathcal{C}$ is the gadget circuit. Then do following computation for $\ell \in [N]$

$$\mathsf{Step}[i].\mathsf{rk}^{\mathsf{w}}_\ell \leftarrow \mathsf{CHR.ReEncKG}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{pk}^{\mathsf{w}}, \mathsf{Step}[i].\mathsf{pk}^{\mathsf{wb}}, \mathsf{sk}_0, \mathsf{Step}[i+1].\mathsf{pk}^{\mathsf{db}}_\ell, f_\ell)$$

where control function f_ℓ is defined in Equation (1).

4. Update circuit $\{C^{\mathsf{up}}_j\}_{j\in[(i+1)\eta]}$: First for $j \in [(i+1)\eta]$, homomorphically compute public key $\mathsf{Step}[i].\mathsf{hompk}^{\mathsf{lt}}_j$ with respect to C^{up}_j. Then, provide a recoding key $\mathsf{Step}[i].\mathsf{rk}^{\mathsf{lt}}_j$ which recode the address list information of i-th step to the $(i+1)$-th step. The detail follows: For $j \in [(i+1)\eta]$, evaluate the update circuit C^{up} (c.f. Figure 4) as

$$\mathsf{Step}[i].\mathsf{hompk}^{\mathsf{lt}}_j \leftarrow \mathsf{CHR.KeyEval}(\{\mathsf{Step}[i].\mathsf{pk}^{\mathsf{lt}}_j\}_{j\in[i\eta]}, \mathsf{Step}[i].\mathsf{pk}^{\mathsf{w}}, \mathsf{pk}^t_i, C^{\mathsf{up}}_j)$$

Then use the secret key sk_0 to generate recoding key as

$$\mathsf{Step}[i].\mathsf{rk}^{\mathsf{lt}}_j \leftarrow \mathsf{CHR.ReEncKG}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{hompk}^{\mathsf{lt}}_j, \mathsf{sk}_0, \mathsf{Step}[i+1].\mathsf{pk}^{\mathsf{lt}}_j, \mathsf{Ind})$$

Output the secret key for RAM program P as $\mathsf{sk}_P = (P, \{\mathsf{Step}[i].\mathsf{KEY}\}_{i=0}^{T-1},)$ where

$$\mathsf{Step}[i].\mathsf{KEY} = (\mathsf{Step}[i].\mathsf{rk}^{\mathsf{out}}, \mathsf{Step}[i].\mathsf{rk}^t, \{\mathsf{Step}[i].\mathsf{rk}^{\mathsf{st}}_j\}_{j\in[\tau]}, \{\mathsf{Step}[i].\mathsf{rk}^{\mathsf{r}}_j\}_{j\in[\theta]}, \{\mathsf{Step}[i].\mathsf{rk}^{\mathsf{ra}}_{kj}\}_{k\in[i-1],j\in[N]}, \{\mathsf{Step}[i].\mathsf{rk}^{\mathsf{w}}_j\}_{j\in[\theta]}, \{\mathsf{Step}[i].\mathsf{rk}^{\mathsf{lt}}_j\}_{j\in[(i+1)\eta]})$$

- $\mathsf{ABE.Enc}(\mathsf{pp}, D, \mu)$: On input the public parameter pp, a database $D = \{x_i\}_{i=1}^N$ and a message μ, the encryption algorithm first picks a secret message s uniformly at random from $\mathcal{S}$ and compute the following ciphertexts:

  ◇ Database Encryption: For $i \in [N]$, generate ciphertexts for each entry of the database. Compute $\mathsf{Step}[0].\mathsf{ct}_i^{\mathsf{db}} = \mathsf{CHR.Enc}(\mathsf{Step}[0].\mathsf{pk}_i^{\mathsf{db}}, x_i, s)$, for $i \in [N]$.

  ◇ Initial state encryption For $i \in [\tau]$, generate ciphertexts for initial state. Compute $\mathsf{Step}[0].\mathsf{ct}_i^{\mathsf{st}} = \mathsf{CHR.Enc}(\mathsf{Step}[0].\mathsf{pk}_i^{\mathsf{st}}, 1, s)$, for $i \in [\tau]$.

  ◇ Initial Reading address Encryption: For $i \in [\theta]$, generate ciphertexts for initial reading address. Compute $\mathsf{Step}[0].\mathsf{ct}_1^{\mathsf{ra}} = \mathsf{CHR.Enc}(\mathsf{Step}[0].\mathsf{pk}_1^{\mathsf{ra}}, 1, s)$, and for $i = 2, \ldots, \theta$ compute $\mathsf{Step}[0].\mathsf{ct}_i^{\mathsf{ra}} = \mathsf{CHR.Enc}(\mathsf{Step}[0].\mathsf{pk}_i^{\mathsf{ra}}, 0, s)$.

  ◇ Initial address list encryption: For $i \in [\eta]$, generate ciphertexts for initial address list. Compute $\mathsf{Step}[0].\mathsf{ct}_i^{\mathsf{lt}} = \mathsf{CHR.Enc}(\mathsf{Step}[0].\mathsf{pk}_i^{\mathsf{lt}}, 0, s)$, for $i \in [\eta]$.

  ◇ Auxiliary information encryption: Encrypt under the anchor public key $c_0 = \mathsf{CHR.Enc}(\mathsf{pk}_0, 0, s)$ and 0-th time step public key $u_0 = \mathsf{CHR.Enc}(\mathsf{Step}[0].\mathsf{pk}^t, 0, s)$.

  ◇ message encryption: If the message $\mu = 0$, then compute $\psi = \mathsf{Enc}(\mathsf{pk}_{\mathsf{out}}, 0, s)$. Otherwise, if the message $\mu = 1$, choose a random vector over the ciphertext space of CHR.

  Output the ciphertext as

  $$\mathsf{ct}_D = (D, c_0, u_1, \{\mathsf{Step}[0].\mathsf{ct}_i^{\mathsf{st}}\}_{i=1}^{\tau}, \{\mathsf{Step}[0].\mathsf{ct}_i^{\mathsf{db}}\}_{i=1}^{N}, \{\mathsf{Step}[0].\mathsf{ct}_i^{\mathsf{ra}}\}_{i=1}^{\theta}, \{\mathsf{Step}[0].\mathsf{ct}_i^{\mathsf{lt}}\}_{i=1}^{\eta}, \psi)$$

- $\mathsf{ABE.Dec}(\mathsf{sk}_P, \mathsf{ct}_D)$: On input secret key sk_P for RAM program P and a ciphertext ct_D, outputs $\perp$ if $P^D \neq 0$. Otherwise, parse $\mathsf{sk}_P = (P, \{\mathsf{Step}[i].\mathsf{KEY}\}_{i=0}^{T-1})$. Parse the step circuit C_i of the RAM program as

  $$C_i = (\{C_j^{\mathsf{st}}\}_{j=1}^{\tau}, \{C_i^{\mathsf{w}}\}_{j=1}^{\theta}, C^{\mathsf{wb}}, \{C_j^{\mathsf{r}}\}_{j=1}^{\theta}, \{C_j^{\mathsf{up}}\}_{j\in[(i+1)\eta]})$$

  1. State circuit $\{C_j^{\mathsf{st}}\}_{j\in[\tau]}$: First, for $j \in [\tau]$, homomorphically compute ciphertext $\mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{st}}$ with respect to C_j^{st}. If at the current step, the state is all 0, the use the terminating key $\mathsf{Step}[i].\mathsf{rk}^{\mathsf{out}}$) to recode the i-th step to the output step, and execute the last step algorithm. Otherwise, use the recoding key $\mathsf{Step}[i].\mathsf{rk}_j^{\mathsf{st}}$ to recode the state information of i-th step to $(i+1)$-th step. The detail follows: For $j \in [\tau]$, evaluate C_j^{st} as

     $$\mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{st}} \leftarrow \mathsf{CHR.CtEval}(\{\mathsf{Step}[i].\mathsf{ct}_k^{\mathsf{st}}\}_{k\in[\tau]}, \{\mathsf{Step}[i].\mathsf{ct}_k^{\mathsf{ra}}\}_{k\in[\theta]}, \mathsf{Step}[i].\mathsf{ct}, C_j^{\mathsf{st}})$$

     If at the current step, the state is all 0, then evaluate the following function $\mathsf{Step}[i].\mathsf{ct}^{\mathsf{st}} = \mathsf{CHR.CtEval}(\{\mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{st}}\}_{j\in[\tau]}, \mathcal{C})$, compute

     $$\mathsf{Step}[i].\mathsf{ct}^{\mathsf{out}} = \mathsf{CHR.ReEnc}(c_0, \mathsf{Step}[i].\mathsf{ct}^{\mathsf{st}}, \mathsf{Step}[i].\mathsf{rk}^{\mathsf{out}})^8$$

     and, jump to the last step. Otherwise, for $j \in [\tau]$, compute

     $$\mathsf{Step}[i+1].\mathsf{ct}_j^{\mathsf{st}} = \mathsf{CHR.ReEnc}(c_0, \mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{st}}, \mathsf{Step}[i].\mathsf{rk}_j^{\mathsf{st}})$$

[8] For ease of notation, we omit the public keys in the input to algorithm $\mathsf{CHR.ReEnc}$ when the context is clear.

