Implementation and Evaluation of a Lattice-Based Key-Policy ABE Scheme

Wei Dai*, Yarkın Doröz*, Yuriy Polyakov†, Kurt Rohloff†, Hadi Sajjadpour†, Erkay Savaş†‡ and Berk Sunar*
* Worcester Polytechnic Institute, Worcester, MA, USA 01609
Email: {wdai, ydoroz, sunar}@wpi.edu
† NJIT Cybersecurity Research Center
New Jersey Institute of Technology, Newark, NJ, USA 07102
Email: {polyakov,rohloff,ss2959,savas}@njit.edu
‡ Sabancı University, Tuzla, Istanbul, Turkey 34956
Email: erkays@sabanciuniv.edu

***Abstract*—In this paper, we report on our implementation of a lattice-based Key-Policy Attribute-Based Encryption (KP-ABE) scheme, which uses short secret keys. The particular KP-ABE scheme can be used directly for Attribute-Based Access Control (ABAC) applications, as well as a building block in more involved applications and cryptographic schemes such as audit log encryption, targeted broadcast encryption, functional encryption, and program obfuscation. We adapt a recently proposed KP-ABE scheme based on the Learning With Errors (LWE) problem to a more efficient scheme based on the Ring Learning With Errors (RLWE) problem, and demonstrate an implementation that can be used in practical applications. Our state-of-the-art GPU implementation shows that the homomorphic public key and ciphertext evaluation operations, which dominate the execution time of the KP-ABE scheme, can be performed in a reasonably short amount of time. Our practicality results also hold when scaled to a relatively large number of attributes. To the best of our knowledge, this is the first KP-ABE implementation that supports both ciphertext and public key homomorphism and the only experimental practicality results reported in the literature.**

***Index Terms*—lattice-based cryptography, attribute-based encryption, GPU computing, RLWE**

I. Introduction

Attribute-Based Encryption (ABE) is a public key cryptographic scheme that enables the decryption of a ciphertext by a user only if a certain access policy defined over attributes is satisfied. ABE is introduced by Sahai and Waters in [1] as a generalization of identity based encryption (IBE) [2]. The concept of ABE is improved to incorporate fine-grain access control in [3], [4]. By enforcing more general access policies, ABE schemes are becoming a source of interest in academia and industry as ABE restricts access to sensitive data without relying on a central access control system. Besides supporting access control applications, ABE can be used to implement other interesting applications such as audit log encryption and targeted/broadcast encryption [3].

ABE has two main flavors of constructions: Ciphertext-Policy ABE (CP-ABE) and Key-Policy ABE (KP-ABE). CP-ABE has been more widely studied and implemented in the literature [4]–[8]. In CP-ABE, an access policy is incorporated into a ciphertext, and a secret decryption key is generated for a subset of attributes held by a user. If a user holds attributes that satisfy the access policy, she can decrypt ciphertext encrypted under that policy. In this model, access policy needs to be known before the encryption and secret keys are bound to a subset of attributes. On the other hand, KP-ABE [1], [3], [9], allows a message to be encrypted using the attribute values as public keys, and a secret key is generated for a particular access policy defined over the set of attributes, potentially after encryption. (See Figure 1 for a representation of this workflow.) Importantly, the access policy may not be known at the time of encryption and can be defined later.

Two classes of cryptographic primitives are generally used in the construction of ABE schemes: bilinear pairings and lattices. The majority of ABE schemes are based on bilinear pairings [2], including [3], [6], [10]–[12]. Software implementations of pairing-based ABE constructions are reported in [4], [8], [13]. Most prior bilinear pairing implementations support CP-ABE schemes. Other ABE schemes are based on lattices with hardness assumptions of Learning With Errors (LWE), Short Integer Solution (SIS), or inhomogenous SIS [14]–[17]. Several lattice-based ABE schemes are known, including a CP-ABE scheme in [18] and a KP-ABE scheme in [19].

A key concept in the KP-ABE scheme from [19] is a *key homomorphism* property which supports homomorphic computations over public keys associated with attributes. By leveraging this property, ciphertexts and public keys can be homomorphically evaluated over a circuit determined by an access policy (represented as a *policy circuit*) to compute a new (and compressed) public key and ciphertexts which can only be decrypted under that policy. In Figure 1, a ciphertext C is homomorphically evaluated over the policy circuit f to obtain a new ciphertext under f (C_f) that can be decrypted only by using the policy secret key α_f.

