Towards Attribute-Based Encryption for RAMs from LWE: Sub-linear Decryption, and More

Prabhanjan Ananth[1], Xiong Fan[2], and Elaine Shi[2]

[1] MIT CASIL, Boston, MA, USA. prabhanjan@csail.mit.edu
[2] Cornell University, Ithaca, NY, USA. {xfan, elaine}@cs.cornell.edu.

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 where the function associated with the ABE key is represented by a random access machine (RAM) and a database, with the runtime of the RAM program being sublinear in the database size. 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 is the first construction based on standard assumptions. Previously, Goldwasser et al. achieved this result based on non-falsifiable knowledge assumptions. We also consider a dual notion of ABE for RAMs, where the database is in the ciphertext and we show how to achieve this dual notion, albeit with large attribute keys, also based on learning with errors.

1 Introduction

Attribute-based encryption (ABE) [54] 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 [54,42,50,41,56,45,57,37,26,36,13,28,58,39,21] has led to several useful applications including verifiable computation [51] and reusable garbled circuits [35]. Special cases of ABE, such as identity-based encryption [12,55,24,18], and generalizations of ABE, such as functional encryption [14,49,25], 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 exceptions, all these constructions [42,56,46,38,13,39] model the program, associated with the constrained keys, as circuits. Real-world programs, however, are composed in the so-called Random Access Machine (RAM) model. In this

paper, we consider the natural question of constructing attributed-based encryption scheme for RAM programs.

As in the circuit setting, an attribute-based encryption scheme for RAM programs consists of the setup, key generation, encryption and decryption algorithms. The key generation algorithm takes as input the master secret key, program P, database D and produces an attribute key associated with (P, D). The encryption algorithm takes as input attribute x, secret message μ and produces a ciphertext. Decrypting this ciphertext using the key of (P, D) yields the secret message μ if and only if $P^D(x) = 0$.

Towards constructing attribute-based encryption for RAMs, a naïve approach is to convert RAM programs generically to circuits: a RAM program initialized with N words of memory and running in time T can be converted to a circuit of size $O((N + T) \cdot T)$ and depth T. Thus, the approach via naïve RAM-to-circuit conversion would incur a $(N + T) \cdot T$ multiplicative factor in the decryption time. In this paper, we are interested in the common case when T is *sublinear* in N, e.g., imagine that the RAM is initialized with a large database with N entries and the RAM program models a binary search on the database. Goldwasser et al. [36] gave the first feasibility result of ABE for RAM programs with sub-linear decryption time based on the existence of extractable witness encryption and succinct non-interactive arguments of knowledge (SNARK). Recent works [27,15,10], however, have brought into question the veracity of the assumptions of extractable witness encryption and SNARKs.

Since building ABE for RAMs on solid cryptographic foundations is an important problem, we ask the following natural question:

> *Is there an ABE for RAMs scheme with sublinear decryption overhead based on standard assumptions? More specifically, we would like the decryption overhead to be* $o(N) \cdot \mathsf{poly}(T, \lambda)$ *where* λ *is the security parameter.*

1.1 Our Results and Contributions

ABE for RAMs with sub-linear decryption efficiency. We construct an ABE scheme for RAMs with sub-linear decryption overhead from the Learning With Errors (LWE) assumption. Henceforth we assume that the scheme is parameterized with N and T which denote the size of the database and the upper bound on the runtime of the RAM respectively, and a security parameter denoted by λ. Our construction achieves the following:

- There is an initial setup phase that generates a global public parameter of size $\mathsf{poly}(T, \lambda)$ and master secret key of size $\mathsf{poly}(T, \lambda)$.
- Anyone that has access to the public parameters can encrypt a message μ to an attribute x of size λ (For simplicity, we set the size of attribute x to be λ. However, the size of the attribute can be set in advance and of any fixed polynomial in λ) — later x will serve as an input to a RAM program. The encryption time and ciphertext size is upper bounded by $\mathsf{poly}(T, \lambda)$.

– An authority with master secret key can generate a decryption key $\mathsf{sk}_{P,D}$ given the description of a RAM program P (where the description will include the RAM's next instruction circuit) and a long attribute vector denoted by D of size N, and the size of the secret key $\mathsf{sk}_{P,D}$ is upper bounded $\mathsf{poly}(T, N, \lambda)$.
– Finally, given the ciphertext ct_x that is associated with the attribute x, anyone with the public parameters and the decryption key $\mathsf{sk}_{P,D}$ can decrypt the plaintext message μ if $P^D(x) = 0$; and importantly decryption time is $\mathsf{poly}(T, \lambda)$, i.e., independent of the RAM's initial memory size N. For security, we show that an adversary learns nothing about the encrypted plaintext μ if he does not possess any sk_P such that $P^D(x) = 0$.

More formally, our main theorem is the following:

Theorem 1.1 (ABE for RAMs) *Assuming the hardness of the Learning With Errors problem (with sub-exponential modulus)*[3]*, there exists an ABE scheme for RAMs with* $\mathsf{poly}(T, \lambda)$ *decryption efficiency, i.e., independent of* N*.*

Moreover, (i) the cost of generating public parameters is $\mathsf{poly}(T, \lambda)$*, i.e., independent of* N*, (ii) the cost of generating secret keys is* $\mathsf{poly}(N, T, \lambda)$ *and, (iii) the cost of generating ciphertexts is* $\mathsf{poly}(T, \lambda)$*.*

Input-Specific Runtime. While the construction in the above theorem has decryption complexity proportional to the worst case running time of the RAM programs, we can transform this scheme into another scheme where the decryption complexity is *input-specific*. This is performed by running $\log T$ copies of the scheme by setting the worst case runtime of the first scheme to be 2, second scheme to be 2^2, so on and the runtime of the $(\log T)$-th scheme is set to be T. This idea has been used in prior works (for instance [36]). Note that this increases the size of the public parameters, keys and ciphertexts by a multiplicative factor of $\log T$.

On fixing the attribute length. In our construction, the length of the attribute is fixed at the time of setup. In particular, both the public parameters and the attribute keys grow with the length of the attribute. Note that the attribute keys already grow proportional to the length of the database and the database size is typically larger than the attribute length. However, achieving public key sizes independent of the attribute length would be interesting, especially given that there are works [21] that have achieved this in the context of ABE for circuits.

Comparison with [36]. As mentioned earlier, [36] also achieves ABE for RAMs with sub-linear decryption complexity from exotic assumptions. The only drawback in our scheme in comparison with [36] is that the parameters in the construction of [36], specifically the public parameters, ciphertext size and the key sizes do not grow with the maximum time bound. On the other hand, our parameters do grow with the maximum time bound T. There is evidence to suggest

[3] All known lattice-based ABE for circuits [13] are based on the same assumption.

that an ABE for RAMs scheme whose parameters do not grow with the maximum time bound can only be based on strong cryptographic assumptions. In particular such a notion would imply succinct randomized encodings for Turing machines [11,6][4]; a notion, despite numerous attempts, we don't yet know how to build from well-studied assumptions.

Comparison with Circuit-Based Schemes. We compare the parameters we obtain in our scheme with the parameters obtained in the naive approach of RAM-to-circuit conversion and then applying previously known ABE for circuits schemes. Refer to Figure 1.

Schemes	# of $\mathbb{Z}_q^{n\times m}$ Public key	# of $\mathbb{Z}_q^m$ Ciphertext of (x,μ)	Size of Key of (P,D)	Decryption complexity
Via ABE for circuits [13]	$\tilde{O}(\|x\|)$	$\tilde{O}(\|x\|)$	$\|P\|+N+\binom{\tilde{O}(1)}{\#\text{ of }\mathbb{Z}_q^{n\times m}}$	$\tilde{O}((T+N)T)$
Via ABE for circuits [21]	$\tilde{O}(1)$	$\tilde{O}(\|x\|)$	$\|P\|+N+\binom{\tilde{O}(1)}{\#\text{ of }\mathbb{Z}_q^{n\times m}}$	$\tilde{O}((T+N)T)$
Our Work for RO-RAMs	$\tilde{O}(\|x\|+T)$	$\tilde{O}(\|x\|+T)$	$\|P\|+\binom{\tilde{O}(TN)}{\#\text{ of }\mathbb{Z}_q^{n\times m}}$	$\tilde{O}(T)$
Our Work for RAMs	$\tilde{O}(\|x\|+T)$	$\tilde{O}(\|x\|+T)$	$\|P\|+\binom{\tilde{O}(T(N+T))}{\#\text{ of }\mathbb{Z}_q^{n\times m}}$	$\tilde{O}(T)$

Fig. 1: We compare the parameters in our work with previous works. The lattice dimension (n,m,q) is asymptotically the same in all three approaches. The $\tilde{O}$ notation suppresses poly-logarithmic factors (in N and T). The encryptor takes an auxiliary input x and the key generator takes as input a program P and a database D of size N. The decryption complexity is calculated in terms of vector-matrix multiplication over $\mathbb{Z}_q$. The attribute key is generated for a RAM program P with worst case runtime to be T and it takes time t to compute on D. In previous works, an attribute key for P is generated by first transforming it into a circuit of size $(T+N)$ and depth T and then generating an attribute key for the resulting circuit.

While the key sizes in our scheme are larger than the ones obtained via circuit ABE schemes, our scheme has the following advantage over ABE for circuit schemes: since the same decryption keys, once generated, can be applied to (unbounded polynomially) many ciphertexts, the cost of key generation and its size can be *amortized over multiple decryption queries*. This is especially useful in scenarios, where a client can perform a one-time cost of generating

[4] The works [11,6] show implication of ABE for Turing machines (as defined in [6]) to succinct randomized encodings (Appendix A.5 in [11])

keys and sending it over to the server during the offline phase and during the online phase, can verifiably delegate multiple computations by suitably sending encryptions of its inputs; note that in this scenario, we are only interested in verifying whether the server has performed the computation correctly and not hiding the computation itself.

Dual ABE for RAMs. We also consider an alternate notion of ABE for RAMs, that we call dual ABE for RAMs. In this notion, the database is part of the ciphertext and not the key. That is, the key generation procedure now only takes as input the master secret key and the RAM program P while the encryption procedure takes as input the database D, the auxiliary input x (in the technical section, we consider x to be part of D) and the secret message μ. As before, we require that it should be possible to recover μ if indeed $P^D(x) = 0$.

We demonstrate a construction of dual ABE for RAMs, also based on the learning with errors problem with the same decryption efficiency as stated in Theorem 1.1. In more detail,

Theorem 1.2 (Dual ABE for RAMs) *Assuming the hardness of the Learning With Errors problem (with sub-exponential modulus)*[5]*, there exists a dual ABE scheme for RAMs with* $\mathsf{poly}(T, \lambda)$ *decryption efficiency, i.e., independent of* N*.*

Moreover, (i) the cost of generating public parameters is $\mathsf{poly}(N, T, \lambda)$*, (ii) the cost of generating secret keys is* $\mathsf{poly}(N, T, \lambda)$ *and, (iii) the cost of generating ciphertexts is* $\mathsf{poly}(N, T, \lambda)$*.*

On Large Attribute Keys. Unlike our construction of ABE for RAMs (Theorem 1.1), our construction of dual ABE for RAMs has public keys that grow proportional to the size of the database. Moreover, even the size of our attribute keys depends on the database size. While this is not inherent and an undesirable feature of our scheme, we see our work as a first step in achieving dual ABE schemes beyond circuits; note that none of the previous ABE schemes achieved sub-linear decryption property and our dual ABE construction is the first to do so.

1.2 Technical Overview

We give an overview of the techniques employed in our main construction. We later reuse some of the techniques used in our main construction to obtain a construction in the dual setting.