2. Reading address circuit $\{C_j^{\mathsf{r}}\}_{j\in[\theta]}$: First, for $j \in [\theta]$, homomorphically compute ciphertext $\mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{ra}}$ with respect to C_j^{r}. Then use the recoding key $\mathsf{Step}[i].\mathsf{rk}_j^{\mathsf{ra}}$ to recode the read address information of i-th step to $(i+1)$-th step. Next, homomorphically evaluate the gadget circuit $\mathcal{C}$ and check circuit C^{ck} to obtain $\mathsf{Step}[i].\mathsf{ct}^{\mathsf{ra}}$ and $\mathsf{Step}[i].\mathsf{ct}^{\mathsf{ck}}$ respectively. Recode the value residing in correct location of database using $\mathsf{Step}[i].\mathsf{rk}_{k\ell}^{\mathsf{r}}$, where k, ℓ can be determined by the execution of P^D. The detail follows: For $j \in [\theta]$, evaluate C_j^{w} as

$$\mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{ra}} \leftarrow \mathsf{CHR.CtEval}(\{\mathsf{Step}[i].\mathsf{ct}_k^{\mathsf{st}}\}_{k\in[\tau]}, \{\mathsf{Step}[i].\mathsf{ct}_k^{\mathsf{ra}}\}_{k\in[\theta]}, \mathsf{Step}[i].\mathsf{ct}, C_j^{\mathsf{r}})$$

Then for $j \in [\theta]$, compute $\mathsf{Step}[i+1].\mathsf{ct}_j^{\mathsf{ra}} = \mathsf{CHR.ReEnc}(c_0, \mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{r}}, \mathsf{Step}[i].\mathsf{rk}_j^{\mathsf{ra}})$. Next evaluate gadget circuit $\mathcal{C}$ and C^{ck} (c.f. Figure 5)

$$\mathsf{Step}[i].\mathsf{ct}^{\mathsf{ra}} = \mathsf{CHR.CtEval}(\{\mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{r}}\}_{j\in[\theta]}, \mathcal{C})$$

$$\mathsf{Step}[i].\mathsf{ct}^{\mathsf{ck}} = \mathsf{CHR.CtEval}(\mathsf{Step}[i].\mathsf{ct}^{\mathsf{r}}, \{\mathsf{Step}[i].\mathsf{ct}_j^{\mathsf{lt}}\}_{j\in[i\eta]}, u_i, C^{\mathsf{ck}})$$

where $\mathsf{Step}[i].\mathsf{ct}^{\mathsf{ra}}$ is encoding of read address j and $\mathsf{Step}[i].\mathsf{ct}^{\mathsf{up}}$ is encoding of last written time k, and u_i is the encoding of time-step i that can be obtained by computing $u_i = \mathsf{CHR.ReEnc}(c_0, u_{i-1}, \mathsf{Step}[i].\mathsf{rk}^t)$. We can determine $k \in [N]$ and $\ell \in [i-1]$ by executing P on the database D. And choose the corresponding re-key $\mathsf{Step}[i].\mathsf{rk}_{k\ell}^{\mathsf{r}}$, then compute

$$\mathsf{Step}[i+1].\mathsf{ct}^{\mathsf{r}} = \mathsf{CHR.ReEnc}(c_0, \mathsf{Step}[i].\mathsf{ct}^{\mathsf{r}}, \mathsf{Step}[i].\mathsf{ct}^{\mathsf{ck}}, \mathsf{Step}[\ell].\mathsf{ct}_k^{\mathsf{db}}, \mathsf{Step}[i].\mathsf{rk}_{k\ell}^{\mathsf{r}})$$

3. Writing address/value circuits $(\{C_j^{\mathsf{w}}\}_{j\in[\theta]}, C^{\mathsf{wb}})$: First, for $j \in [\theta]$, homomorphically compute ciphertext $\mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{w}}$ with respect to C_j^{w} and compute $\mathsf{Step}[i].\mathsf{ct}^{\mathsf{wb}}$ with respect to C^{wb}. Then, evaluate the gadget circuit $\mathcal{C}$ on input ciphertexts $\{\mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{w}}\}$ to obtain $\mathsf{Step}[i].\mathsf{ct}^{\mathsf{w}}$. Next, recode the writing value along with the writing address to $\mathsf{ct}_k^{\mathsf{db}}$, using recoding key $\mathsf{Step}[i].\mathsf{rk}_k^{\mathsf{w}}$, where k can be determined by the execution of P^D. The detail follows: For $j \in [\theta]$, evaluate C_i^{w} and C^{wb} as

$$\mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{w}} \leftarrow \mathsf{CHR.CtEval}(\{\mathsf{Step}[i].\mathsf{ct}_k^{\mathsf{st}}\}_{k\in[\tau]}, \{\mathsf{Step}[i].\mathsf{ct}_k^{\mathsf{ra}}\}_{k\in[\theta]}, \mathsf{Step}[i].\mathsf{ct}, C_j^{\mathsf{w}})$$

$$\mathsf{Step}[i].\mathsf{ct}^{\mathsf{wb}} \leftarrow \mathsf{CHR.CtEval}(\{\mathsf{Step}[i].\mathsf{ct}_k^{\mathsf{st}}\}_{k\in[\tau]}, \{\mathsf{Step}[i].\mathsf{ct}_k^{\mathsf{ra}}\}_{k\in[\theta]}, \mathsf{Step}[i].\mathsf{ct}, C^{\mathsf{wb}})$$

Then evaluate $\mathsf{Step}[i].\mathsf{ct}^{\mathsf{w}} = \mathsf{CHR.CtEval}(\{\mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{w}}\}_{j\in[\theta]}, \mathcal{C})$. Next, pick the corresponding recoding key $\mathsf{Step}[i].\mathsf{rk}_k^{\mathsf{w}}$, where k is the writing address, and compute

$$\mathsf{ct}_k^{\mathsf{db}} = \mathsf{CHR.ReEnc}(c_0, \mathsf{Step}[i].\mathsf{ct}^{\mathsf{w}}, \mathsf{Step}[i].\mathsf{ct}^{\mathsf{wb}}, \mathsf{Step}[i].\mathsf{rk}_k^{\mathsf{w}})$$

4. Update circuit $\{C_j^{\mathsf{up}}\}_{j\in[(i+1)\eta]}$: First, for $j \in [(i+1)\eta]$, homomorphically compute ciphertext $\mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{lt}}$ with respect to C_j^{up}. Then use the recoding key $\mathsf{Step}[i].\mathsf{rk}_j^{\mathsf{lt}}$ to recode the address list information of i-th step to $(i+1)$-th step. The detail follows: For $j \in [i+1]\eta$, evaluate the update circuit C_j^{up} (c.f. Figure 4) as

$$\mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{lt}} \leftarrow \mathsf{CHR.CtEval}(\{\mathsf{Step}[i].\mathsf{ct}_\ell^{\mathsf{lt}}\}_{\ell\in[i\eta]}, \mathsf{Step}[i].\mathsf{ct}^{\mathsf{w}}, u_i, C_j^{\mathsf{up}})$$

Then for $j \in [(i+1)\eta]$, compute

$$\mathsf{Step}[i+1].\mathsf{ct}_j^{\mathsf{lt}} = \mathsf{CHR.ReEnc}(c_0, \mathsf{Step}[i].\mathsf{rk}_j^{\mathsf{lt}}, \mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{lt}})$$

The algorithm of last step is to compute $\mathsf{CHR.EqTest}(\mathsf{pk}_{\mathsf{out}}, \mathsf{Step}[t].\mathsf{ct}^{\mathsf{out}}, \psi)$, where t denotes the time step when P^D halts. If output of $\mathsf{CHR.EqTest}$ is equal, then output 0; otherwise output 1.

We show that the above scheme is a secure ABE for RAMs scheme. In particular, we prove the following theorem.

Theorem 4.1. *Assuming* CHR *for a class of controlled functions* $\mathcal{F}$ *(as defined in Equation (3)),* ABE *(described above) is a secure ABE for RAMs scheme.*

4.1 Correctness and Efficiency Analysis

We now show that the above ABE scheme satisfies the properties of correctness and desired efficiency.

Correctness. We show the correctness proof below.