In this work, we present a Ring Learning With Errors (RLWE) variant of the KP-ABE scheme proposed in [19], which leads to practical implementation. The particular construction has several novel properties, which constitutes our main motivation of its selection for implementation. First, the key homomorphism property allows public keys and ciphertexts to be evaluated over an access policy. Second, the scheme's complexity depends on its depth rather than

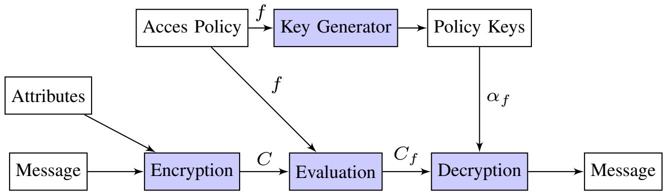

Fig. 1: Block diagram of KP-ABE Scheme

on the size of the policy circuit, which is beneficial to its efficient implementation and ultimately its usability in real world applications. Third, the scheme supports different applications such as garbled circuits as suggested in [19], functional encryption [20] and token-based program obfuscation [21]. Last, the construction is considered post-quantum as it is based on lattice problems that are believed to be secure against quantum computer attacks.

Our Contribution To the best of our knowledge, we provide the first implementation of the KP-ABE scheme proposed in [19]. An efficient implementation of this scheme is a technical challenge due to the difficulty of implementing the powerful key and ciphertext homomorphism properties. The primary goal of this paper is therefore to demonstrate that the KP-ABE construction can be practical by leveraging acceleration techniques at both algorithmic and implementation levels.

To this end, we first propose an RLWE-based construction of KP-ABE, which is originally constructed using LWE problem in [19]. The RLWE construction provides two main advantages: 1) it is more efficient and 2) supports the encryption of multiple bits in a single ciphertext. Our KP-ABE scheme enjoys the RLWE hardness assumptions.

Second, we design and implement parallel algorithms tailored to take full advantage of Graphics Processing Unit (GPU) processors. We particularly focus on using GPU algorithms and techniques to accelerate the ABE encryption and homomorphic evaluation operations because these operations are computational bottlenecks. The other ABE operations are either already fast (i.e., decryption) or performed occasionally (i.e., private key generation or setup). We compare our experimental GPU results for bottleneck ring operations with prior results in the literature. Our comparison shows that our GPU implementation of ring multiplication, which dominates all ABE operations, outperforms all other similar GPU implementations. Our results clearly confirm our claim that a sophisticated KP-ABE scheme such as the one in [19] can, indeed, be made practical.

We also quantify the noise growth in the ciphertext, which can be a factor that limits the feasibility of the scheme. We observe that the noise grows faster than the estimates based on Central Limit Theorem because the main exponential term in the correctness constraint is not zero-centered. To reduce the noise growth, we propose a new technique based on balanced non-adjacent form (NAF) of integers to transform the main exponential term to zero-centered representation.

II. Related Work

The original KP-ABE construction [19] is based on the LWE problem, which is believed to be as hard as worst-case lattice problems such as the shortest vector problem (SVP) and the shortest independent vector problem (SIVP) [22]. The constructions based on ideal lattices, referred to as RLWE-based constructions, are shown to be more efficient than LWE-based constructions, in terms of both execution time and memory requirement [23]. Also, the RLWE problems are proved to be hard due to a quantum reduction from worst-case approximate SVP on ideal lattices to the search version of RLWE [24], [25]. It is also proved that the RLWE distribution is pseudorandom if the RLWE search problem is hard [24].

KP-ABE schemes are generally much more complicated to implement than CP-ABE schemes as the latter requires no homomorphic computation over public keys or ciphertexts. As a recent CP-ABE scheme based on the LWE problem [18] uses a similar construction to [19], we can extend our construction to implement a CP-ABE scheme. The key and ciphertext homomorphisms efficiently implemented in our scheme are novel properties that make the KP-ABE scheme suitable for a much more diverse set of applications.

An important advantage of our construction based on the original scheme [19] is that the secret key size is much smaller than those of similar constructions, in which the secret key size is proportional to the size of the policy circuit [26], [27]. Also, constructions based on multi-linear maps [28] such as [27] and the second scheme in [19] are beyond the scope of our paper.

To the best of our knowledge, all implementations reported in the literature [4], [8], [13], [29] are CP-ABE constructions based on bilinear pairings [2]. Since we implement a KP-ABE construction based on lattice primitives, a direct and fair comparison is not possible.

Like many other ABE schemes in the literature, our construction utilizes the concept of lattice trapdoors introduced in [14]. In trapdoor based ABE schemes, a secret key corresponding to an access policy (KP-ABE) or a subset of attributes (CP-ABE) is generated by a trusted third party known as a private key generator (PKG) that is in possession of trapdoor information. Some works in the literature are devoted to improve the efficiency of trapdoor generation [30], [31]. Applications of lattice trapdoors such as signature schemes are proved to be practical [23] as the timing results of actual software implementations are highly promising. In our construction, we utilize the lattice trapdoor sampling optimizations recently developed in [32], [33].

As shown in [34], the secret key for a function f that represents the access policy in KP-ABE corresponds to the garbled circuit for f and the ciphertext encrypting attribute vector $\mathbf{x}$ similarly corresponds to the garbled input in the *reusable* garbled circuit scheme. In conjunction with obfuscation schemes such as the one in [21], KP-ABE can be used as a building block to implement token-based obfuscation. Considered as a generalization of ABE, predicate-based encryption (PBE) schemes [35] benefit from efficient implementation of KP-ABE.