Starting Point: Garbled RAMs. A natural idea to build ABE for RAMs is to use garbled RAMs [32,29,30]. A garbled RAM allows for separately encoding a RAM program[6]-database pair (P, D) and encoding an input x such that the encodings only leak the output $P^D(x)$; computing both the encodings requires

[5] All known lattice-based ABE for circuits [13] are based on the same assumption.
[6] The formal definition of a RAM program can be found in the preliminaries.

a private key not revealed to the adversary. Notice that a garbled RAM scheme implies a *one-time, secret key* ABE for RAM scheme; meaning that the adversary only gets to make a single ciphertext query and a single attribute key query. Indeed, it is unclear how to remove the one-time restriction while simultaneously achieve a public-key ABE for RAMs scheme by generically using garbled RAMs. Hence, we circumvent this conundrum by diving into the innards of the existing garbled RAMs schemes. The hope would be to adopt some of the techniques used in constructing garbled RAMs to build an ABE for RAMs scheme.

Most of the current known constructions of garbled RAMs have the following blueprint: to garble a RAM program P (associated with a step circuit C), database D, generate T garbled circuits, where T is an upper bound on the running time of P. The i^{th} garbled circuit performs the execution of the i^{th} time step of P. Also, every entry of the database D is suitably encoded using an appropriate encoding scheme (for instance, in [32], an IBE (identity-based encryption) key is associated with every entry of the database). The garbling of P consists of all the T garbled circuits and the encoding of the database D. The encoding of input x consists of wire labels of the first garbled circuit corresponding to the input x.

To evaluate a garbling of P on an encoded database D and wire labels of x, perform the following operations for $i = 1, \ldots, T-1$:

- If $i = 1$, evaluate the first garbled circuit on wire labels of x.
- If $i > 1$, evaluate the i^{th} garbled circuit to obtain output encodings of the i^{th} step of execution of P^D on x.
- Next, we compute the *recoding step* that converts the output encodings of the i^{th} step into the input encodings of the $(i+1)^{th}$ step. These input encodings will be fed to the $(i+1)^{th}$ garbled circuit.

The output of the T^{th} garbled circuit determines the output of execution of $P^D(x)$.

From Garbled RAMs to ABE for RAMs: Challenges. Towards realizing our hope of using garbled RAMs techniques to build an ABE for RAMs scheme, we encounter the following fundamental issues:

- The garbling and encoding operations in a garbled RAM scheme are inherently secret-key operations; they require a shared secret-key to compute garbled program and database encodings respectively. Since our goal is to construct public-key ABE for RAMs, the encryptor can perform neither the garbling nor the encoding procedures.
- Garbling schemes typically do not offer any reusability property[7]; they are useful only when a single computation needs to be hidden. It is unclear how to

[7] An exception is the reusable garbling scheme of Goldwasser et al. [35], however their scheme only offers one-sided reusability: that is, their scheme only allows the adversary to get a single garbled circuit which can be reused across multiple input encodings. This is not useful in our setting since the adversary gets to query multiple keys. Moreover, just like any garbling scheme, even reusable garbled circuits require secret-key to perform the encoding operations.

use garbled circuits, an integral part of current garbled RAM constructions, in the ABE setting, where multiple attribute keys need to be issued.
– Tied to the issue of using garbled circuits is also the issue of implementing the recoding step. We need to implement a recoding step that can be reused across different computations.

Our Solution in a Nutshell. The main technical contribution of this paper is to identify a template to solve this problem and instantiate this template using a novel combination of existing lattice-based techniques.

We describe our template of ABE for RAMs. This will be an over-simplifcation of our actual scheme and is intended to help the reader towards understanding our final construction. For now, focus on the setting when the keys are only associated with read-only RAMs (i.e., they only read from the memory and never write into the memory). This template can be easily adapted to the setting where the program can also write to the memory.

– A key for a program P and a database D will consist of two parts: the first part, denoted by sk_D, is associated with the database and the second part, denoted by $(\mathsf{StepKey}_1, \ldots, \mathsf{StepKey}_T)$, consists of T sets of **recoding** keys with T being the maximum running time of $P^D(\cdot)$.
– A ciphertext for an input x and a secret message μ consists of two parts $(\mathsf{ct}_x^{(1)}, \mathsf{ct}_x^{(2)})$ and an encryption of μ, namely ct_μ (we will need a scheme that satisfies some specific properties): the first part $\mathsf{ct}_x^{(1)}$ serves as encoding of the initial input to the step circuit of the RAM program. We describe the role of the second part $\mathsf{ct}_x^{(2)}$ when we describe the decryption operation below.
– The decryption of a ciphertext of (x, μ) using a key of (P, D) proceeds in the following steps:
  • **Translation Step**: First, using the second part of the ciphertext, i.e., using $\mathsf{ct}_x^{(2)}$, and using the key associated with the database sk_D in the attribute key, obtain a probabilistic encoding of D.
  • The following operations are executed for time steps $t = 1, \ldots, T$:
    ∗ **Evaluation Step**: Homomorphically evaluate on the input encodings of the t^{th} step to obtain the output encodings of the t^{th} step. This is akin to the evaluation of the t^{th} garbled circuit in the garbled RAM constructions.
    ∗ **Recoding Step**: Recode the output encodings of the t^{th} step to obtain the input encodings of the $(t+1)^{th}$ step. This is akin to the recoding step of the garbled RAM constructions.

    The t^{th} evaluation and the recoding steps are performed using the key $\mathsf{StepKey}_t$. Moreover, they interact with the probabilistic encoding of D produced in the translation step.

    If the output of the final T^{th} step is an encoding of 0 then this is used to decrypt the encryption of μ, given as part of the ciphertext, to obtain the result μ.

We now show how to implement the above template using lattice-based techniques. The starting point to our construction is the work of [13].

Implementation of Our Template: Read-only RAMs. We implement our template using lattice-based techniques; as before, we first consider the read-only setting. We start with the high level description of the encryption procedure: let $(\mathsf{ct}_x^{(1)}, \mathsf{ct}_x^{(2)}, \mathsf{ct}_\mu)$ be the ciphertext associated with the input x and secret message μ. The first part $\mathsf{ct}_x^{(1)}$ consists of lattice-based encodings of x, initial state, initial read address and the initial read value of the RAM program. A lattice-based encoding of a bit b is computed using $\mathbf{s} \cdot (\mathbf{A} + b \cdot \mathbf{G}) + e$, where $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^{1\times n}, \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ and $e \in \mathbb{Z}_q^{1\times m}$ is drawn from a suitable error distribution; such lattice-based encodings has been studied by many works in the past [38,13]. We generate ct_μ to be $\langle \mathbf{s}, \mathbf{u}\rangle + \mu\lceil q/2\rceil + e^*$, where $\mathbf{u}, \mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^{1\times n}$ and $e^* \in \mathbb{Z}_q$ is drawn from a suitable error distribution.

We will postpone the discussion on the generation of $\mathsf{ct}_x^{(2)}$ and the attribute keys. Instead, we first mention the main ideas incorporated in the translation, evaluation and the recoding steps; this will then guide us towards identifying the attribute keys and also $\mathsf{ct}_x^{(2)}$ that will let us execute these steps.

– **Implementing the translation step**: The goal of this step is to obtain a lattice-based encoding of the database D; in particular, this encoding should be computed with respect to the same secret $\mathbf{s}$ used in the lattice-based encoding in $\mathsf{ct}_x^{(1)}$. To do this, we generate $\mathsf{ct}_x^{(2)}$ and sk_D (belonging to the attribute key) such that evaluating sk_D on $\mathsf{ct}_x^{(2)}$ yields encodings of the form $(\{\mathbf{s} \cdot (\mathbf{A}_i^* + D[i]\mathbf{G}) + e_i\})$[8]. In more detail, $\mathsf{ct}_x^{(2)}$ contains auxiliary encodings of many Boolean matrices such that given any matrix, using these auxiliary encodings, we can compute an encoding of this specific matrix. That is, $\mathsf{ct}_x^{(2)}$ will consist of encodings of the form $\mathbf{s}\cdot(\mathbf{B}_{jk\ell} + 2^\ell \mathbf{M}_{jk}) + \boldsymbol{e}_{jk\ell}$ and $\mathbf{s}\cdot\mathbf{B}'_{jk\ell} + \boldsymbol{e}'_{jk\ell}$, where $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^{1\times n}, (\mathbf{B}_{jk\ell}, \mathbf{B}'_{jk\ell}) \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ for $j \in [n], k \in [m], \ell \in [\log(q)]$ and $(\boldsymbol{e}_{jk\ell}, \boldsymbol{e}'_{jk\ell}) \in \mathbb{Z}_q^{1\times m}$ is drawn from a suitable error distribution. Here, $\mathbf{M}_{jk}$ is a matrix with 1 in the $(j,k)^{th}$ entry and zeroes everywhere else. Now, observe that using the additive homomorphic properties, we can compute an encoding that is approximately $\mathbf{s} \cdot \left(\underbrace{\sum_{jk\ell} a_{jk\ell}\mathbf{B}_{jk\ell} + (1-a_{jk\ell})\mathbf{B}'_{jk\ell} + \mathbf{A}'_i}_{\mathbf{A}_i^*} + D[i]\cdot\mathbf{G} \right)$, where $a_{jk\ell}$ denotes the ℓ^{th} bit in the bit decomposition of the $(j,k)^{th}$ entry in the matrix $\mathbf{A}'_i + D[i]\cdot\mathbf{G}$, with $\mathbf{A}'_i + D[i]\cdot\mathbf{G}$ being part of sk_D. Finally, we note that $\mathsf{ct}_x^{(2)}$ is independent of the size of the database D; this is necessary since we require that the ciphertext is of of size independent of the database length. We note that this technique of transforming encodings of bit decomposition of matrices into encodings of matrices have been studied in the past albeit for different reasons (see [19] for example).

[8] $\mathbf{A}_i^* + D[i]\mathbf{G}$ will be denoted by $\mathbf{E_i}$ in the technical sections.

An astute reader would notice that the translation step takes time proportional to the database size and thus, would violate the sub-linear decryption property! We avoid this problem by *only* translating only those database entries that are going to read during the evaluation of $P^D(x)$; note that P, D and x are public and hence, the entries that are going to be read can be correctly identified.

- **Implementing the evaluation step**: This step would be a direct adaptation of the lattice-based evaluation procedure of [13]. Given approximate encodings $(\{\mathbf{s} \cdot (\mathbf{A}_i + b_i\mathbf{G})\})$, for bits $b_1, \ldots, b_n$, and for any circuit C with a single-bit output, the evaluation procedure of [13] (we will use the notation later for this procedure as CtEval) allows for obtaining $(\{\mathbf{s} \cdot (\mathbf{A}_C + C(b_1, \ldots, b_n)\mathbf{G})e_i\})$. The matrix $\mathbf{A}_C$ is obtained by homomorphically evaluating the matrices $(\mathbf{A}_1, \ldots, \mathbf{A}_n)$ using the circuit C (later, we will refer to this procedure as $\mathsf{PubEval}$). We use the procedure of [13] to homomorphically evaluate the step circuit.
- **Implementing the recoding step**: We use lattice trapdoors [33] to convert output encodings of one time step into input encodings of the next time step. To give a flavor of how the lattice trapdoors are generated, we will take a simple case: suppose we need to translate an encoding of the read address $i \in [N]$ output by the τ^{th} evaluation step, we first sample a matrix $\mathbf{A}^{\mathsf{val},\tau}$ and then generate $\mathbf{T}_i^{\mathsf{rd},\tau}$ such that the following holds:

$$\left[\mathbf{A}^{\mathsf{rd},\tau} + i\mathbf{G} || \mathbf{A}_i^* + D[i] \cdot \mathbf{G}\right] \begin{pmatrix} \mathbf{T}_i^{\mathsf{rd},\tau} \\ \mathbf{I} \end{pmatrix} = \mathbf{A}^{\mathsf{val},\tau} + D[i] \cdot \mathbf{G} \tag{1}$$

  where $\mathbf{A}^{\mathsf{rd},\tau}$ is the matrix computed during the τ^{th} evaluation step. Recall that $\mathbf{A}_i^* + D_i \cdot \mathbf{G}$ is output by encryptor. Moreover, $\mathbf{A}^{\mathsf{val},\tau} + b\mathbf{G}$ will serve as the matrix that is used to encode the read value for the $(\tau+1)^{th}$ step. (Later we will see that in order to make the security proof work, we also additionally need an anchor matrix $\mathbf{A}$ and this will be taken into account when we generate the trapdoor matrices; see the technical sections for more details). All the lattice trapdoors generated during the t^{th} step will be part of $\mathsf{StepKey}_t$.