Lemma 4.2. *Assuming the correctness of CHR for the set of controlled functions $\mathcal{F}$ as defined in Equation (3), the above ABE construction satisfies correctness as defined in Definition 2.1.*

Proof. We first define a notion, named i-th step temporary key

$$\mathsf{Step}[i].\mathsf{TempKey} = (\{\mathsf{Step}[i].\mathsf{pk}_k^{\mathsf{st}}\}_{k\in[\tau]}, \{\mathsf{Step}[i].\mathsf{pk}_k^{\mathsf{ra}}\}_{k\in[\theta]}, \mathsf{Step}[i].\mathsf{pk})$$

Let the ciphertext be ct_D and secret key be sk_P. At i-th step, by evaluating the ciphertext with respect to circuits $\{C_j^{\mathsf{st}}\}_{j\in[\tau]}, \{C_j^{\mathsf{r}}\}_{j\in[\theta]}, \{C_j^{\mathsf{w}}\}_{j\in[\theta]}, C^{\mathsf{wb}}$ and auxiliary circuits $C^{\mathsf{ck}}, \{C_j^{\mathsf{up}}\}_{j\in[(i+1)\eta]}$, we obtain the homomorphic ciphertexts

$$\{\mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{st}}\}_{j\in[\tau]}, \{\mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{ra}}, \mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{w}}\}_{j\in[\theta]}\mathsf{Step}[i].\mathsf{ct}^{\mathsf{wb}}, \{\mathsf{Step}[i].\mathsf{homct}_j^{\mathsf{lt}}\}_{j\in[(i+1)\eta]}$$

By the correctness of CHR scheme (c.f. Definition 3.1), we have that ciphertexts $\mathsf{Step}[i+1].\mathsf{ct}_j^{\mathsf{st}}, \mathsf{Step}[i+1].\mathsf{ct}_j^{\mathsf{r}}, \mathsf{Step}[i+1].\mathsf{ct}_j^{\mathsf{lt}}$ encrypt the same message under $\mathsf{Step}[i+1].\mathsf{TempKey}$ as their i-th step homomorphically evaluated ciphertexts respectively. Since the evaluation of RAM program P on database D is in the clear, so in the reading part, we choose the correct recoding key $\mathsf{Step}[i].\mathsf{rk}_{k\ell}^{\mathsf{r}}$ to recode the value in the k-th location to $\mathsf{Step}[i+1].\mathsf{ct}$.

Suppose at step t, where $t \le T$, we have $P^D = 0$, then evaluate

$$\mathsf{Step}[t].\mathsf{ct}^{\mathsf{st}} = \mathsf{CHR.CtEval}(\{\mathsf{Step}[t].\mathsf{homct}_j^{\mathsf{st}}\}_{j\in[\tau]}, \mathcal{C})$$

where $\mathsf{Step}[t].\mathsf{ct}^{\mathsf{st}}$ encrypts 0. Again by correctness of CHR scheme, the re-encrypted ciphertext $\mathsf{Step}[i].\mathsf{ct}^{\mathsf{out}}$ is also an encryption of 0. At last, the equality test $\mathsf{CHR.EqTest}(\mathsf{pk}_{\mathsf{out}}, \mathsf{Step}[t].\mathsf{ct}^{\mathsf{out}}, \psi)$ outputs correct μ by the property of algorithm $\mathsf{CHR.EqTest}$. □

Remark 4.3. *As mentioned in Remark 2.1, our ABE construction can support auxiliary input y, i.e. $P^D(y)$, where this additional input y serves as initial input of step circuit. The only change in the current construction is that in the encryption algorithm, we encode y as $\mathsf{ct}^{\mathsf{aux}} = \mathsf{CHR.Enc}(\mathsf{pk}^{\mathsf{aux}}, y, s)$, where $\mathsf{pk}^{\mathsf{aux}}$ denotes public key for auxiliary input. The correctness and security proofs closely follow the current ones.*

Efficiency Analysis. We would like to show the following: if a program P on input database D takes time at most T then correspondingly, the decryption of secret key for P on input an encryption of message x associated with attribute database D takes time $p(\lambda, T)$, for a fixed polynomial p, independent of the input length.

We analyze the time to decrypt an encryption of database D associated with message x using a key of RAM program with runtime bounded by T: observe that in the description of $\mathsf{ABE.Dec}$, bullets 1,2,3 and 4 are executed T number of steps. We focus on bounding the running time of bulets 1,2,3 and 4 in any given step. We analyze all four cases below.

- **State circuit**: The runtime of $\mathsf{CHR.CtEval}$ is a polynomial in $(\lambda, \tau, \theta, T\eta)$. Observe that τ is the length of the state, which is independent of the input length, and $\theta = \log N, \eta = \log T + \log N$. Thus, the runtime of $\mathsf{CHR.CtEval}$ is upper bounded by a polynomial in (λ, T). The runtime of $\mathsf{CHR.ReEnc}$ is bounded by a polynomial in (λ, τ).

- **Reading address circuit**: In this step, CHR.CtEval is executed twice. The runtime of first execution of CHR.CtEval is a polynomial in (λ, θ). The runtime of CHR.CtEval is upper bounded by $(\lambda, T\eta)$. Determining j and k takes time at most T. The runtime of CHR.ReEnc is bounded by a polynomial in λ.

- **Writing address/value circuits**: In this step, CHR.CtEval is executed twice. In both executions, the runtime is bounded by a polynomial in $(\lambda, \tau, \theta, T\eta)$. The runtime of CHR.ReEnc is bounded by a polynomial in λ.

- **Update circuit**: In this step, CHR.CtEval is bounded by a polynomial in $(\lambda, T\eta)$. In this step, the runtime of CHR.ReEnc is upper bounded by $(\lambda, T\eta)$. The runtime of CHR.ReEnc is upper bounded by (λ).

From the above observations, it follows that the runtime of the decryption algorithm is a polynomial in (λ, T), where the polynomial is independent of the length of the database.

In particular, notice that if T is polylogarithmic in the input length then the decryption time is sub-linear in the input length.

4.2 Security Proof

We prove the security of our ABE construction based on security of controlled homomorphic recoding scheme. Before proceeding to the proof, we describe some auxiliary algorithms that are useful to the proof. There are five algorithms:

- Sim.ABESetup produces "programmed" public keys. That is, every public key produced as part of setup has hardwired in it, a bit of the challenge database. To perform this operation, we invoke the indistinguishability of setup security of CHR (Definition 3.2).

- Sim.StepKey takes as input the i-th layer of simulated public keys (called temporary keys below) and produces the i-th layer of simulated recoding keys and $(i+1)$-th layer of simulated public keys, except for terminating keys $\mathsf{Step}[i].\mathsf{rk}^{\mathsf{out}}$, which is used for recoding from current step to final step is the program terminates. To perform this operation, we invoke the indistinguishability of simulated keys (Definition 3.3).

- Sim.OutKey takes as input the i-th layer of simulated public keys and produces the simulated terminating keys. We use that fact $P_i^{D^*} \neq 0$ for any queried program P_i to simulate the terminating keys $\mathsf{Step}[i].\mathsf{Simrk}^{\mathsf{out}}$. To perform this operation, we invoke the indistinguishability of recoding keys (Definition 3.4).

- Real.StepKey on input the i-th layer of simulated public keys (called temporary keys) and master secret key of CHR, it produces the i-th layer of 'real' recoding keys and $(i+1)$-th layer of 'real' public keys.

- Sim.Enc produces a simulated encryption of the message. To perform this, we invoke the pseudorandomness of ciphertext property (Definition 3.5).

Proof Intuition. We explain the intuition of the proof next. For explaining the intuition, we focus on weak selective security, where the adversary submits all the queries in the very beginning of the security experiment. The adversary $\mathcal{A}$ submits the database D^*, secret message μ and program queries $P_1, \ldots, P_Q$ such that $P_i^{D^*} \neq 0$.

Hybrid H_1 corresponds to the real experiment, where all the parameters are sampled according to CHR. First, the challenger simulates the public keys of ABE using the algorithm Sim.ABESetup on input the database D^* (hybrid H_2). Also, in the same hybrid, generate all the layers of the recoding keys in every attribute key using Real.StepKey. The challenger, over a sequence of hybrids ($\mathsf{H}_{3,1,\star}$), starts manipulating the attribute key of P_1. Recall that the attribute key of P_1 consists of T sets of recoding keys, the step-keys. Next, switch every intermediate layer of recoding layers to be simulated. That is, in the j-th step ($\mathsf{H}_{3,1,j}$), all the $(j-1)$ layers of recoding keys are simulated using Sim.StepKey while all the layer from $(j+1)$-th onwards are computed using Real.StepKey. Switch the j-th layer of recoding keys to be simulated using Sim.StepKey. At the end of this sequence of hybrids, all the layers of recoding keys in the attribute key of P_1 are simulated. Perform this sequence of hybrids to the rest of the attribute keys associated with $P_2, \ldots, P_Q$. Once this is done, simulate the ciphertext of μ using Sim.Enc.

We present a formal descriptions of the above algorithms.

- $\mathsf{Sim.ABESetup}(1^\lambda, D^*)$: On input security parameter λ and challenge database $D^* = \{x_i^*\}_{i=1}^N$, the simulated setup algorithm first generate the anchor public key pk_0 along with sk_0 by running $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{CHR.Setup}(1^\lambda)$ and simulate public key for 0-th time step pk_0^t and final public key $\mathsf{pk}_{\mathsf{out}}$ as
$$(\mathsf{Simpk}_0^t, \mathsf{Simpk}_{\mathsf{out}}) \leftarrow \mathsf{Sim.GenCHRSetup}(1^\lambda, 0, 2; \mathsf{pk}_0)$$
Then simulate the rest of public parameters as
  1. Simulate public keys $\mathsf{Step}[0].\mathsf{Simpk}_i^{\mathsf{st}}$ for initial state as
  $$\{\mathsf{Step}[0].\mathsf{Simpk}_i^{\mathsf{st}}\}_{i\in[\tau]} \leftarrow \mathsf{Sim.GenCHRSetup}(1^\lambda, 1, \tau; \mathsf{pk}_0)$$
  2. Embed the challenge attribute database D^* into $\mathsf{Step}[0].\mathsf{Simpk}_i^{\mathsf{db}}$ as
  $$\{\mathsf{Step}[0].\mathsf{Simpk}_i^{\mathsf{db}}\}_{i\in[N]} \leftarrow \mathsf{Sim.GenCHRSetup}(1^\lambda, D^*, N; \mathsf{pk}_0)$$
  3. Simulate public keys $\mathsf{Step}[0].\mathsf{Simpk}_i^{\mathsf{r}}$ for read address as $\mathsf{Step}[0].\mathsf{Simpk}_1^{\mathsf{r}} \leftarrow \mathsf{Sim.GenCHRSetup}(1^\lambda, 1, 1; \mathsf{pk}_0)$
  $$\{\mathsf{Step}[0].\mathsf{Simpk}_i^{\mathsf{r}}\}_{i=2}^{\theta} \leftarrow \mathsf{Sim.GenCHRSetup}(1^\lambda, 0, \theta-1; \mathsf{pk}_0)$$
  4. Simulate public keys $\mathsf{Step}[0].\mathsf{Simpk}_i^{\mathsf{lt}}$ for address list as
  $$\{\mathsf{Step}[0].\mathsf{Simpk}_i^{\mathsf{lt}}\}_{i\in[\eta]} \leftarrow \mathsf{Sim.GenCHRSetup}(1^\lambda, 0^\eta, \eta; \mathsf{pk}_0)$$

  Output master secret key $\mathsf{msk} = \mathsf{sk}_0$ and simulated public parameter Sim.pp as
$$\begin{aligned}\mathsf{Sim.pp} = (\{\mathsf{Step}[0].\mathsf{Simpk}_i^{\mathsf{st}}\}_{i\in[\tau]}, &\{\mathsf{Step}[0].\mathsf{Simpk}_j^{\mathsf{r}}\}_{j\in[\theta]}, \{\mathsf{Step}[0].\mathsf{Simpk}_k^{\mathsf{lt}}\}_{k\in[\eta]},\\ &\{\mathsf{Step}[0].\mathsf{Simpk}_i^{\mathsf{db}}\}_{i\in[N]}, \mathsf{Simpk}_0^t, \mathsf{pk}_0, \mathsf{Simpk}_{\mathsf{out}})\end{aligned}$$

- $\mathsf{Sim.StepKey}(\mathsf{Sim.pp}, \mathsf{msk}, \{\mathsf{Step}[j].\mathsf{SimTempKey}\}_{j\in[i]}, P^{D^*})$: On input simulated public parameters Sim.pp, master secret key msk, temporary keys $\{\mathsf{Step}[i].\mathsf{SimTempKey}\}_{j\in[i]}$ and i-th step internal state $\boldsymbol{y}_i$ (including state, reading address, etc., obtained by executing a RAM program on challenge database D^* upto the i-th step), the simulated step key generation outputs i-th step key $\mathsf{Step}[i].\mathsf{SimKEY}$ and $(i+1)$-th simulated temporary key $\mathsf{Step}[i+1].\mathsf{SimTempKey}$. For $j \in [i]$, the algorithm parses
$$\begin{aligned}\mathsf{Step}[j].\mathsf{SimTempKey} = (\mathsf{Step}[j].\mathsf{Simpk}^t, &\{(\mathsf{Step}[j].\mathsf{Simpk}_k^{\mathsf{st}}\}_{k\in[\tau]}, \{(\mathsf{Step}[j].\mathsf{Simpk}_k^{\mathsf{ra}}\}_{k\in[\theta]},\\ &\{\mathsf{Step}[j].\mathsf{Simpk}_k^{\mathsf{db}}\}_{k\in[N]}, \{\mathsf{Step}[j].\mathsf{Simpk}_k^{\mathsf{lt}}\}_{k\in(j+1)\eta}, \mathsf{Step}[j].\mathsf{Simpk})\end{aligned}$$

We use $\mathsf{Step}[j].\mathsf{SimTrdr}$ to denote the trapdoor of $\mathsf{Step}[j].\mathsf{SimTrdr}$. For the i-th step of execution P^{D^*}, let $\{\mathsf{st}_{ik}\}_{k\in[\tau]}$ be the internal state, $\{\mathsf{ra}_{ik}\}_{k\in[\theta]}$ be the reading address, w_i be the writing address, wb_i be the writing bit, and $\{\mathsf{lt}_{ik}\}_{k\in(i+1)\eta}$ be the update list. Denote the i-th step execution status E_i for P^{D^*} as

$$E_i = (\{\mathsf{st}_{ik}\}_{k\in[\tau]}, \{\mathsf{ra}_{ik}\}_{k\in[\theta]}, \mathsf{rb}_i)$$

Output $\perp$ if $\mathsf{Step}[i].\mathsf{rk}^{\mathsf{out}}$ if $f(\{C_j^{\mathsf{st}}(E_i)\}_{j=1}^{\tau}) = 1$, where controlled function f is defined in Definition (4). Otherwise, evaluate the public keys as

$$\mathsf{Step}[i].\mathsf{Simhompk}_j^{\mathsf{st}} \leftarrow \mathsf{CHR.KeyEval}(\{\mathsf{Step}[i].\mathsf{Simpk}_k^{\mathsf{st}}\}_{k\in[\tau]}, \{\mathsf{Step}[i].\mathsf{Simpk}_k^{\mathsf{ra}}\}_{k\in[\theta]}, \mathsf{Step}[i].\mathsf{Simpk}, C_j^{\mathsf{st}})$$

And compute $\mathsf{Step}[i].\mathsf{Simhompk}^{\mathsf{st}} = \mathsf{CHR.KeyEval}(\{\mathsf{Step}[i].\mathsf{Simhompk}_j^{\mathsf{st}}\}_{j\in[\tau]}, \mathcal{C})$, then use the secret key sk_0 to compute the recoding key as

$$\mathsf{Step}[i].\mathsf{rk}^{\mathsf{out}} \leftarrow \mathsf{CHR.ReEncKG}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{Simhompk}^{\mathsf{st}}, \mathsf{sk}_0, \mathsf{pk}_{\mathsf{out}}, h)$$

Do the following:

1. **State circuit** $\{C_k^{\mathsf{st}}\}_{k\in[\tau]}$: For $k \in [\tau]$, compute

$$(\mathsf{Step}[i].\mathsf{Simrk}_k^{\mathsf{st}}, \mathsf{Step}[i+1].\mathsf{Simpk}_k^{\mathsf{st}}) \leftarrow \mathsf{Sim.CHR}_{\mathsf{key}}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{SimTempKey}, \mathsf{Step}[j].\mathsf{SimTrdr}, E_i, C_k^{\mathsf{st}}, \mathsf{Ind})$$

2. **Reading circuit** $\{C_k^{\mathsf{r}}\}_{k\in[\theta]}$, for $k \in [\theta]$, compute

$$(\mathsf{Step}[i].\mathsf{Simrk}_k^{\mathsf{ra}}, \mathsf{Step}[i+1].\mathsf{Simpk}_k^{\mathsf{ra}}) \leftarrow \mathsf{Sim.CHR}_{\mathsf{key}}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{SimTempKey}, \mathsf{Step}[j].\mathsf{SimTrdr}, E_i, C_k^{\mathsf{r}}, \mathsf{Ind})$$

3. **Writing circuit** $\{C_k^{w}\}_{k\in[\theta]}$ and C^{wb}: For $j \in [N]$, compute

$$(\mathsf{Step}[i].\mathsf{Simrk}_j^{\mathsf{w}}, \mathsf{Step}[i+1].\mathsf{Simpk}_j^{\mathsf{db}}) \leftarrow \mathsf{Sim.CHR}_{\mathsf{key}}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{SimTempKey}, \mathsf{Step}[j].\mathsf{SimTrdr}, E_i, C^{\mathsf{w}}, f_j)$$

4. **Update circuit** $\{C_k^{\mathsf{up}}\}_{k\in[i+1]\eta}$: For $k \in [i+2]\eta$, compute

$$\begin{aligned}(\mathsf{Step}[i].\mathsf{Simrk}_k^{\mathsf{lt}}, \mathsf{Step}[i+1].\mathsf{Simpk}_k^{\mathsf{lt}}) \leftarrow \mathsf{Sim.CHR}_{\mathsf{key}}(&\mathsf{pk}_0, \mathsf{Step}[j].\mathsf{Simpk}^t, \{\mathsf{Step}[j].\mathsf{Simpk}_k^{\mathsf{lt}}\}_{k\in(j+1)\eta}\\ &\mathsf{Step}[j].\mathsf{SimTrdr}, w_i, i, \{\mathsf{lt}_{ik}\}_{k\in(i+1)\eta}, C_k^{\mathsf{lt}}, \mathsf{Ind})\end{aligned}$$