Implementation of Our Template: Handling Write Operations. To handle RAM programs that also write to the memory, we do the following: first, we view the database as an append-only data structure, with initial size to be N. That is, every time the program wishes to write to some memory location i, it instead appends this value to the end of the database, say at the $(N+\tau)^{th}$ location. However, this procedure introduces an additional issue. Before we describe the issue, we point out that the current lattice-based techniques disallow us from rewriting to the same location twice[9] and thus, our only other option is to use the append-only data structure.

[9] This would tantamount to obtaining two approximate encodings of the form $\mathbf{s}(\mathbf{A}_i + b_i \cdot \mathbf{G})$ and $\mathbf{s}(\mathbf{A}_i + b_i' \cdot \mathbf{G})$, where b_i is the old value and b_i' is the newly written value; assuming $b_i' \neq b_i$, having these two encodings is sufficient to break LWE.

Suppose the i^{th} location is written during the τ^{th} step. This means that the $(N+\tau)^{th}$ location would now encode the latest value corresponding to the i^{th} location. If at a later point in time, i.e., in time step $\gg \tau$, the i^{th} location needs to be read, there is no mechanism in place that prevents an adversarial evaluator to use the old encoding of the i^{th} memory location to perform an illegal evaluation.

To solve this problem, we introduce an auxiliary circuit C^{up} which keeps track of all the writes done so far and thus, for any given location i, can correctly identify the latest encoding to be used. In particular, the evaluation step from the read-only setting needs to be revised to also take into account the circuit C^{up}. That is, first the step circuit is homomorphically evaluated to obtain the location i to be read next and then the circuit C^{up} is executed to correctly identify the $(N+\tau)^{th}$ encoding that contains the value associated with location i, where τ is the time step where the i^{th} memory location was last written to. The translation and the recoding steps will be defined along the same lines as that of the read-only setting; we defer the details to the technical sections.

Careful readers may notice that the run-time of circuit C^{up} is $O(T)$, which implies the decryption time would additionally incur a multiplicative overhead of T. However, we can resolve this issue by first compiling a RAM into a last-write-aware RAM. Given a RAM P, we can compile it into another machine denoted RAM P' where the next-instruction circuit is replaced with a "next-instruction RAM" that not only emits the next address to access, but also when the next address was last written. We show such a compilation algorithm that incurs only logarithmic overhead. The idea is to maintain a balanced search tree (e.g., a 2-3 tree) that records for each logical address, when the last write was. Moreover, in this balanced search tree, each parent also keeps track of the last written times of its children. Now, when the next-instruction circuit of RAM P decides to access some logical address addr, P' would search for addr in this search tree to find out when addr was last written. Note that every search-tree operation touches constant number of tree-paths, and since the parent knows the last-written times of the children, during the search-tree operation, every memory access always knows its last-written time.

The construction for the dual setting, where the database is part of the ciphertext as against the attribute key, is obtained by a simple modification of the above template. In particular, the translation step is not necessary for the dual setting and hence, will be removed. The other steps, evaluation and recoding steps, will be defined along the same lines as the above template.

1.3 Related Work

The constructions of ABE systems has a rich literature. The seminal result of Goyal, Pandey, Sahai and Waters [42] presented the first construction of ABE for boolean formulas from bilinear DDH assumption. Since then, several prominent works achieved stronger security guarantees [46], better efficiency or design guarantees [58,9,1] and achieving stronger models of ABE for a restricted class of functions [43]. The breakthrough work of Gorbunov, Vaikuntanathan and

Wee [38] presented the first construction of ABE for all polynomial-sized circuits assuming learning with errors. Following this, several works [13,21] improved this result in terms of efficiency and also considering stronger security models [38]. In addition to [36], there are a few works that consider ABE in other models of computation. Waters [57] proposed a construction of functional encryption for regular languages and subsequently, Agarwal and Singh [4] constructed reusable garbled finite automata from LWE. Ananth and Sahai [8] construct functional encryption for Turing machines assuming sub-exponentially secure functional encryption for circuits; later this assumption was weakened to polynomially secure functional encryption by [3,31,7,44]. Deshpande et al. [23] present an alternate construction of attribute based encryption for Turing machines under the same assumptions.

2 Preliminaries

Notations. 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 $\boldsymbol{v}$ (row vector). 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 from [32]. 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 $\tau \in T$, where T is the upper running time bound

$$(\mathsf{st}^\tau, \mathsf{rd}^\tau, \mathsf{wt}^\tau, \mathsf{wb}^\tau) \leftarrow C(\mathsf{st}^{\tau-1}, \mathsf{rd}^{\tau-1}, b^\tau)$$

where we have the following:

- $\mathsf{st}^{\tau-1}$ denotes the state from the $(\tau-1)$-th step and st^{τ} denotes the state in the τ-th step.
- $\mathsf{rd}^{\tau-1}$ denotes the location to be read from, as output by the $(\tau-1)$-th step.
- b^{τ} denotes the bit at the location $\mathsf{rd}^{\tau-1}$.
- rd^{τ} denotes the location to be read from, in the next step.
- wt^{τ} denotes the location to be written into.
- wb^{τ} denotes the value to be written at τ-th step at the location wt^{τ}.

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 are interested only in RAM programs with boolean outputs.

2.2 Attribute-Based Encryption for RAMs

We state the syntax and security definition of (key-policy) public-key attribute-based encryption (ABE) for RAMs. It consists of 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,** $\mathsf{KeyGen}(\mathsf{msk}, P, D)$: On input a master secret key msk, a RAM program P and database D, it outputs a secret key $\mathsf{sk}_{P,D}$.
- **Encryption,** $\mathsf{Enc}(\mathsf{pp}, x, \mu)$: On input public parameters pp, an input x and a message μ, it outputs a ciphertext ct_x.
- **Decryption,** $\mathsf{Dec}(\mathsf{sk}_{P,D}, \mathsf{ct}_x)$: This is modeled as a RAM program. In particular, this algorithm will have random access to the binary representations of the key $\mathsf{sk}_{P,D}$ and the ciphertext ct_x. It outputs the corresponding plaintext μ if $P^D(x) = 0$; otherwise, it outputs $\perp$.

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

We define the efficiency and security properties below.

Efficiency. We define two efficiency properties associated with an ABE for RAMs scheme: namely sub-linear decryption and input-specific runtime property. The latter property implies the former.

Sub-linear Decryption: This property states that the complexity of decryption is $p(\lambda, T)$ for some fixed polynomial p, where T is the maximum runtime bound specified as part of the setup. We call this sub-linear decryption for the following reason: suppose T is *sufficiently* sublinear in $|D|$ (for instance, poly-logarithmic in $|D|$) then the decryption time is sub-linear in $|D|$. More specifically, suppose $p(\lambda, T) = \lambda^{c'} \cdot T^{c}$ and if $T << |D|^{\frac{1}{c}}$, for some constants $c', c \in \mathbb{N}$, then the decryption complexity is sub-linear in $|D|$.

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

Input-specific Runtime: This property states that the time to decrypt a ciphertext ct of (D, μ) using an attribute key of sk_P is $p(\lambda, t)$ for some fixed polynomial p, where t is the execution time of P on input database D. Note that t could be much smaller than T, where T is the maximum bound on the running time of the P.

Definition 2.3 (Input-specific Runtime) *An ABE for RAMs scheme* ABE *is said to satisfy input-specific runtime property if the following holds: for any database* D*, message* μ*, program* P*, input* x*, (i)* $(\mathsf{msk}, \mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda, 1^T)$*, (ii)* $\mathsf{sk}_{P,D} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, P, D)$*, (iii)* $\mathsf{ct}_x \leftarrow \mathsf{Enc}(\mathsf{pp}, x)$ *and, (iv) the decryption* Dec *of the functional key* $\mathsf{sk}_{P,D}$ *on input the ciphertext* ct_x *takes time* $poly(t, \lambda)$*, where* t *is the running time of* $P^D(x)$*.*

Remark 2.4 While the above properties focus on the decryption complexity, we can also correspondingly define efficiency measures for setup, key generation and encryption. Since the focus of this work is on decryption complexity, we postpone the discussion of these properties to future works.

Security. Our definition of security for ABE for RAMs will be simulation-based and in the selective setting; along the same lines as that of ABE for circuits. Informally speaking, the adversary is allowed to make multiple RAM program and database queries and submit an input query x^* such that for every program/database (P, D) queried, we have $P^D(x^*) \neq 0$. The adversary is also allowed to submit the challenge message μ. We require that the adversary cannot distinguish the two worlds: (i) when the attribute keys and ciphertext are computed as per the scheme, (ii) when the attribute keys and ciphertext can be simulated even without given μ.

Definition 2.5 *An ABE scheme* Π *for RAMs is simulation-based selectively secure if there exists* 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* $\{\mathsf{Expt}^{\mathsf{real}}_{\mathcal{A}}(1^\lambda)\}_{\lambda \in \mathbb{N}} \overset{c}{\approx} \{\mathsf{Expt}^{\mathsf{ideal}}_{\mathcal{S}}(1^\lambda)\}_{\lambda \in \mathbb{N}}$ *are computationally indistinguishable*

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

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

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

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

We call adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ admissible*, if the query (P_i, D_i) made by $\mathcal{A}_2$ satisfies $P_i^{D_i}(x^*) \neq 0$.* In the ideal experiment $\mathsf{Expt}_{\mathcal{S}}^{\mathsf{ideal}}(1^\lambda)$: $\mathcal{S}_1$ is used to generate simulated public paramters, $\mathcal{S}_2$ generates challenge ciphertext, and $\mathcal{S}_3$ answers secret key queries.

Dual Setting. We also consider the dual setting of the syntax described above, where the database is associated with ciphertext. We term this notion as *dual ABE for RAMs*. As in the above definition, the dual scheme consists of algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. The algorithms Setup and Dec are defined the same way as above. We define KeyGen and Enc as follows.

- $\mathsf{KeyGen}(\mathsf{msk}, P)$: On input a master secret key msk, a RAM program P, it outputs a secret key sk_P.
- $\mathsf{Enc}(\mathsf{pp}, D, x, \mu)$: On input public parameters pp, a database D, an input x and a message μ, it outputs a ciphertext $\mathsf{ct}_{D,x}$.

We omit the descriptions of the correctness, efficiency and security properties of dual ABE for RAMs as they are defined analogously. Due to space limit, the construction and its security proof are presented in the full version.

2.3 Learning With Errors

The learning with errors assumption was introduced by Regev [53]. This assumption has been influential in basing the security of many cryptographic primitives and most notably, fully homomorphic encryption.