5. **Time step**: Compute

$$(\mathsf{Step}[i].\mathsf{Simrk}^t, \mathsf{Step}[i+1].\mathsf{Simpk}^t) \leftarrow \mathsf{Sim.CHR}_{\mathsf{key}}(\mathsf{pk}_0, \mathsf{Step}[j].\mathsf{Simpk}^t, \mathsf{Step}[j].\mathsf{SimTrdr}, g_i)$$

Let circuit $C^{\mathsf{ra}} = \mathcal{C}(\{C_k^{\mathsf{ra}}\}_{k\in[\theta]})$ and calculate $C^{\mathsf{ra}}(\boldsymbol{y}_i) = j^*, C^{\mathsf{up}}(\boldsymbol{y}_i, 0) = k^*$. Then compute

$$\begin{aligned}(\{\mathsf{Step}[i].\mathsf{Simrk}_{jk}^{\mathsf{r}}\}_{j\in[N],k\in[i-1]}, \mathsf{Step}[i+1].\mathsf{Simpk}) \leftarrow \mathsf{Sim.GenCHR}_{\mathsf{key}}(&\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{SimTempKey},\\ &\mathsf{Step}[j].\mathsf{SimTrdr}, E_i, C^{\mathsf{ra}}, C^{\mathsf{up}}, N, i-1)\end{aligned}$$

Output $(E_{i+1}, \mathsf{Step}[i].\mathsf{SimKEY}, \mathsf{Step}[i+1].\mathsf{SimTempKey})$ as

$$\begin{aligned}\mathsf{Step}[i].\mathsf{SimKEY} = (&\mathsf{Step}[i].\mathsf{rk}^{\mathsf{out}}, \mathsf{Step}[i].\mathsf{rk}^t, \{\mathsf{Step}[i].\mathsf{Simrk}_j^{\mathsf{st}}\}, \{\mathsf{Step}[i].\mathsf{Simrk}_k^{\mathsf{ra}}\},\\ &\{\mathsf{Step}[i].\mathsf{Simrk}_j^{\mathsf{w}}\}, \{\mathsf{Step}[i].\mathsf{Simrk}_k^{\mathsf{lt}}\}, \{\mathsf{Step}[i].\mathsf{Simrk}_{jk}^{\mathsf{r}}\})\end{aligned}$$

$$\begin{aligned}\mathsf{Step}[i+1].\mathsf{SimTempKey} = (&\mathsf{Step}[i+1].\mathsf{Simpk}^t, \{(\mathsf{Step}[j].\mathsf{Simpk}_k^{\mathsf{st}}\}_{k\in[\tau]}, \{(\mathsf{Step}[j].\mathsf{Simpk}_k^{\mathsf{ra}}\}_{k\in[\theta]},\\ &\{\mathsf{Step}[j].\mathsf{Simpk}_k^{\mathsf{db}}\}_{k\in[N]}, \{\mathsf{Step}[j].\mathsf{Simpk}_k^{\mathsf{lt}}\}_{k\in(j+1)\eta}, \mathsf{Step}[j].\mathsf{Simpk})\end{aligned}$$

- $\mathsf{Sim.OutKey}(\mathsf{Sim.pp}, \{\mathsf{Step}[j].\mathsf{SimTempKey}\}_{j\in[i]}, P^{D^*})$: On input simulated public parameters $\mathsf{Sim.pp}$, temporary keys $\{\mathsf{Step}[i].\mathsf{SimTempKey}\}_{j\in[i]}$ and i-th step internal state $\boldsymbol{y}_i$ (including state, reading address, etc., obtained by executing a RAM program on challenge database D^* upto the i-th step), the simulated out key generation outputs i-th simulated out key $\mathsf{Step}[i].\mathsf{Simpk}^{\mathsf{out}}$. Denote the i-th step execution status E_i for P^{D^*} as above. Output $\perp$ if $f(\{C_j^{\mathsf{st}}(E_i)\}_{j=1}^{\tau}) = 1$. Otherwise, compute and output

$$\mathsf{Step}[i].\mathsf{Simpk}^{\mathsf{out}} \leftarrow \mathsf{Sim.GenCHR}_{\mathsf{rk}}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{SimTempKey}, E_i, \{C_k^{\mathsf{st}}\}_{k=1}^{\tau}, h)$$

- $\mathsf{Real.StepKey}(\mathsf{Sim.pp}, \mathsf{msk}, \{\mathsf{Step}[j].\mathsf{SimTempKey}\}_{j\in[i]})$: On input simulated public parameters $\mathsf{Sim.pp}$, simulated temporary keys $\{\mathsf{Step}[j].\mathsf{SimTempKey}\}_{j\in[i]}$ and master secret key msk, the real step key generation outputs i-th step key $\mathsf{Step}[i].\mathsf{KEY}$ and $(i+1)$-th temporary key $\mathsf{Step}[i+1].\mathsf{TempKey}$. the algorithm parses

$$\mathsf{Step}[i].\mathsf{SimTempKey} = (\{(\mathsf{Sim.Step}[i].\mathsf{pk}_j^{\mathsf{st}}\}_{j\in[\tau]}, \{(\mathsf{Sim.Step}[i].\mathsf{pk}_j^{\mathsf{ra}}\}_{j\in[\theta]}, \{\mathsf{Sim.Step}[i].\mathsf{pk}_j^{\mathsf{db}}\}_{j\in[N]}, \{\mathsf{Sim.Step}[i+1].\mathsf{pk}_j^{\mathsf{lt}}\}_{j\in i\eta}, \mathsf{Sim.Step}[i].\mathsf{pk})$$

Output $\perp$ if $\mathsf{Step}[i].\mathsf{rk}^{\mathsf{out}}$ if $f(\{C_j^{\mathsf{st}}(E_i)\}_{j=1}^{\tau}) = 1$, where controlled function f is defined in Definition (4). Otherwise, set $C^{\mathsf{st}} = \mathcal{C}(\{C_j^{\mathsf{st}}\}_{j\in[\tau]})$ and compute

$$\mathsf{Step}[i].\mathsf{rk}^{\mathsf{out}} \leftarrow \mathsf{DerivReKey}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{SimTempKey}, \boldsymbol{y}_i, \mathsf{Sim.pk}^{\mathsf{out}}, \mathsf{msk}, C^{\mathsf{st}})$$

For time step, first generate $\mathsf{Step}[i+1].\mathsf{pk}^t$ using $\mathsf{CHR.Setup}(1^\lambda)$, and then compute

$$\mathsf{Step}[i].\mathsf{rk}^t \leftarrow \mathsf{CHR.ReEncKG}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{Simpk}^t, \mathsf{sk}_0, \mathsf{Step}[i+1].\mathsf{pk}^t, g_i)$$

Then for $h \in [N], j \in [\tau], k \in [\theta], \ell \in [(i+1)\eta]$, generate $\mathsf{Step}[i].\mathsf{pk}_h^{\mathsf{db}}, \mathsf{Step}[i+1].\mathsf{pk}_j^{\mathsf{st}}, \mathsf{Step}[i+1].\mathsf{pk}_k^{\mathsf{ra}}, \mathsf{Step}[i+1].\mathsf{pk}_\ell^{\mathsf{lt}}$ using $\mathsf{CHR.Setup}(1^\lambda)$ and then execute

$$\mathsf{Step}[i].\mathsf{rk}_j^{\mathsf{st}} \leftarrow \mathsf{DerivReKey}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{SimTempKey}, \mathsf{msk}, C_j^{\mathsf{st}}, \mathsf{Step}[i+1].\mathsf{pk}_j^{\mathsf{st}}, \mathsf{Ind})$$

$$\mathsf{Step}[i].\mathsf{rk}_k^{\mathsf{ra}} \leftarrow \mathsf{DerivReKey}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{SimTempKey}, \mathsf{msk}, C_k^{\mathsf{r}}, \mathsf{Step}[i+1].\mathsf{pk}_k^{\mathsf{ra}}, \mathsf{Ind})$$

$$\mathsf{Step}[i].\mathsf{rk}_h^{\mathsf{w}} \leftarrow \mathsf{DerivReKey}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{SimTempKey}, \mathsf{msk}, C^{\mathsf{w}}, C^{\mathsf{wb}}, \mathsf{Step}[i].\mathsf{pk}_h^{\mathsf{db}}, f_h)$$

$$\mathsf{Step}[i].\mathsf{rk}_\ell^{\mathsf{lt}} \leftarrow \mathsf{DerivReKey}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{SimTempKey}, \mathsf{msk}, C_\ell^{\mathsf{lt}}, \mathsf{Step}[i+1].\mathsf{pk}_\ell^{\mathsf{lt}}, \mathsf{Ind})$$

where circuit $C^{\mathsf{w}} = \mathcal{C}(\{C_h^{\mathsf{w}}\}_{h\in[\theta]})$ Next, for $k \in [i-1], j \in [N]$, compute the following

$$\mathsf{Step}[i].\mathsf{rk}_{kj}^{\mathsf{r}} \leftarrow \mathsf{DerivReKey}(\mathsf{pk}_0, \mathsf{Step}[i].\mathsf{SimTempKey}, \mathsf{msk}, C^{\mathsf{r}}, C^{\mathsf{up}}, \mathsf{Step}[k].\mathsf{Simpk}_j^{\mathsf{db}}, f_{kj}^{\mathsf{r}})$$

where circuit $C^{\mathsf{r}} = \mathcal{C}(\{C_k^{\mathsf{r}}\}_{k\in[\theta]})$. Output i-th step key $\mathsf{Step}[i].\mathsf{KEY}$ and $(i+1)$-th temporary key $\mathsf{Step}[i+1].\mathsf{TempKey}$.