Definition 2.6 (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 $\mathsf{LWE}_{n,q,\chi}$ *is to distinguish between the following pairs of distributions (e.g. as given by a sampling oracle $\mathcal{O} \in \{\mathcal{O}_{\boldsymbol{s}}, \mathcal{O}_\$\}$):*

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

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

In this work we only consider the case where the modulus $q \leq 2^n$. Recall that GapSVP_γ is the (promise) problem of distinguishing, given a basis for a lattice

and a parameter d, between the case where the lattice has a vector shorter than d, and the case where the lattice does not have any vector shorter than $\gamma \cdot d$.

There are known reductions between $\mathsf{LWE}_{n,q,\chi}$ and those problems, which allows us to appropriately choose the LWE parameters for our scheme. We summarize in the following corollary (which addresses the regime of sub-exponential modulus-to-noise ratio).

Theorem 2.7 ([53,52,47,48,17]) *For any function $B = B(n) \geq \tilde{O}(\sqrt{n})$ there exists a B-bounded distribution ensemble $\chi = \chi(n)$ over the integers s.t. for all $q = q(n)$, letting $\gamma = \tilde{O}(\sqrt{b}q/B)$, it holds that $\mathsf{LWE}_{n,q,\chi}$ is at least as hard as the quantum hardness of GapSVP$_\gamma$. Classical hardness GapSVP$_\gamma$ follows if $q(n) \geq 2n/2$ or for other values of q for $\tilde{\Omega}(\sqrt{n})$ dimensional lattices and approximation factor $q/B \cdot \mathsf{poly}(n\lceil \log q \rceil)$.*

2.4 Trapdoors and Discrete Gaussians

Let $n, q \in \mathbb{Z}$, and $m = n\lceil \log q \rceil$ and $\boldsymbol{g} = (1, 2, 4, \ldots, 2^{\lceil \log q \rceil - 1})$. The *gadget matrix* [48] $\mathbf{G}$ is defined as the diagonal concatenation of vector $\boldsymbol{g}$ n times. Formally, $\mathbf{G} = \boldsymbol{g} \otimes \mathbf{I}_n \in \mathbb{Z}_q^{n \times m}$. For any $t \in \mathbb{Z}$, the function $\mathbf{G}^{-1} : \mathbb{Z}_q^{n \times t} \to \{0,1\}^{m \times t}$ expands each entry $a \in \mathbb{Z}_q$ of the input matrix into a column of size $\lceil \log q \rceil$ consisting of the bit-representation of a. For any matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times t}$, it holds that $\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A} \bmod q$.

The (centered) discrete Gaussian distribution over $\mathbb{Z}^m$ with parameter τ, denoted $\mathcal{D}_{\mathbb{Z}^m,\tau}$, is the distribution over $\mathbb{Z}^m$ where for all $\boldsymbol{x}$, $\mathsf{Pr}[\boldsymbol{x}] \propto e^{-\pi ||\boldsymbol{x}||^2/\tau^2}$.

The following lemmas have been established in a sequence of works.

Lemma 2.8 (Trapdoor Generation [33,48]) *Let q, n, m be positive integers with $q \geq 2$ and sufficiently large $m = \Omega(n \log q)$. There exists a* ppt *algorithm $\mathsf{TrapGen}(1^n, q, m)$ that with overwhelming probability outputs a pair $(\mathbf{A} \in \mathbb{Z}_q^{n \times m}, \mathbf{T}_\mathbf{A} \in \mathbb{Z}^{m \times m})$ such that the distribution of $\mathbf{A}$ is statistically close to uniform distribution over $\mathbb{Z}_q^{n \times m}$ and $||\mathbf{T}_\mathbf{A}|| \leq O(\sqrt{n \log q})$.*

Lemma 2.9 ([33,22,2]) *Given integers $n \geq 1, q \geq 2$ there exists some $m = m(n,q) = O(n \log q)$ There are sampling algorithms as follows:*

- *There is a* ppt *algorithm $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{T}_\mathbf{A}, \boldsymbol{u}, s)$, 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 $\mathbf{T}_\mathbf{A}$ for lattice $\Lambda_q^\perp(\mathbf{A})$, a vector $\boldsymbol{u} \in \mathbb{Z}_q^n$, (3) a Gaussian parameter $s > ||\widetilde{\mathbf{T}_\mathbf{A}}|| \cdot \omega(\sqrt{\log(m + m_1)})$; then outputs a vector $\boldsymbol{r} \in \mathbb{Z}^{m+m_1}$ distributed statistically close to $\mathcal{D}_{\Lambda_q^\mathbf{u}(\mathbf{F}),s}$ where $\mathbf{F} := [\mathbf{A}||\mathbf{B}]$.*
- *There is a* ppt *algorithm $\mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{T}_\mathbf{B}, \boldsymbol{u}, s)$, 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 $\mathbf{T}_\mathbf{B}$ for lattice $\Lambda_q^\perp(\mathbf{B})$, a vector $\boldsymbol{u} \in \mathbb{Z}_q^n$, (3) a Gaussian parameter $s > ||\widetilde{\mathbf{T}_\mathbf{B}}|| \cdot s_\mathbf{R} \cdot \omega(\sqrt{\log m})$; then outputs a vector $\boldsymbol{r} \in \mathbb{Z}^{2m}$ distributed statistically close to $\mathcal{D}_{\Lambda_q^\mathbf{u}(\mathbf{F}),s}$ where $\mathbf{F} := [\mathbf{A}||\mathbf{A}\mathbf{R} + \mathbf{B}]$.*

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

Lemma 2.10 ([40]) *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 $s > ||\widetilde{\mathbf{T}_\mathbf{A}}|| \cdot \omega(\sqrt{\log(m)})$ such that for all $m \geq m^*$ and all k, we have the following two distributions are statistically close*

$$(\mathbf{A}, \mathbf{T}_\mathbf{A}, \mathbf{B}, \mathbf{U}, \mathbf{V}) \approx (\mathbf{A}, \mathbf{T}_\mathbf{A}, \mathbf{B}, \mathbf{U}', \mathbf{V}')$$

where $(\mathbf{A}, \mathbf{T}_\mathbf{A}) \leftarrow \mathsf{TrapGen}(q, n, m), (\mathbf{A}', \mathbf{B}) \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$ and $\mathbf{U} \leftarrow \mathcal{D}_{\mathbb{Z}^{2m \times k}}, \mathbf{V} = \mathbf{A} \cdot \mathbf{U}$, $\mathbf{V}' \xleftarrow{\$} \mathbb{Z}_q^{n \times k}$ and $\mathbf{U}' \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{T}_\mathbf{A}, \mathbf{V}', s)$.

We conclude with a variant of Leftover Hash Lemma [2,13]:

Lemma 2.11 *Suppose that $m > (n+1)\log q + \omega(\log n)$ and that $q > 2$ is prime. Let $\mathbf{S}$ be an $m \times k$ matrix chosen uniformly in $\{0,1\}^{m \times k}$ where $k = k(n)$ is polynomial in n. Let $\mathbf{A}$ and $\mathbf{B}$ be matrices chosen uniformly in $\mathbb{Z}_q^{n \times m}$ and $\mathbb{Z}_q^{n \times k}$ respectively. Then, for all vectors $\boldsymbol{e}$ in $\mathbb{Z}_q^m$, the distribution $(\mathbf{A}, \mathbf{AS}, \boldsymbol{e}\mathbf{S})$ is statistically close to the distribution $(\mathbf{A}, \mathbf{B}, \boldsymbol{e}\mathbf{S})$.*

2.5 Homomorphic Evaluation Procedures

The following is an abstraction of the evaluation procedure in recent LWE based FHE and ABE schemes that developed in a long sequence of works [2,48,34,5,13,40]. We use a similar formalism as in [20,16,19].

Theorem 2.12 *There exist efficient deterministic algorithms* $\mathsf{PubEval}$ *and* CtEval *such that for all $n, q, \ell \in \mathbb{N}$, and for any sequence of matrices $(\mathbf{D}_1, \ldots, \mathbf{D}_\ell) \in (\mathbb{Z}_q^{n \times n\lceil \log q \rceil})^\ell$, for any depth-$d$ Boolean circuit $f : \{0,1\}^\ell \to \{0,1\}$ and for every $\boldsymbol{x} = (x_1, \ldots, x_\ell) \in \{0,1\}^\ell$, the following properties hold:*

- $\mathsf{PubEval}(f, \{\mathbf{D}_i \in \mathbb{Z}_q^{n \times n\lceil \log q \rceil}\}_{i \in [\ell]})$*: On input matrices $\{\mathbf{D}_i\}_{i \in [d]}$ and a function $f \in \mathcal{F}$, the public evaluation algorithm outputs $\mathbf{D}_f \in \mathbb{Z}_q^{n \times n\lceil \log q \rceil}$ as the result.*
- $\mathsf{TrapEval}(f, \boldsymbol{x}, \mathbf{A} \in \mathbb{Z}_q^{n \times \lceil \log q \rceil}, \{\mathbf{R}_i\}_{i \in [\ell]})$*: the trapdoor evaluation algorithm outputs $\mathbf{R}_f$, such that*

$$\mathsf{PubEval}(f, \{\mathbf{AR}_i + x_i\mathbf{G}\}_{i \in [\ell]}) = \mathbf{AR}_f + f(\boldsymbol{x})\mathbf{G}$$

  Furthermore, we have $||\mathbf{R}_f|| \leq \delta \cdot \max_{i \in [\ell]} ||\mathbf{R}_i||$.
- $\mathsf{CtEval}(f, \boldsymbol{x}, \{\boldsymbol{c}_i\}_{i=1}^\ell)$*: On input vectors $\{\boldsymbol{c}_i\}_{i=1}^\ell \in \mathbb{Z}_q^m$, an attribute $\boldsymbol{x}$ and function f, the ciphertext evaluation algorithm outputs $\boldsymbol{c}_{f(\boldsymbol{x})} \in \mathbb{Z}_q^{n\lceil \log q \rceil}$, such that*

$$\mathsf{CtEval}(f, \boldsymbol{x}, \{\boldsymbol{s}^\mathsf{T}(\mathbf{D}_i + x_i\mathbf{G}) + \boldsymbol{e}_i\}_{i \in [\ell]}) = \boldsymbol{s}^\mathsf{T}(\mathbf{D}_f + f(\boldsymbol{x})\mathbf{G}) + \boldsymbol{e}'$$

  where $\boldsymbol{x} = (x_1, \ldots, x_\ell)$ and $\mathbf{D}_f = \mathsf{PubEval}(f, \{\mathbf{D}_i \in \mathbb{Z}_q^{n \times n\lceil \log q \rceil}\}_{i \in [\ell]})$. Furthermore, we require $||\boldsymbol{e}'|| \leq \delta \cdot \max_{i \in [\ell]} ||\boldsymbol{e}_i||$.

3 ABE for RAMs: Read-Only Case

In this part, we describe our ABE construction for read-only RAMs. A RAM program P, with random access to database D and input x, is said to be read-only if it only reads from D and never writes to it. The step circuit for read-only RAM will be defined as follows:

$$(\mathsf{st}^\tau, \mathsf{rd}^\tau) \leftarrow C(\mathsf{st}^{\tau-1}, \mathsf{rd}^{\tau-1}, b^\tau)$$

where st^τ denotes the state information at τ-th step, rd^τ denotes the read address at τ-th step and b^τ is the read value.

Parameters of the Scheme. In the description below, the parameters we use are specified in Table 1.

Parameters	Description	Setting
N	maximum database length	$\mathsf{poly}(\lambda)$
T	maximum running time	$\mathsf{poly}(\lambda)$
L_{st}	state bit-length	$\mathsf{poly}(\lambda)$
L_{rd}	address bit-length	$\log N$

Table 1: Read-only ABE Parameters

We use notation $\{\mathsf{rd}_i^\tau\}_{i\in[L_{\mathsf{rd}}]}$ to denote the bit representation of read address $\mathsf{rd}^\tau \in [N]$.

3.1 Subroutines TranslatePK, StepEvalPK and StepEvalCT

Before proceeding to our ABE construction, we first describe the syntax of three following subroutines that are used in the construction:

- $\mathsf{ListMxDB} \leftarrow \mathsf{TranslatePK}\,(\mathsf{MxPK}_{\mathsf{aux}},\ D)$: On input auxiliary encoding public key $\mathsf{MxPK}_{\mathsf{aux}}$ and database $D = \{D_i\}_{i\in[N]}$, the translation algorithm outputs encoding matrices $\mathsf{ListMxDB}$ for the database.
- $(\mathsf{StepKey}_\tau, \mathsf{ListMxPK}_\tau) \leftarrow \mathsf{StepEvalPK}\,(C, \tau, \mathsf{ListMxPK}_{\tau-1}, \mathsf{msk}, D)$: On input the step circuit C, step index τ, matrices $\mathsf{ListMxPK}_{\tau-1}$ for the $(\tau-1)$-th step and master secret key msk, the key evaluation outputs the τ-th step key $\mathsf{StepKey}_\tau$ and encoding matrices $\mathsf{ListMxPK}_\tau$ for the τ-th step.
- $\mathsf{ListVecCT}_\tau \leftarrow \mathsf{StepEvalCT}\,(C, \tau, \mathsf{ListVecCT}_{\tau-1},\ \mathsf{StepKey}_\tau, D)$: On input the step circuit C, step index τ, ciphertext $\mathsf{ListVecCT}_{\tau-1}$ of the $(\tau-1)$-th step, τ-th attribute key and databae D, the ciphertext evaluation outputs the ciphertext $\mathsf{ListVecCT}_\tau$ of the τ-th step.

In the following description, we set function $f : \{0,1\}^{L_{\mathsf{st}}} \to \mathbb{Z}$ to be $f\left(\{x_i\}_{i=1}^{L_{\mathsf{st}}}\right) = \sum x_i \cdot 2^i$. The construction of $\mathsf{StepEvalPK}$ and $\mathsf{StepEvalCT}$ with respect to step circuit C are as follows:

$\underline{\mathsf{TranslatePK}\,(\mathsf{MxPK}_{\mathsf{aux}}, D)}$: the translation algorithm does the following:

– Parse $\mathsf{MxPK}_{\mathsf{aux}}$ as $\left\{\mathbf{B}_{jk\ell}, \mathbf{B}'_{jk\ell}\right\}_{j\in[n],k\in[m],\ell\in\lceil\log q\rceil}$.
– Sample N random matrices $\{\mathbf{A}'_i\}_{i\in[N]}$ from uniform distribution over $\mathbb{Z}_q^{n\times m}$.
– For $i \in [N]$, set $\mathbf{A}_i = \mathbf{A}'_i + D[i]\mathbf{G}$.
– For $i \in [N]$, compute the encoding of i-th entry $\mathbf{E}_i$ as

$$\mathbf{E}_i = \sum_{j,k,\ell} \left(a_{jk\ell}\left(\mathbf{B}_{jk\ell} + 2^\ell \mathbf{M}_{j,k}\right) + \bar{a}_{jk\ell}\mathbf{B}'_{jk\ell}\right) = \mathbf{A}_i + \sum_{j,k,\ell} \left(a_{jk\ell}\mathbf{B}_{jk\ell} + \bar{a}_{jk\ell}\mathbf{B}'_{jk\ell}\right)$$

where $\mathbf{M}_{j,k} \in \{0,1\}^{n\times m}$ is matrix with 1 on the (j,k)-th element and 0 elsewhere, $a_{jk\ell}$ is ℓ-th bit of the bit-decomposition of (j,k)-th element a_{jk} in matrix $\mathbf{A}_i$, and $\bar{a}_{jk\ell}$ is its complement. For ease of notation, we set $\mathbf{B}_i = \sum_{j,k,\ell}\left(a_{jk\ell}\mathbf{B}_{jk\ell} + \bar{a}_{jk\ell}\mathbf{B}'_{jk\ell}\right)$.

Output matrices $\mathsf{ListMxDB} = \{(\mathbf{A}'_i, \mathbf{B}_i, \mathbf{E}_i)\}_{i\in[N]}$.

$\underline{\mathsf{StepEvalPK}\,(C, \tau, \mathsf{ListMxPK}_{\tau-1}, \mathsf{msk} = \mathbf{T}_{\mathbf{A}}, D)}$: the key evaluation algorithm does the following:

– Parse the encoding matrices $\mathsf{ListMxPK}_{\tau-1}$ as

$$\left(\mathbf{A}, \mathsf{ListMxPK}, \left\{\mathbf{A}_i^{\mathsf{st},\tau-1}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},\tau-1}\right\}_{i\in[L_{\mathsf{rd}}]}, \mathbf{A}^{\mathsf{val},\tau-1}\right)$$

– Compute $\left(\left\{\mathbf{A}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}\right) = \mathsf{PubEval}\left(\mathsf{ListMxPK}_{\tau-1}, C\right)$, where algorithm $\mathsf{PubEval}$ is defined in Theorem 2.12.
– Sample $\mathbf{A}^{\mathsf{val},\tau} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$. For $i \in [N]$, compute $\mathbf{T}_i^{\mathsf{rd},\tau}$ as

$$\mathbf{T}_i^{\mathsf{rd},\tau} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{T}_{\mathbf{A}}, \mathbf{A}^{\mathsf{rd},\tau} + i\mathbf{G}, \mathbf{A}^{\mathsf{val},\tau} - \mathbf{A}'_i - \mathbf{B}_i, s)$$

where $\mathbf{A}^{\mathsf{rd},\tau} = \mathsf{PubEval}\left(f, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}\right)$, and $\mathsf{ListMxPK} = \{\mathbf{A}'_i, \mathbf{B}_i, \mathbf{E}_i\}_{i\in[N]}$ is computed from algorithm $\mathsf{TranslatePK}\left(\mathsf{MxPK}_{\mathsf{aux}},\ D\right)$. We have that

$$\left[\mathbf{A}||\mathbf{A}^{\mathsf{rd},\tau} + i\mathbf{G}||\mathbf{A}'_i + \mathbf{B}_i + D[i]\mathbf{G}\right]\begin{pmatrix}\mathbf{T}_i^{\mathsf{rd},\tau}\\ \mathbf{I}\end{pmatrix} = \mathbf{A}^{\mathsf{val},\tau} + D[i]\mathbf{G}$$

– Set $\mathsf{StepKey}_\tau = \left\{\mathbf{T}_i^{\mathsf{rd},\tau}\right\}_{i\in[N]}$ and

$$\mathsf{ListMxPK}_\tau = \left(\mathbf{A}, \mathsf{ListMxPK}, \left\{\mathbf{A}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}, \mathbf{A}^{\mathsf{val},\tau}\right)$$

Output $(\mathsf{StepKey}_\tau, \mathsf{ListMxPK}_\tau)$.

$\underline{\mathsf{StepEvalCT}\,(C, \tau, \mathsf{ListVecCT}_{\tau-1},\ \mathsf{StepKey}_\tau, D)}$: the ciphertext evaluation algorithm does the following:

– Parse the ciphertext $\mathsf{ListVecCT}_{\tau-1}$ as

$$\left(\{\mathsf{ct}_{ijk}, \mathsf{ct}'_{ijk}\}_{\substack{i\in[n],j\in[m],\\ k\in[\log q]}}, \left\{\mathsf{ct}_i^{\mathsf{st},\tau-1}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathsf{ct}_i^{\mathsf{rd},\tau-1}\right\}_{i\in[L_{\mathsf{rd}}]}, \mathsf{ct}^{\mathsf{val},\tau-1}\right)$$

along with its associated value $\mathsf{ListST}_{\tau-1} = \left(\{\mathsf{st}^{\tau-1}\}_{i\in[L_{\mathsf{st}}]}, \{\mathsf{rd}^{\tau-1}\}_{i\in[L_{\mathsf{rd}}]}, \mathsf{val}^{\tau-1}\right)$.

– **Ciphertext evaluation**: Compute $\left(\left\{\mathsf{ct}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathsf{ct}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}\right) = \mathsf{CtEval}\left(\mathsf{ListMxPK}_{\tau-1}, \mathsf{ListST}_{\tau-1}, C\right)$, where algorithm CtEval is defined in Theorem 2.12.

– **Ciphertext translation and recoding steps**: Compute

$$\mathsf{ct}^{\mathsf{val},\tau} = \left(\widehat{\mathsf{ct}}, \mathsf{ct}^{\mathsf{rd},\tau}, \mathsf{ct}_{\mathsf{rd}^\tau}\right) \begin{pmatrix} \mathbf{T}_{\mathsf{rd}^\tau}^{\mathsf{rd},\tau} \\ \mathbf{I} \end{pmatrix}$$

where $\mathsf{ct}^{\mathsf{rd},\tau} = \mathsf{CtEval}\left(\left\{\mathsf{ct}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}, \{\mathsf{rd}_i^\tau\}_{i\in[L_{\mathsf{rd}}]}, f\right)$ and

$$\mathsf{ct}_{\mathsf{rd}^\tau} = \sum_{j,k,\ell} \left(a_{jk\ell}\mathsf{ct}_{jk\ell} + \bar{a}_{jk\ell}\mathsf{ct}'_{jk\ell}\right)$$

$a_{jk\ell}$ is ℓ-th bit of the bit-decomposition of (j,k)-th element a_{jk} in matrix $\mathbf{A}_{\mathsf{rd}^\tau} = \mathbf{A}'_{\mathsf{rd}^\tau} + D[\mathsf{rd}^\tau]\mathbf{G}$.

Output $\mathsf{ListVecCT}_\tau = \left(\{\mathsf{ct}_{ijk}, \mathsf{ct}'_{ijk}\}_{\substack{i\in[n],j\in[m],\\ k\in[\log q]}}, \left\{\mathsf{ct}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathsf{ct}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}, \mathsf{ct}^{\mathsf{val},\tau}\right)$.

We note that $\mathsf{StepEvalCT}$ *incorporates the translation, evaluation and the recoding steps described in the technical overview.*

3.2 Construction

In our construction below, we assume the initial states are all 1, the initial read address is always the first index of database.

Our read-only ABE for RAMs construction $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ can be described as follows:

Setup, $\mathsf{Setup}(1^\lambda, T)$: On input security parameter λ and time bound T, the setup algorithm computes:

– $(\mathbf{A}, \mathbf{T}_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(1^n, 1^q, m)$, the anchor matrix and its associated trapdoor.
– $\forall i \in [L_{\mathsf{st}}]$, sample $\mathbf{A}_i^{\mathsf{st},0} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, encoding matrix for the initial state.
– $\forall i \in [L_{\mathsf{rd}}]$, sample $\mathbf{A}_i^{\mathsf{rd},0} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, encoding matrix for the initial read address.
– $\forall j \in [n], k \in [m], \ell \in \lceil \log q \rceil$, sample $\left(\mathbf{B}_{jk\ell}, \mathbf{B}'_{jk\ell}\right) \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, encoding matrix for the database.
– For $i \in [\lambda]$, sample $\mathbf{A}_i^{\mathsf{val},0} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, encoding matrix for the initial read value.

– Sample $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n$, encoding vector for the plaintext.