- $\mathsf{Sim.Enc}(\mathsf{Sim.pp}, D^*, \mu)$: On input simulated public parameters $\mathsf{Sim.pp}$, challenge database $D^* = \{x_i^*\}_{i=1}^N$ and message μ, the simulated encryption algorithm outputs simulated ciphertext $\mathsf{Sim.ct}$. For ease of notation, we use $\mathsf{Step}[0].\mathsf{SimTrdr}$ to denote the all trapdoor information of step 0. The algorithm randomly chooses a secret message s, and for $i \in [N]$, encrypts the database as

$$\mathsf{Step}[0].\mathsf{Simct}_i^{\mathsf{db}} = \mathsf{Sim.CHR}_{\mathsf{ct}}(\mathsf{Step}[0].\mathsf{Simpk}_i^{\mathsf{db}}, \mathsf{Step}[0].\mathsf{SimTrdr}, x_i^*, s)$$

For $i \in [\tau]$, encrypt the initial state as

$$\mathsf{Step}[0].\mathsf{Simct}_i^{\mathsf{st}} = \mathsf{Sim.CHR}_{\mathsf{ct}}(\mathsf{Step}[0].\mathsf{Simpk}_i^{\mathsf{st}}, \mathsf{Step}[0].\mathsf{SimTrdr}, 1, s)$$

Next, it encrypt the auxiliary information as $c_0 = \mathsf{Sim.CHR_{ct}}(\mathsf{Sim.pk}_0, \mathbf{I}, 0, s), u_1 = \mathsf{Sim.CHR_{ct}}(\mathsf{Sim.pk}_0^t,$ $\mathsf{Step}[0].\mathsf{SimTrdr}, 0, s)$ and initial list as $\mathsf{Step}[0].\mathsf{Simct}_i^{\mathsf{lt}} = \mathsf{Sim.CHR_{ct}}(\mathsf{Step}[0].\mathsf{Simpk}_i^{\mathsf{lt}},$ $\mathsf{Step}[0].\mathsf{SimTrdr}, 0, s)$ for $i \in [\eta]$. Finally, choose a random vector ψ over the ciphertext space of CHR.

Theorem 4.4. *Assuming the security of* CHR *for controlled funcions* $\mathcal{F}$*, the scheme* ABE *satisfies the definition of ABE security (c.f. Definition 2.3).*

Proof. Let Q be the number of key queries made by the adversary. We first describe a sequence of hybrids as follows:

Hybrid H_1: This corresponds to the real experiment.

- $\mathcal{A}$ specifies challenge attribute database D^* and message μ.
- Challenger computes $\mathsf{Setup}(1^\lambda)$ to obtain the public parameters pp and secret key msk. Then challenger generates the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pp}, D^*, \mu)$. It sends ct^* and pp to $\mathcal{A}$.
- For $i \in [Q]$, adversary $\mathcal{A}$ specifies the programs P_i such that $P_i^{D^*} \neq 0$. Challenger generates the attribute keys for P_i, for $i \in [Q]$, $\mathsf{sk}_{P_i} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, P_i)$. In more detail, sk_{P_i} is generated as follows:
  - For every $j \in [T]$, compute
  $$\mathsf{Step}[j].\mathsf{KEY}_i \leftarrow \mathsf{Real.StepKey}(\mathsf{pp}, \mathsf{msk}, \{\mathsf{Step}[j].\mathsf{TempKey}_i\}_{k\in[j]})$$
  - Set $\mathsf{sk}_i = (\{\mathsf{Step}[j].\mathsf{KEY}_i\}_{j\in[T]})$.
- Let b be the output of adversary. Output b.

Hybrid H_2: H_2 is the same as H_1 except that it uses $\mathsf{Sim.ABESetup}(1^\lambda, D^*)$ to generate $\mathsf{Sim.pp}$ and msk.

- $\mathcal{A}$ specifies attribute D^* and message μ.
- Challenger generates the setup $\mathsf{Sim.ABESetup}(1^\lambda, D^*)$ to obtain the simulated public key $\mathsf{Sim.pp}$ and master secret key msk. Then challenger generates the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{Sim.pp}, D^*, \mu)$. It sends ct^* and $\mathsf{Sim.pp}$ to $\mathcal{A}$.
- For $i \in [Q]$, adversary $\mathcal{A}$ specifies the programs P_i such that $P_i^{D^*} \neq 0$. Challenger generates the attribute keys for P_i, for $i \in [Q]$, $\mathsf{sk}_{P_i} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, P_i)$. In more detail, sk_{P_i} is generated as follows:
  - For every $j \in [T]$, compute
  $$\mathsf{Step}[j].\mathsf{KEY}_i \leftarrow \mathsf{Real.StepKey}(\mathsf{Sim.pp}, \mathsf{msk}, \{\mathsf{Step}[k].\mathsf{TempKey}_i\}_{k\in[j]})$$
  - Set $\mathsf{sk}_i = (\{\mathsf{Step}[j].\mathsf{KEY}_i\}_{j\in[T]})$.[9]
- Let b be the output of adversary. Output b.

Hybrid $\{\mathsf{H}_{3,i^*,j^*}\}_{i^*\in[Q],j^*\in[T]}$: Simply put, in hybrid $\mathsf{H}_{3,i,j}$, for $i < i^*$, the secret key for query P_i is simulated. For query P_{i^*}, upto the j^*-th step, the step keys are simulated, for step $j > j^*$, the step keys are generated normally. For query P_i, where $i > i^*$, its step keys are all generated normally. We describe it in details below:

[9] The subscript of $\mathsf{Step}[j].\mathsf{KEY}_i$, i here, denotes for the i-th key query.

- Adversary specifies attribute D^* and message μ.
- Challenger generates the setup $\mathsf{Sim.ABESetup}(1^\lambda, D^*)$ to obtain the simulated public key $\mathsf{Sim.pp}$ and master secret key msk. Then challenger generates the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{Sim.pp}, D^*, \mu)$. It sends ct^* and $\mathsf{Sim.pp}$ to $\mathcal{A}$.
- For $i \in [Q]$, adversary $\mathcal{A}$ specifies the programs P_i such that $P_i^{D^*} \neq 0$. Challenger generates the secret key $\mathsf{sk}_{P_i} = (\{\mathsf{Step}[j].\mathsf{KEY}\}_{j\in[T]})$ for P_i, as follows:
  - For $i < i^*$,
    1. For every $j \in [T]$, compute

$$(\mathsf{Step}[j].\mathsf{SimKEY}_i, \mathsf{Step}[j+1].\mathsf{SimTempKey}_i) \leftarrow \mathsf{Sim.StepKey}(\mathsf{Sim.pp}, \mathsf{msk}, \{\mathsf{Step}[k].\mathsf{SimTempKey}_i\}_{k\in[j]}, P_i^{D^*})$$

       And then replace the $\mathsf{Step}[j].\mathsf{Simrk}_i^{\mathsf{out}}$ in $\mathsf{Step}[j].\mathsf{SimKEY}_i$ using

$$\mathsf{Step}[j].\mathsf{Simrk}_i^{\mathsf{out}} \leftarrow \mathsf{Sim.OutKey}(\mathsf{Sim.pp}, \{\mathsf{Step}[k].\mathsf{SimTempKey}_i\}_{k\in[j]}, P_i^{D^*})$$

    2. Set $\mathsf{sk}_i = (\{\mathsf{Step}[j].\mathsf{SimKEY}_i\}_{j\in[T]})$.
  - For $i = i^*$,
    1. For $j < j^*$, generate

$$(\mathsf{Step}[j].\mathsf{SimKEY}_i, \mathsf{Step}[j+1].\mathsf{SimTempKey}_i) \leftarrow \mathsf{Sim.StepKey}(\mathsf{Sim.pp}, \mathsf{msk}, \{\mathsf{Step}[k].\mathsf{SimTempKey}_i\}_{k\in[j]}, P_{i^*}^{D^*})$$

    2. For $j = j^*$, generate

$$(\mathsf{Step}[j].\mathsf{KEY}_i, \mathsf{Step}[j{+}1].\mathsf{TempKey}_i) \leftarrow \mathsf{Real.StepKey}(\mathsf{Sim.pp}, \mathsf{msk}, \{\mathsf{Step}[j].\mathsf{SimTempKey}_i\}_{j\in[i]})$$

    3. For $j > j^*$, generate

$$\mathsf{Step}[j].\mathsf{KEY}_i \leftarrow \mathsf{Real.StepKey}(\mathsf{Sim.pp}, \{\mathsf{Step}[k].\mathsf{SimTempKey}_i\}_{k\in[j^*]}, \{\mathsf{Step}[k].\mathsf{TempKey}_i\}_{k=j^*+1}^{j}, \mathsf{msk})$$