Set $\mathsf{MxPK}_{\mathsf{aux}} = \left\{\left(\mathbf{B}_{jk\ell}, \mathbf{B}'_{jk\ell}\right)\right\}_{j\in[n],k\in[m],\ell\in\lceil\log q\rceil}$. Output $\mathsf{msk} = (\mathsf{pp}, \mathbf{T}_{\mathbf{A}})$ and

$$\mathsf{pp} = \left(\mathbf{A}, \mathsf{MxPK}_{\mathsf{aux}}, \left\{\mathbf{A}_i^{\mathsf{st},0}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},0}\right\}_{i\in[L_{\mathsf{rd}}]}, \left\{\mathbf{A}_i^{\mathsf{val},0}\right\}_{i\in[\lambda]}, \boldsymbol{u}\right)$$

Key Generation, $\mathsf{KeyGen}(\mathsf{msk}, P, D)$: On input master secret key msk, RAM program P with step circuit C and database D, it does the following:

– First compute the translation algorithm

$$\mathsf{ListMxDB} \leftarrow \mathsf{TranslatePK}\left(\mathsf{MxPK}_{\mathsf{aux}},\ D\right)$$

where $\mathsf{ListMxDB} = \{(\mathbf{A}'_i, \mathbf{B}_i, \mathbf{E}_i)\}_{i\in[N]}$. Set

$$\mathsf{ListMxPK}_0 = \left(\mathbf{A}, \mathsf{ListMxDB}, \left\{\mathbf{A}_i^{\mathsf{st},0}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},0}\right\}_{i\in[L_{\mathsf{rd}}]}, \left\{\mathbf{A}_i^{\mathsf{val},0}\right\}_{i\in[\lambda]}\right)$$

– For $\tau \in [T]$, compute

$$(\mathsf{ListMxPK}_\tau, \mathsf{StepKey}_\tau) \leftarrow \mathsf{StepEvalPK}\left(C, \tau, \mathsf{ListMxPK}_{\tau-1}, \mathbf{T}_{\mathbf{A}}, D\right)$$

– Compute $\boldsymbol{t}^{\mathsf{st},T}$ as

$$\boldsymbol{t}^{\mathsf{st},T} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{T}_{\mathbf{A}}, \mathbf{A}_1^{\mathsf{st},T}, \boldsymbol{u}, s)$$

such that

$$\left[\mathbf{A}||\mathbf{A}_1^{\mathsf{st},T}\right] \cdot \boldsymbol{t}^{\mathsf{st},T} = \boldsymbol{u}$$

Output $\mathsf{sk}_{P,D} = \left(P, D, \mathsf{ListMxDB}, \{\mathsf{StepKey}_\tau\}_{\tau\in[T]}, \boldsymbol{t}^{\mathsf{st},T}\right)$.

Encryption, $\mathsf{Enc}(\mathsf{pp}, x, \mu)$: On input public parameters pp, input $\boldsymbol{x} \in \{0,1\}^\lambda$, message μ, the encryption algorithm does the following:

– Sample vector $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$ and error vectors $\widehat{\mathbf{e}}, \mathbf{e}^*$ from Gaussian distribution $\mathcal{D}_{\mathbb{Z}^m}$.
– $\forall i \in [L_{\mathsf{st}}]$, compute $\mathsf{ct}_i^{\mathsf{st},0} = \boldsymbol{s}\left(\mathbf{A}_i^{\mathsf{st},0} + \mathbf{G}\right) + \widehat{\mathbf{e}}\mathbf{R}_i^{\mathsf{st},0}$, encoding of the initial state, where $\mathbf{R}_i^{\mathsf{st},0} \leftarrow \{0,1\}^{m\times m}$.
– $\forall i \in [L_{\mathsf{rd}}]$, compute $\mathsf{ct}_i^{\mathsf{rd},0} = \boldsymbol{s}\left(\mathbf{A}_i^{\mathsf{rd},0} + \mathsf{rd}_i^0\mathbf{G}\right) + \widehat{\mathbf{e}}\mathbf{R}_i^{\mathsf{rd},0}$, encoding of the initial read address, where $\mathbf{R}_i^{\mathsf{rd},0} \leftarrow \{0,1\}^{m\times m}$ and $\{\mathsf{rd}_i^0\}_{i\in[L_{\mathsf{rd}}]}$ is the bit representation of 1.
– For $i \in [\lambda]$, compute $\mathsf{ct}_i^{\mathsf{val},0} = \boldsymbol{s}\left(\mathbf{A}_i^{\mathsf{val},0} + \boldsymbol{x}[i]\mathbf{G}\right) + \widehat{\mathbf{e}}\mathbf{R}_i^{\mathsf{val},0}$, encoding of the initial read value, where $\mathbf{R}_i^{\mathsf{val},0} \leftarrow \{0,1\}^{m\times m}$.

– $\forall j \in [n], k \in [m], \ell \in \lceil \log q \rceil$, compute

$$\mathsf{ct}_{jk\ell} = \boldsymbol{s}\left(\mathbf{B}_{jk\ell} + 2^{\ell}\mathbf{M}_{j,k}\right) + \widehat{\mathbf{e}}\mathbf{R}_{jkl}, \quad \mathsf{ct}'_{jk\ell} = \boldsymbol{s}\mathbf{B}'_{jk\ell} + \widehat{\mathbf{e}}\mathbf{R}'_{jkl}$$

auxiliary encodings, where $\mathbf{R}_{jk\ell}, \mathbf{R}'_{jk\ell} \leftarrow \{0,1\}^{m\times m}$.
– Compute $\widehat{\mathsf{ct}} = \boldsymbol{s}\mathbf{A} + \widehat{\mathbf{e}}$ and $\mathsf{ct}^* = \boldsymbol{s}\boldsymbol{u}^{\mathsf{T}} + \mu\lceil q/2\rceil + e^*$.
– Set

$$\mathsf{ListVecCT}_0 = \left(\left\{\mathsf{ct}_i^{\mathsf{st},0}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathsf{ct}_i^{\mathsf{rd},0}\right\}_{i\in[L_{\mathsf{rd}}]}, \left\{\mathsf{ct}_i^{\mathsf{val},0}\right\}_{i\in[\lambda]} \{\mathsf{ct}_{ijk}, \mathsf{ct}'_{ijk}\}_{\substack{i\in[n],j\in[m],\\ k\in[\log q]}}\right)$$

Output ciphertext $\mathsf{ct}_{\boldsymbol{x}} = \left(\widehat{\mathsf{ct}},\ \mathsf{ct}^*,\ \mathsf{ListVecCT}_0, \boldsymbol{x}\right)$.

Decryption, $\mathsf{Dec}(\mathsf{sk}_{P,D}, \mathsf{ct}_{\boldsymbol{x}})$: On input secret key $\mathsf{sk}_{P,D}$, ciphertext $\mathsf{ct}_{\boldsymbol{x}}$, the decryption algorithm does the following:

– Output $\perp$ if $P^D(\boldsymbol{x}) \neq 0$.
– For $\tau \in [T]$, compute,

$$\mathsf{ListVecCT}_\tau \leftarrow \mathsf{StepEvalCT}\left(C, \tau, \mathsf{ListVecCT}_{\tau-1}, \mathsf{StepKey}_\tau, D\right)$$

Check if $\left|\left|\left([\widehat{\mathsf{ct}}||\mathsf{ct}_1^{\mathsf{st},T}] \cdot (\boldsymbol{t}^{\mathsf{st},T})^{\mathsf{T}}\right) - \mathsf{ct}^*\right|\right|_\infty < q/4$ and if so, output 0, otherwise output 1.

3.3 Analysis of Correctness, Efficiency and Parameters

In this part, we show that the ABE construction described above is correct (c.f. Definition 2.1), then analysis decryption time and set lattice parameters afterwards.

Lemma 3.1 *The ABE construction for read-only RAMs satisfies correctness as defined in Definition 2.1.*

Proof. Let the ciphertext be ct_x and secret key be $\mathsf{sk}_{P,D}$, such that $P^D(x) = 0$. At the τ-th step, by evaluating the ciphertext using algorithm $\mathsf{StepEvalCT}$ with respect to the step circuit, we have $\left\{\mathsf{ct}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathsf{ct}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}$ are encryption of state and read address at the τ-th step respectively. Unfolding ciphertext $\mathsf{ct}^{\mathsf{val},\tau}$ (ignoring the error terms), we obtain

$$\begin{aligned}
\mathsf{ct}^{\mathsf{val},\tau} &= \left(\widehat{\mathsf{ct}}, \mathsf{ct}^{\mathsf{rd},\tau}, \mathsf{ct}_{\mathsf{rd}^\tau}\right)\begin{pmatrix}\mathbf{T}_k^{\mathsf{rd},\tau}\\ \mathbf{I}\end{pmatrix}\\
&\approx \boldsymbol{s}\left[\mathbf{A}||\mathbf{A}^{\mathsf{rd},\tau} + \mathsf{rd}^\tau\mathbf{G}||\mathbf{E}_{\mathsf{rd}^\tau}\right]\begin{pmatrix}\mathbf{T}_i^{\mathsf{rd},\tau}\\ \mathbf{I}\end{pmatrix}\\
&\approx \mathbf{A}^{\mathsf{val},\tau} + D[\mathsf{rd}^\tau]\mathbf{G}
\end{aligned}$$

Thus, ciphertext $\mathsf{ct}^{\mathsf{val},\tau}$ encodes the read value of database at rd^τ index, which can be used in the next step evaluation.

Suppose at step T, we have $P^D = 0$, then $\mathsf{ct}_1^{\mathsf{st},t}$ encrypts state value 0. Thus,

$$\begin{aligned}\left([\widehat{\mathsf{ct}}||\mathsf{ct}_1^{\mathsf{st},t}] \cdot \boldsymbol{t}^{\mathsf{st},t}\right) - \mathsf{ct}^* &= \boldsymbol{s}\left[\mathbf{A}||\mathbf{A}_1^{\mathsf{st},\tau}\right] \cdot (\boldsymbol{t}^{\mathsf{st},t})^{\mathsf{T}} + \mathbf{e}^t - \mathsf{ct}^* \\ &= \mathbf{e}^t - \mu\lceil q/2\rceil - \mathbf{e}^*\end{aligned}$$

By setting parameters appropriately as below, our ABE scheme is correct. □

Parameters Setting. If the step circuit being evaluated has length d, then the noise in ciphertext grows in the worst case by a factor of $O(m^d)$. Thus, to support a RAM program with maximum running time T (the unit of time corresponds to one step), we set (n, m, q) as

- Lattice dimension n is an integer such that $n \geq (Td \log n)^{1/\epsilon}$, for some fixed $0 < \epsilon < 1/2$.
- Modulus q is set to be $q = 2^{n^\epsilon}$, since the noise in the ciphertexts grows by a factor of $O(m^{Td})$. Hence, we need q to be on the order of $\Omega(Bm^{Td})$, where $B = O(n)$ is the maximum magnitude of noise (from discrete Gaussian distribution) added during encryption. To ensure correctness of decryption and hardness of LWE, we set $q = 2^{n^\epsilon}$.
- Lattice column parameter m is set to be $m = \Theta(n \log q)$ to make the leftover hash lemma hold.

The parameter s used in algorithms $\mathsf{SampleLeft}$ and $\mathsf{SampleRight}$ are set as $s > \sqrt{n \log q} \cdot \omega(\sqrt{\log m})$, as required by Lemma 2.9.

For security we rely on the hardness of the LWE problem, which requires that the ratio q/B is not too large, where $B = O(n)$ is the maximum magnitude of noise (from discrete Gaussian distribution) added during encryption. In particular, the underlying problem is believed to be hard even when q/B is 2^{n^ϵ}.

Efficiency Analysis. The (space/time) complexity of our construction can be analyzed by the following aspects. The polynomial $n(\cdot, \cdot)$ denotes the lattice dimension.

- The public parameters contain $(L_{\mathsf{st}} + L_{\mathsf{rd}} + nmT)$ random $n \times m$ matrices in $\mathbb{Z}_q$, which is $\tilde{O}(n(\lambda, T)^2 \cdot n^2T^2)$ in bit complexity. The master secret key is one $m \times m$ matrix.
- The secret key for program and database pair (P, D) contains $T(N+1)$ small $m \times m$ matrices, which is $\tilde{O}(n(\lambda, T)^2 \cdot NT)$ in bit complexity.
- The ciphertext for input $\boldsymbol{x}$ contains $(L_{\mathsf{st}} + L_{\mathsf{rd}} + nmT + \lambda)$ dimension-m vectors in $\mathbb{Z}_q$, which is $\tilde{O}(n(\lambda, T) \cdot \lambda n^2T^2)$ in bit complexity.
- Decryption involves matrix-vector multiplication. The time complexity of decryption is $\tilde{O}(T)$.

Next, we would like to show the following: if a program P^D on input $\boldsymbol{x}$ takes time at most T then correspondingly, the decryption of secret key for P^D on

input an encryption of message μ associated with attribute input x takes time $p(\lambda, T)$, for a fixed polynomial p.

We analyze the time to decrypt an encryption of database x associated with message μ using a key of RAM program/database with runtime bounded by T. The essential algorithm $\mathsf{StepEvalCT}$, which may be computed T times, in decryption algorithm can be divided into two steps, as analyzed below

- **Step circuit**: The runtime of CtEval with respect to step circuit C is a polynomial in $(\lambda, L_{\mathsf{st}}, L_{\mathsf{rd}})$. Observe that L_{st} is the length of the state, which is independent of the input length, and $L_{\mathsf{rd}} = \log N$. Thus, the runtime of CtEval is upper bounded by a polynomial in $(\lambda, L_{\mathsf{st}})$.
- **Recoding part**: In this step, we compute CtEval with respect to the gadget circuit f, then the translation part, and last multiplication. This part is upper bounded by a polynomial in $(\lambda, L_{\mathsf{rd}})$.

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.

3.4 Security Proof

In this part, we show the security of our ABE for read-only RAM construction, assuming the hardness of LWE assumption. We first describe algorithms $(\mathsf{Sim.Setup}, \mathsf{Sim.Enc}, \mathsf{Sim.StepEvalPK})$ in the following:

- $\mathsf{Sim.Setup}$ produces "programmed" public parameters. That is, every pubic matrix produced as part of algorithm $\mathsf{Sim.Setup}$ has hardwired in it, a bit of the challenge ciphertext, initial state, read address, etc.
- $\mathsf{Sim.Enc}$ produces a simulated encryption of the message.
- $\mathsf{Sim.StepEvalPK}$ takes as input the $(\tau-1)$-th layer of simulated public keys $\mathsf{Sim.ListMxPK}_{\tau-1}$ and produces the τ-th layer of simulated public keys $\mathsf{Sim.ListMxPK}_{\tau}$ and τ-th layer of step keys $\mathsf{Sim.StepKey}_{\tau}$.

These simulated algorithms can be constructed as follows:

$\underline{\mathsf{Sim.Setup}(1^\lambda, \boldsymbol{x}^*)}$: On input the challenge input $\boldsymbol{x}^*$, the simulated setup algorithm does:

- Compute $(\mathbf{A}, \mathbf{T}_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(1^n, 1^q, m)$ and sample $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n$.
- $\forall i \in [L_{\mathsf{st}}]$, set $\mathbf{A}_i^{\mathsf{st},0} = \mathbf{A}\mathbf{R}_i^{\mathsf{st},0} - \mathbf{G}$, where $\mathbf{R}_i^{\mathsf{st},0} \leftarrow \{0,1\}^{m\times m}$.
- $\forall i \in [L_{\mathsf{rd}}]$, set $\mathbf{A}_i^{\mathsf{rd},0} = \mathbf{A}\mathbf{R}_i^{\mathsf{rd},0} - \mathbf{G}$, where $\mathbf{R}_i^{\mathsf{rd},0} \leftarrow \{0,1\}^{m\times m}$.
- $\forall j \in [n], k \in [m], \ell \in \lceil \log q \rceil$, set

$$\mathbf{B}_{jk\ell} = \mathbf{A}\mathbf{R}_{jk\ell} - 2^\ell \mathbf{M}_{j,k}, \quad \mathbf{B}'_{jk\ell} = \mathbf{A}\mathbf{R}'_{jk\ell}$$

  where $(\mathbf{R}_{jk\ell}, \mathbf{R}'_{jk\ell}) \leftarrow \{0,1\}^{m\times m}$.

– $\forall i \in [\lambda]$, set $\mathbf{A}^{\mathsf{val},0} = \mathbf{A}\mathbf{R}_i^{\mathsf{val},0} - \boldsymbol{x}^*[i]\mathbf{G}$, where $\mathbf{R}_i^{\mathsf{val},0} \leftarrow \{0,1\}^{m\times m}$.

Let $\mathsf{Sim.MxPK}_{\mathsf{aux}} = \left(\mathbf{B}_{jk\ell}, \mathbf{B}'_{jk\ell}\right)_{j\in[n],k\in[m],\ell\in\lceil\log q\rceil}$, and denote trapdoor matrix for initial step as

$$\mathsf{ListMxTD}_0 = \left(\left\{\mathbf{R}_i^{\mathsf{st},0}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{R}_i^{\mathsf{rd},0}\right\}_{i\in[L_{\mathsf{rd}}]}, \mathbf{R}^{\mathsf{val},0}\right)$$

Output $\mathsf{msk} = (\mathsf{pp}, \mathbf{T}_{\mathbf{A}})$ and

$$\mathsf{Sim.pp} = \left(\mathbf{A}, \mathsf{Sim.MxPK}_{\mathsf{aux}}, \left\{\mathbf{A}_i^{\mathsf{st},0}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},0}\right\}_{i\in[L_{\mathsf{rd}}]}, \left\{\mathbf{A}_i^{\mathsf{val},0}\right\}_{i\in[\lambda]}, \boldsymbol{u}\right)$$

$\underline{\mathsf{Sim.Enc}(\mathsf{Sim.pp}, \boldsymbol{x}^*, 1^{|\mu|}, (\mathbf{A}, \boldsymbol{u}), (\boldsymbol{b}, b'))}$: On input simulated public parameters $\mathsf{Sim.pp}$, challenge input $\boldsymbol{x}^*$ and message length $|\mu|$ and LWE instance $((\mathbf{A}, \boldsymbol{u}), (\boldsymbol{b}, b'))$, the simulated encryption algorithm does

– $\forall i \in [L_{\mathsf{st},0}]$, compute $\mathsf{ct}_i^{\mathsf{st},0} = \boldsymbol{b}\mathbf{R}_i^{\mathsf{st},0}$, where $\mathbf{R}_i^{\mathsf{st},0}$ is generated in $\mathsf{Sim.Setup}$.
– $\forall i \in [L_{\mathsf{rd},0}]$, compute $\mathsf{ct}_i^{\mathsf{rd},0} = \boldsymbol{b}\mathbf{R}_i^{\mathsf{rd},0}$, where $\mathbf{R}_i^{\mathsf{rd},0}$ is generated in $\mathsf{Sim.Setup}$.
– $\forall i \in [\lambda]$, compute $\mathsf{ct}_i^{\mathsf{val},0} = \boldsymbol{b}\mathbf{R}_i^{\mathsf{val},0}$, where $\mathbf{R}_i^{\mathsf{val},0}$ is generated in $\mathsf{Sim.Setup}$.
– $\forall j \in [n], k \in [m], \ell \in \lceil\log q\rceil$, compute $\mathsf{ct}_{jk\ell} = \boldsymbol{b}\mathbf{R}_{jk\ell}$, $\mathsf{ct}'_{jk\ell} = \boldsymbol{b}\mathbf{R}'_{jk\ell}$ where $\left(\mathbf{R}_{jk\ell}, \mathbf{R}'_{jk\ell}\right)$ is generated in $\mathsf{Sim.Setup}$.
– Set $\widehat{\mathsf{ct}} = \boldsymbol{b}$ and $\mathsf{ct}^* = b'$.
– Define $\mathsf{ListVecCT}_0$ in the same way as the real scheme.

Output challenge ciphertext $\mathsf{ct}_{x^*} = \left(\widehat{\mathsf{ct}}, \mathsf{ct}^*, \mathsf{ListVecCT}_0, x^*\right)$.

$\underline{\mathsf{Sim.StepEvalPK}(C, \tau, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{Sim.pp}, D)}$: On input the step circuit C of program P satisfying $P^D(\boldsymbol{x}^*) = 1$, step index τ, simulated $(\tau-1)$-th layer of simulated public keys $\mathsf{Sim.ListMxPK}_{\tau-1}$, simulated public parameters $\mathsf{Sim.pp}$ and database query D, if $\tau = 1$, compute the translation algorithm

$$\mathsf{ListMxDB} \leftarrow \mathsf{TranslatePK}\left(\mathsf{MxPK}_{\mathsf{aux}}, D\right)$$

where $\mathsf{ListMxDB} = \{(\mathbf{A}'_i, \mathbf{B}_i, \mathbf{E}_i)\}_{i\in[N]}$. Set

$$\mathsf{ListMxPK}_0 = \left(\mathbf{A}, \mathsf{ListMxDB}, \left\{\mathbf{A}_i^{\mathsf{st},0}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},0}\right\}_{i\in[L_{\mathsf{rd}}]}, \left\{\mathbf{A}_i^{\mathsf{val},0}\right\}_{i\in[\lambda]}\right)$$

Otherwise, it does:

– Compute $\left(\left\{\mathbf{A}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}\right) = \mathsf{PubEval}\left(\mathsf{Sim.ListMxPK}_{\tau-1}, C\right)$, and then $\mathbf{A}^{\mathsf{rd},\tau} = \mathsf{PubEval}\left(f, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}\right)$, where $\mathbf{A}^{\mathsf{rd},\tau}$ encodes the actual read address rd^τ of $P^D(x^*)$ at τ-th step.

– Sample $\mathbf{T}^{\mathsf{rd},\tau}_{\mathsf{rd}^\tau} = \left(\mathbf{T}^{\mathsf{rd},\tau}_{\mathsf{rd}^\tau,0}, \mathbf{T}^{\mathsf{rd},\tau}_{\mathsf{rd}_\tau,1}\right) \leftarrow \mathcal{D}_{Z^{m\times m}}$ and set $\mathbf{A}^{\mathsf{val},\tau} = \mathbf{A}\left(\mathbf{T}^{\mathsf{rd},\tau}_{\mathsf{rd}^\tau,0} + \mathbf{R}^{\mathsf{rd},\tau}\mathbf{T}^{\mathsf{rd},\tau}_{\mathsf{rd}_\tau,1} + \mathbf{R}_i\right)$, where $\mathbf{R}^{\mathsf{rd},\tau} = \mathsf{TrapEval}\left(f \circ C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{ListMxTD}_{\tau-1}\right)$ and $\mathbf{R}_i = \sum_{jk\ell}\left(d_{jk\ell}\mathbf{R}_{jk\ell} + \bar{d}_{jk\ell}\mathbf{R}'_{jk\ell}\right)$ and algorithm $\mathsf{TrapEval}$ is defined in Theorem 2.12.
– For $i \in [N] - \{\mathsf{rd}^\tau\}$, compute $\mathbf{T}^{\mathsf{rd},\tau}_i$ as

$$\mathbf{T}^{\mathsf{rd},\tau}_i \leftarrow \mathsf{SampleRight}\left(\mathbf{A}, (i-\mathsf{rd}^\tau)\mathbf{G}, \mathbf{R}^{\mathsf{rd},\tau}, \mathbf{T}_{\mathbf{G}}, \mathbf{A}^{\mathsf{val},\tau} - \mathbf{A}'_i - \mathbf{B}_i, s\right)$$

such that

$$\left[\mathbf{A}||\mathbf{A}\mathbf{R}^{\mathsf{rd},\tau} + (i-\mathsf{rd}^\tau)\mathbf{G}||\mathbf{E}_i\right]\begin{pmatrix}\mathbf{T}^{\mathsf{rd},\tau}_i\\ \mathbf{I}\end{pmatrix} = \mathbf{A}^{\mathsf{val},\tau} + D[i]\mathbf{G}$$

where $\{\mathbf{A}'_i, \mathbf{B}_i, \mathbf{E}_i\}$ is computed by algorithm $\mathsf{TranslatePK}$.
– As $P^D(\boldsymbol{x}^*) = 1$, so $\mathbf{A}^{\mathsf{st},T}_1$ is the encoding of 1, i.e. $\mathbf{A}^{\mathsf{st},T}_1 = \mathbf{A}\mathbf{R}^{\mathsf{st},T}_1 - \mathbf{G}$. Compute $\boldsymbol{t}^{\mathsf{st},T}$ as

$$\boldsymbol{t}^{\mathsf{st},T} \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{R}^{\mathsf{st},T}_1, \mathbf{T}_{\mathbf{G}}, \boldsymbol{u}, s)$$

such that

$$\left[\mathbf{A}||\mathbf{A}\mathbf{R}^{\mathsf{st},T}_1 - \mathbf{G}\right]\cdot \boldsymbol{t}^{\mathsf{st},T} = \boldsymbol{u}$$