    4. Set $\mathsf{sk}_i = (\{\mathsf{Step}[j].\mathsf{SimKEY}_i\}_{j\in[T], j<j^*}, \{\mathsf{Step}[j].\mathsf{KEY}_i\}_{j\in[T], j\geq j^*})$.
  - For $i > i^*$,
    1. For every $j \in [T]$, generate

$$\mathsf{Step}[j].\mathsf{KEY}_i \leftarrow \mathsf{Real.StepKey}(\mathsf{Sim.pp}, \mathsf{msk}, \{\mathsf{Step}[k].\mathsf{TempKey}_i\}_{k\in[j]})$$

    2. Set $\mathsf{sk}_i = (\{\mathsf{Step}[j].\mathsf{KEY}_i\}_{j\in[T]})$.
- Let b be the output of adversary. Output b.

Hybrid $\{\widetilde{\mathsf{H}_{3,i^*,j^*}}\}_{i^*\in[Q], j^*\in[T]}$: Simply put, hybrid $\widetilde{\mathsf{H}_{3,i^*,j^*}}$ happens right after H_{3,i^*,j^*}, and the only difference between these two consecutive hybrids is in $\widetilde{\mathsf{H}_{3,i^*,j^*}}$, the recoding key $\mathsf{Step}[j^*].\mathsf{Simrk}_{i^*}^{\mathsf{out}}$ in $\mathsf{Step}[j^*].\mathsf{SimKEY}_i^*$ is generated using algorithm $\mathsf{Sim.OutKey}$ instead of using $\mathsf{Sim.StepKey}$ (in hybrid H_{3,i^*,j^*}).

Hybrid H_4: In H_4, the secret keys for all queries are simulated without using msk. Therefore, we sample the anchor public key $\mathsf{Sim.pk}_0$ (with its trapdoor $\mathbf{I}$) randomly from space $\mathbb{Z}_q^{n\times m}$.

- Adversary specifies attribute D^* and message μ.
- Challenger generates the setup $\mathsf{Sim.ABESetup}(1^\lambda, D^*)$ to obtain the simulated public key $\mathsf{Sim.pp}$. Then challenger generates the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{Sim.pp}, D^*, \mu)$. It sends ct^* and $\mathsf{Sim.pp}$ to $\mathcal{A}$.
- For $i \in [Q]$, adversary $\mathcal{A}$ specifies the programs P_i such that $P_i^{D^*} \neq 0$. Challenger generates the attribute keys for P_i, for $i \in [Q]$. In more detail, sk_{P_i} is generated as follows:
  1. For every $j \in [T]$, generate
  $$(\mathsf{Step}[j].\mathsf{SimKEY}_i, \mathsf{Step}[j{+}1].\mathsf{SimTempKey}_i) \leftarrow \mathsf{Sim.StepKey}(\mathsf{Sim.pp}, \{\mathsf{Step}[k].\mathsf{SimTempKey}_i\}_{k\in[j]})$$
  2. Set $\mathsf{sk}_i = (\{\mathsf{Step}[j].\mathsf{SimKEY}_i\}_{j\in[T]})$.
- Let b be the output of adversary. Output b.

Hybrid H_5: H_5 is the same as H_4 except that it simulates challenge ciphertext.

- Adversary specifies attribute D^* and message μ.
- Challenger generates the setup $\mathsf{Sim.ABESetup}(1^\lambda)$ to obtain the simulated public key $\mathsf{Sim.pp}$. It sends $\mathsf{Sim.pp}$ to $\mathcal{A}$.
- Challenger generates the simulated ciphertext $\mathsf{Sim.ct}^* \leftarrow \mathsf{Sim.Enc}(\mathsf{pp}, D^*, \mu)$. It sends $\mathsf{Sim.ct}^*$ to $\mathcal{A}$.
- For $i \in [Q]$, adversary $\mathcal{A}$ specifies the programs P_i such that $P_i^{D^*} \neq 0$. Challenger generates the attribute keys for P_i, for $i \in [Q]$, $\mathsf{sk}_{P_i} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, P_i)$. In more detail, sk_{P_i} is generated as follows:
  1. For every $j \in [T]$, generate
  $$(\mathsf{Step}[j].\mathsf{SimKEY}_i, \mathsf{Step}[j{+}1].\mathsf{SimTempKey}_i) \leftarrow \mathsf{Sim.StepKey}(\mathsf{Sim.pp}, \{\mathsf{Step}[k].\mathsf{SimTempKey}_i\}_{k\in[j]})$$
  2. Set $\mathsf{sk}_i = (\{\mathsf{Step}[j].\mathsf{SimKEY}_i\}_{j\in[T]})$.
- Let b be the output of adversary. Output b.

Lemma 4.5. *By the indistinguishability of setup property of* CHR *scheme (c.f. Definition 3.2), we have* $\mathsf{H}_1 \overset{s}{\approx} \mathsf{H}_2$.

Proof. The only difference between hybrid H_1 and H_2 is that in H_2, the public parameters generated by $\mathsf{Sim.ABESetup}(1^\lambda, D^*)$ as described above. By the indistinguishability of setup property of CHR, the distribution $\{\mathsf{Sim.pp}\}$ is statistically close to $\{\mathsf{pp}\}$. Therefore, we have $\mathsf{H}_1 \overset{s}{\approx} \mathsf{H}_2$. □

Lemma 4.6. *The output distributions of hybrids* H_2 *and* $\mathsf{H}_{3,1,0}$ *are identical.*

Proof. As described above, hybrid $\mathsf{H}_{3,1,0}$ is obtained by simulating $\{\mathsf{Step}[0].\mathsf{SimKEY}_1\}$ and generating all other step keys normally. As $\{\mathsf{Step}[0].\mathsf{SimKEY}_1\}$ does not exists in sk_1, therefore we have that hybrids H_2 and $\mathsf{H}_{3,1,0}$ are identical. □

Lemma 4.7. *By the indistinguishability of simulated keys property of* CHR *scheme (c.f. Definition 3.3) with respect to* $\mathcal{E}_{\mathsf{aux}}$, *we have* $\widetilde{\mathsf{H}_{3,i^*,j^*}} \overset{s}{\approx} \mathsf{H}_{3,i^*,j^*+1}$, *where* $j^* \in [T-1]$ *and* $\mathcal{E}_{\mathsf{aux}}$ *consists of two cases:*

- $\mathcal{E}_{\mathsf{aux}}$ *is the distribution of simulated keys produced by* $\mathsf{Sim.CHRSetup}$ *in the* $(j^*)^{th}$ *step.*

- $\mathcal{E}_{\mathsf{aux}}$ *is the distribution of simulated keys produced in* 1^{st} *step,* $\mathsf{Sim.CHR_{key}}$

Proof. The difference between $\widetilde{\mathsf{H}_{3,i^*,j^*}}$ and H_{3,i^*,j^*+1} is that in H_{3,i^*,j^*+1} the (j^*+1)-th step key $\mathsf{Step}[j^*+1].\mathsf{SimKEY}_{i^*}$ of query i^* is simulated instead of normal generation. In algorithm $\mathsf{Sim.StepKey}(\mathsf{Sim.pp}, \{\mathsf{Step}[k].\mathsf{SimTempKey}_{i^*}\}_{k\in[j^*+1]})$, $\mathsf{Step}[i].\mathsf{Simrk}^{\mathsf{out}}$ and the others in $\mathsf{Step}[j^*+1].\mathsf{SimKEY}_{i^*}$ are computed as described above. By indistinguishability of simulated keys (c.f. Definition 3.3), we have

$$\{\mathsf{Step}[j^*+1].\mathsf{SimKEY}_{i^*}, \mathsf{Step}[j^*+2].\mathsf{SimTempKey}\} \overset{s}{\approx} \{\mathsf{Step}[j^*+1].\mathsf{KEY}_{i^*}, \mathsf{Step}[j^*+2].\mathsf{TempKey}\}$$

Thus, we have $\{\mathsf{Step}[j^*+1].\mathsf{SimKEY}_{i^*}\} \overset{s}{\approx} \{\mathsf{Step}[j^*+1].\mathsf{KEY}_{i^*}\}$, which means $\widetilde{\mathsf{H}_{3,i^*,j^*}} \overset{s}{\approx} \mathsf{H}_{3,i^*,j^*+1}$. □

Lemma 4.8. *By the indistinguishability of recoding keys property of* CHR *scheme (c.f. Definition 3.4) with respect to* $\mathcal{E}_{aux}$*, we have* $\mathsf{H}_{3,i^*,j^*} \overset{s}{\approx} \widetilde{\mathsf{H}_{3,i^*,j^*}}$*, where* $j^* \in [T-1]$ *and* $\mathcal{E}_{aux}$ *is the distribution of public keys produced by* $\mathsf{Sim.CHRSetup}$ *in the* $(T-1)^{th}$ *step.*

Proof. The only difference between these two consecutive hybrids is in $\widetilde{\mathsf{H}_{3,i^*,j^*}}$, the recoding key $\mathsf{Step}[j^*].\mathsf{Simrk}^{\mathsf{out}}_{i^*}$ in $\mathsf{Step}[j^*].\mathsf{SimKEY}^*_i$ is generated using algorithm $\mathsf{Sim.OutKey}$ instead of using $\mathsf{Sim.StepKey}$ (in hybrid H_{3,i^*,j^*}). By the indistinguishability of recoding keys property of CHR scheme (c.f. Definition 3.4), the distribution of recoding key $\mathsf{Step}[j^*].\mathsf{Simrk}^{\mathsf{out}}_{i^*}$ is computationally close to $\mathsf{Step}[j^*].\mathsf{rk}^{\mathsf{out}}_{i^*}$, thus we have $\mathsf{H}_{3,i^*,j^*} \overset{s}{\approx} \widetilde{\mathsf{H}_{3,i^*,j^*}}$. □

Lemma 4.9. *The output distributions of hybrids* $\widetilde{\mathsf{H}_{3,i^*,T}}$ *and* $\mathsf{H}_{3,i^*+1,0}$ *are identical, when* $i^* \in [Q]$.