Set $\mathsf{Sim.StepKey}_\tau = \left\{\mathbf{T}^{\mathsf{rd},\tau}_i\right\}_{i\in[N]}$. Output $(\mathsf{Sim.StepKey}_\tau, \mathsf{Sim.ListMxPK}_\tau)$.

Theorem 3.2 *Assuming the hardness of LWE assumption (with parameters as specified above), our ABE construction is secure (c.f. Definition 2.5).*

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

Hybrid Hyb_1: This corresponds to the real experiment:
– $\mathcal{A}$ specifies challenge attribute input $\boldsymbol{x}^*$ 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 $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pp}, \boldsymbol{x}^*, \mu)$. It sends ct^* and pp to $\mathcal{A}$.
– For $\gamma \in [Q]$, adversary $\mathcal{A}$ specifies the programs/database (P_γ, D_γ) such that $P^{D_\gamma}_\gamma(\boldsymbol{x}^*) = 1$. Challenger generates the attribute keys for (P_γ, D_γ), for $\gamma \in [Q]$, $\mathsf{sk}_{P_\gamma,D_\gamma} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, P_\gamma, D_\gamma)$.
– Let b be the output of adversary. Output b.

Hybrid Hyb_2: Hyb_2 is the same as Hyb_1 except that it uses $\mathsf{Sim.Setup}(1^\lambda, \boldsymbol{x}^*)$ to generate $\mathsf{Sim.pp}$.

Hybrid $\{\mathsf{Hyb}_{3,i,j}\}_{i\in[Q],j\in[T]}$: Simply put, in hybrid $\mathsf{Hyb}_{3,i,j}$, for $\gamma < i$, the secret key for query (P_γ, D_γ) is simulated. For query (P_i, D_i), upto the j-th step, the step keys are simulated. For $\tau > j$, the step keys are normally generated. For query (P_γ, D_γ), where $\gamma > i$, the secret key is normally generated. We describe it in details below:
– Adversary specifies challenge attribute input $\boldsymbol{x}^*$ and message μ.

– Challenger computes $\mathsf{Sim.Setup}(1^\lambda)$ to obtain the public parameters pp and secret key msk. Then challenger generates the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pp}, x^*, \mu)$. It sends ct^* and pp to $\mathcal{A}$.
– For $\gamma \in [Q]$, adversary $\mathcal{A}$ specifies the program/database (P_γ, D_γ) such that $P_\gamma^{D_\gamma}(x^*) = 1$. Challenger generates the secret key $\mathsf{sk}_{P_\gamma, D_\gamma}$ as

$$\mathsf{sk}_{P_\gamma, D_\gamma} = (P_\gamma, D_\gamma, \{\mathsf{StepKey}_\tau\}_{\tau \in [T]}, \boldsymbol{t}^{\mathsf{st},T})$$

- For $\gamma < i$, answer secret key query P_γ as
  1. For every $\tau \in [T]$, compute

$$(\mathsf{Sim.ListMxPK}_\tau, \mathsf{Sim.StepKey}_\tau) \leftarrow \mathsf{Sim.StepEvalPK}(C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{Sim.pp})$$

  2. Set $\mathsf{sk}_\gamma = (\{\mathsf{Sim.StepKey}_\tau\}_{\tau \in [T]})$.
- For $\gamma = i$, answer secret key query (P_i, D_i) as
  1. For $\tau < j$, generate

$$(\mathsf{Sim.ListMxPK}_\tau, \mathsf{Sim.StepKey}_\tau) \leftarrow \mathsf{Sim.StepEvalPK}(C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{Sim.pp})$$

  2. For $\tau \geq j$, generate

$$(\mathsf{ListMxPK}_\tau, \mathsf{StepKey}_\tau) \leftarrow \mathsf{StepEvalPK}(C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{Sim.pp})$$

  Set $\mathsf{sk}_i = (\{\mathsf{Sim.StepKey}_\tau\}_{\tau<i}, \{\mathsf{StepKey}_\tau\}_{\tau \geq i})$.
- For $\gamma > i$, answer secret key query (P_γ, D_γ) as
  1. For every $\tau \in [T]$, generate

$$(\mathsf{ListMxPK}_\tau, \mathsf{StepKey}_\tau) \leftarrow \mathsf{StepEvalPK}(C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{Sim.pp})$$

  2. Set $\mathsf{sk}_\gamma = (\{\mathsf{StepKey}_\tau\}_{\tau \in [T]})$.

Hybrid Hyb_4: Hyb_4 is the same as $\mathsf{Hyb}_{3,Q,T}$ except that the anchor public key $\mathbf{A}$ is sampled randomly from $\mathbb{Z}_q^{n \times m}$. In $\mathsf{Hyb}_{3,Q,T}$ the secret keys for all queries are simulated without using $\mathsf{msk} = \mathbf{T}_\mathbf{A}$.

Hybrid Hyb_5: Hyb_5 is the same as Hyb_4 except that it uses algorithm $\mathsf{Sim.Enc}$ to generate the challenge ciphertext.

Due to the space limit, we show the indistinguishability proof between adjacent hybrids in the full version. □

Acknowledgements.

We would like to thank the anonymous reviewers of Asiacrypt 2019 and Jiaxin Pan for helpful suggestions to improve the presentation of the paper. Xiong Fan is supported in part by IBM under Agreement 4915013672 and NSF Award CNS-1561209. Elaine Shi is supported by NSF Award CNS-1617676.

References

1. Shashank Agrawal and Melissa Chase. A study of pair encodings: Predicate encryption in prime order groups. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 259–288. Springer, Heidelberg, January 2016.
2. Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 553–572. Springer, Heidelberg, May / June 2010.
3. Shweta Agrawal and Monosij Maitra. FE and iO for turing machines from minimal assumptions. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 473–512. Springer, Heidelberg, November 2018.
4. Shweta Agrawal and Ishaan Preet Singh. Reusable garbled deterministic finite automata from learning with errors. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 80. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
5. 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.
6. Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation from functional encryption for simple functions. Cryptology ePrint Archive, Report 2015/730, 2015. http://eprint.iacr.org/2015/730.
7. Prabhanjan Ananth and Alex Lombardi. Succinct garbling schemes from functional encryption through a local simulation paradigm. In *Theory of Cryptography Conference*, pages 455–472. Springer, 2018.
8. Prabhanjan Ananth and Amit Sahai. Functional encryption for turing machines. In *Theory of Cryptography Conference*, pages 125–153. Springer, 2016.
9. Nuttapong Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 557–577. Springer, Heidelberg, May 2014.
10. 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.
11. Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In Madhu Sudan, editor, *ITCS 2016*, pages 345–356. ACM, January 2016.
12. Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology—CRYPTO 2001*, pages 213–229. Springer, 2001.
13. 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 Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Heidelberg, May 2014.
14. Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. *Theory of Cryptography*, pages 253–273, 2011.
15. 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.
16. Zvika Brakerski, David Cash, Rotem Tsabary, and Hoeteck Wee. Targeted homomorphic attribute-based encryption. In Martin Hirt and Adam D. Smith, editors,

TCC 2016-B, Part II, volume 9986 of *LNCS*, pages 330–360. Springer, Heidelberg, October / November 2016.
17. Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 575–584. ACM Press, June 2013.
18. 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.
19. Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained PRFs (and more) from LWE. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 264–302. Springer, Heidelberg, November 2017.
20. Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 1–30. Springer, Heidelberg, March 2015.
21. Zvika Brakerski and Vinod Vaikuntanathan. Circuit-abe from lwe: unbounded attributes and semi-adaptive security. In *Annual Cryptology Conference*, pages 363–384. Springer, 2016.
22. David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 523–552. Springer, Heidelberg, May / June 2010.
23. 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.
24. Nico Döttling and Sanjam Garg. Identity-based encryption from the diffie-hellman assumption. In *Annual International Cryptology Conference*, pages 537–569. Springer, 2017.
25. 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.
26. 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.
27. 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.
28. Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure attribute based encryption from multilinear maps. Cryptology ePrint Archive, Report 2014/622, 2014. http://eprint.iacr.org/2014/622.
29. Sanjam Garg, Steve Lu, and Rafail Ostrovsky. Black-box garbled ram. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 210–229. IEEE, 2015.
30. Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. Garbled ram from one-way functions. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 449–458. ACM, 2015.
31. Sanjam Garg and Akshayaram Srinivasan. A simple construction of io for turing machines. In *Theory of Cryptography Conference*, pages 425–454. Springer, 2018.

32. 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.
33. 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.
34. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, August 2013.
35. 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.
36. 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.
37. Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 545–554. ACM Press, June 2013.
38. Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. *Journal of the ACM (JACM)*, 62(6):45, 2015.
39. Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from lwe. In *Annual Cryptology Conference*, pages 503–523. Springer, 2015.
40. Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 469–477. ACM Press, June 2015.
41. Vipul Goyal, Abhishek Jain, Omkant Pandey, and Amit Sahai. Bounded ciphertext policy attribute based encryption. *Automata, languages and programming*, pages 579–591, 2008.
42. 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.
43. 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.
44. Fuyuki Kitagawa, Ryo Nishimaki, Keisuke Tanaka, and Takashi Yamakawa. Adaptively secure and succinct functional encryption: Improving security and efficiency, simultaneously. Cryptology ePrint Archive, Report 2018/974, 2018. https://eprint.iacr.org/2018/974.
45. 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.
46. 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.

47. Daniele Micciancio and Petros Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 465–484. Springer, Heidelberg, August 2011.
48. 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.
49. Adam O'Neill. Definitional issues in functional encryption. *IACR Cryptology ePrint Archive*, 2010:556, 2010.
50. 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.
51. 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.
52. Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 333–342. ACM Press, May / June 2009.
53. 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.
54. Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Eurocrypt*, volume 3494, pages 457–473. Springer, 2005.
55. Brent Waters. Efficient identity-based encryption without random oracles. In *Eurocrypt*, volume 3494, pages 114–127. Springer, 2005.
56. Brent Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In *Crypto*, volume 5677, pages 619–636. Springer, 2009.
57. Brent Waters. Functional encryption for regular languages. In *CRYPTO*, volume 7417, pages 218–235. Springer, 2012.
58. Hoeteck Wee. Dual system encryption via predicate encodings. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 616–637. Springer, Heidelberg, February 2014.