Proof. The only difference between hybrids $\widetilde{\mathsf{H}_{3,i^*,T}}$ and $\mathsf{H}_{3,i^*+1,0}$ is that in hybrid $\mathsf{H}_{3,i^*+1,0}$, the step key $\{\mathsf{Step}[0].\mathsf{SimKEY}_{i+1}\}$ is simulated. As $\{\mathsf{Step}[0].\mathsf{SimKEY}_{i+1}\}$ does not exists in sk_{i+1}, thus we have that hybrids $\widetilde{\mathsf{H}_{3,i^*,T}}$ and $\mathsf{H}_{3,i^*+1,0}$ are identical. □

Lemma 4.10. *The output distributions of hybrids* $\widetilde{\mathsf{H}_{3,Q,T}}$ *and* H_4 *are statistically close.*

Proof. In hybrids $\widetilde{\mathsf{H}_{3,Q,T}}$ and H_4, the only difference is that in $\widetilde{\mathsf{H}_{3,Q,T}}$, the anchor public key pk_0 is generated along with sk_0, using algorithm $\mathsf{TrapGen}$, while in H_4 the anchor public key pk_0 is sampled from random distribution. By Corollary 3.8, hybrids $\widetilde{\mathsf{H}_{3,Q,T}}$ and H_4 are statistically close. □

Lemma 4.11. *By the pseudorandomness of ciphertexts of* CHR *scheme (c.f. Definition 3.5), we have* $\mathsf{H}_4 \overset{c}{\approx} \mathsf{H}_5$.

Proof. The only difference between $\mathsf{H}_4 \overset{c}{\approx} \mathsf{H}_5$ is that in H_5 is challenge ciphertext is generated by algorithm $\mathsf{Sim.Enc}(\mathsf{Sim.pp}, D^*, \mu)$, where $\mathsf{CHR.Enc}$ is used as a subroutine and a randomly chose vector ψ is chosen over ciphertext space. By the pseudorandomness of ciphertexts of CHR, we have that the ciphertexts of both hybrids are computationally close to the uniformly random distribution over ciphertext space. Therefore, we have $\mathsf{H}_4 \overset{c}{\approx} \mathsf{H}_5$. □

Combining the hybrids and lemmas proved above, we prove that our ABE construction is secure, as defined in Definition 2.3. □

References

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Gilbert [Gil10], pages 553–572.

[AC16] Shashank Agrawal and Melissa Chase. A study of pair encodings: predicate encryption in prime order groups. In *Theory of Cryptography Conference*, pages 259–288. Springer, 2016.

[AJS15] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation with constant size overhead. *IACR Cryptology ePrint Archive*, 2015:1023, 2015.

[AP10] Joël Alwen and Chris Peikert. Generating shorter bases for hard random lattices. *Theory of Computing Systems*, 48(3):535–553, 2010.

[AP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 297–314. Springer, Heidelberg, August 2014.

[AS16] Prabhanjan Ananth and Amit Sahai. Functional encryption for turing machines. In *Theory of Cryptography Conference*, pages 125–153. Springer, 2016.

[Att14] Nuttapong Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 557–577. Springer, 2014.

[BF01] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Advances in CryptologyCRYPTO 2001*, pages 213–229. Springer, 2001.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit abe and compact garbled circuits. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 533–556. Springer, 2014.

[BGJ+16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 345–356. ACM, 2016.

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In Servedio and Rubinfeld [SR15], pages 439–448.

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

[BP14] Elette Boyle and Rafael Pass. Limits of extractability assumptions with distributional auxiliary input. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 236–261. Springer, 2014.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. *Theory of Cryptography*, pages 253–273, 2011.

[BSW16] Mihir Bellare, Igors Stepanovs, and Brent Waters. New negative results on differing-inputs obfuscation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 792–821. Springer, 2016.

[BV16] Zvika Brakerski and Vinod Vaikuntanathan. Circuit-abe from lwe: unbounded attributes and semi-adaptive security. In *Annual Cryptology Conference*, pages 363–384. Springer, 2016.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for RAM programs. In Servedio and Rubinfeld [SR15], pages 429–437.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In Gilbert [Gil10], pages 523–552.

[CR72] Stephen A Cook and Robert A Reckhow. Time-bounded random access machines. In *Proceedings of the fourth annual ACM symposium on Theory of computing*, pages 73–80. ACM, 1972.

[DG17] Nico Döttling and Sanjam Garg. Identity-based encryption from the diffie-hellman assumption. In *Annual International Cryptology Conference*, pages 537–569. Springer, 2017.

[DKW16] Apoorvaa Deshpande, Venkata Koppula, and Brent Waters. Constrained pseudorandom functions for unconstrained inputs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 124–153. Springer, 2016.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In *Advances in Cryptology–CRYPTO 2013*, pages 479–499. Springer, 2013.

[GGH+16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM Journal on Computing*, 45(3):882–929, 2016.

[GGHW14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In *International Cryptology Conference*, pages 518–535. Springer, 2014.

[GGHZ14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure attribute based encryption from multilinear maps. *IACR Cryptology EPrint Archive*, 2014:622, 2014.

[GHL+14] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled RAM revisited. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 405–422. Springer, Heidelberg, May 2014.

[Gil10] Henri Gilbert, editor. *EUROCRYPT 2010*, volume 6110 of *LNCS*. Springer, Heidelberg, May 2010.

[GJPS08] Vipul Goyal, Abhishek Jain, Omkant Pandey, and Amit Sahai. Bounded ciphertext policy attribute based encryption. *Automata, languages and programming*, pages 579–591, 2008.

[GKP+13a] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 555–564. ACM, 2013.

[GKP+13b] Shafi Goldwasser, Yael Tauman Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In *Advances in Cryptology–CRYPTO 2013*, pages 536–553. Springer, 2013.

[GKW16] Rishab Goyal, Venkata Koppula, and Brent Waters. Semi-adaptive security and bundling functionalities made generic and easy. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 361–388. Springer, Heidelberg, October / November 2016.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm, 2006.

[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.

[GV15] Sergey Gorbunov and Dhinakaran Vinayagamurthy. Riding on asymmetry: Efficient ABE for branching programs. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 550–574. Springer, Heidelberg, November / December 2015.

[GVW15a] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. *Journal of the ACM (JACM)*, 62(6):45, 2015.

[GVW15b] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from lwe. In *Annual Cryptology Conference*, pages 503–523. Springer, 2015.

[GVW15c] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Servedio and Rubinfeld [SR15], pages 469–477.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In Servedio and Rubinfeld [SR15], pages 419–428.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. *Advances in Cryptology–EUROCRYPT 2008*, pages 146–162, 2008.

[LOS+10] Allison B Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *Eurocrypt*, volume 6110, pages 62–91. Springer, 2010.

[LW11] Allison Lewko and Brent Waters. Decentralizing attribute-based encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 568–588. Springer, 2011.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. *IACR Cryptology ePrint Archive*, 2010:556, 2010.

[OSW07] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 195–203. ACM, 2007.

[PF79] Nicholas Pippenger and Michael J Fischer. Relations among complexity measures. *Journal of the ACM (JACM)*, 26(2):361–381, 1979.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, volume 7194, pages 422–439. Springer, 2012.

[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.

[SR15] Rocco A. Servedio and Ronitt Rubinfeld, editors. *47th ACM STOC*. ACM Press, June 2015.

[SW+05] Amit Sahai, Brent Waters, et al. Fuzzy identity-based encryption. In *Eurocrypt*, volume 3494, pages 457–473. Springer, 2005.

[W+09] Brent Waters et al. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In *Crypto*, volume 5677, pages 619–636. Springer, 2009.

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In *Eurocrypt*, volume 3494, pages 114–127. Springer, 2005.

[Wat12] Brent Waters. Functional encryption for regular languages. In *CRYPTO*, volume 7417, pages 218–235. Springer, 2012.

[Wee14] Hoeteck Wee. Dual system encryption via predicate encodings. In *TCC*, volume 8349, pages 616–637, 2014.