III. Preliminaries

In this section, we provide mathematical background and preliminaries that are necessary to follow the discussions in the paper. We also present algorithms and techniques behind our implementation and briefly introduce the basics of CUDA-enabled GPUs.

A. Mathematical Notations And Definitions

Let $\mathcal{R} = \mathbb{Z}[x]/(x^n + 1)$ be a cyclotomic ring, where the ring elements are polynomials of at most degree $n-1$ with integer coefficients. Here, the ring dimension n is a power of 2 for efficient ring arithmetic. And let also $\mathcal{R}_q = \mathcal{R}/qR$ be a ring where the arithmetic operations on polynomial coefficients are performed modulo q and coefficients are represented as integers in the interval $\left(-\left\lfloor \frac{q}{2} \right\rfloor, \left\lfloor \frac{q}{2} \right\rfloor\right]$. $\mathcal{R}_2$ is the ring of binary polynomials. Also, $\mathcal{R}_q^{1\times m}$, $\mathcal{R}_q^{m\times 1}$, and $\mathcal{R}_q^{m\times m}$ stand for row vector, column vector and matrix of ring elements in $\mathcal{R}_q$, respectively, for an integer $m > 1$. Finally $k = \lceil \log_2 q \rceil$ stands for the number of bits in q.

Let $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ be integers in the interval $[0,\ q-1]$. Then, $\mathbb{F}_q = \mathbb{Z}/q\mathbb{Z}$, if q is a prime power, forms a finite field. Throughout the paper, we use boldfaced symbols to denote vectors and matrices, e.g. $\mathbf{a} = (a_0,\ a_1,\ \ldots,\ a_{n-1})$, where $a_i \in \mathbb{Z}_q$ or $a_i \in \mathcal{R}_q$, while small case letters usually denote single elements.

Let $[\cdot]_q$ be modulo q reduction on an integer or on coefficients of a vector, that is, $[a]_q = a \mod q \in \mathbb{Z}_q$ or $[\mathbf{a}]_q = \mathbf{a} \mod q \in \mathbb{Z}_q^n$. A polynomial in $\mathcal{R}_q$ can be represented as a vector in $\mathbb{Z}_q^n$ with its coefficients lifted to the interval $[0, q-1]$: if $a < 0$, q is added to a.

We also denote the infinity norm of a polynomial or a vector as $||\cdot||_\infty$ (only $||\cdot||$ for simplicity, i.e. the largest absolute value of coefficients). When the infinity norm of a polynomial is below a relatively small upper bound, it is referred to as a *short* polynomial. Also a vector of short polynomials is called a short vector.

$D_{\Lambda,\mathbf{c},\sigma}$ denotes n-th dimensional discrete Gaussian distribution over a lattice $\Lambda \subset \mathbb{R}^n$, where $\mathbf{c} \in \mathbb{R}^n$ is the center and $\sigma \in \mathbb{R}$ is the distribution parameter. Lattice sampling operation $\mathbf{x} \leftarrow D_{\Lambda,\mathbf{c},\sigma}$ assigns the probability $\rho(\mathbf{x})/\sum_{\mathbf{z}\in\Lambda} \rho_{\mathbf{c},\sigma}(\mathbf{z})$ for $\mathbf{x} \in \Lambda$, where $\rho = \exp(-\pi||\mathbf{x} - \mathbf{c}||/\sigma^2)$. When omitted, $\mathbf{c} = \mathbf{0}$ and $\sigma = 1.0$. The discrete Gaussian distribution $D_{\mathbb{Z},c,\sigma}$ is defined over integers and used as the primitive in all discrete Gaussian sampling operations. The Gaussian distribution $D_{\mathcal{R},\mathbf{c},\sigma} = D_{\mathbb{Z}^n,\mathbf{c},\sigma}$ denotes the discrete Gaussian sampling operation applied to cyclotomic rings.

The notation $a \leftarrow_U \mathbb{Z}_q$, (or $a \leftarrow_U \mathbb{Z}_q^n$, $a \leftarrow_U \mathcal{R}_q$) is used for discrete uniformly random distribution.

B. Ring Learning with Errors

Let s be an arbitrary (and unknown) polynomial in $\mathcal{R}_q$. We consider a number of pairs of the form $(a_i, a_i s + e_i) \in \mathcal{R}_q^2$, where $a_i \leftarrow_U \mathcal{R}_q$ and $e_i \leftarrow D_{\mathcal{R},\sigma}$ with a relatively small $\sigma > 1.0$. We then define RLWE hardness assumptions used in the security proofs of the construction in this paper.

Definition 3.1: The search RLWE assumption is that it is hard to find s given a list of pairs $(a_i, a_i s + e_i)$ for $i = 0, \ldots, t$.

Definition 3.2: The decision RLWE assumption is that it is hard to distinguish polynomials $(a_i s + e_i)$ and b_i for $i = 0, \ldots, t$, where each b_i is uniformly randomly chosen in $\mathcal{R}_q$.

Informally speaking, in both definitions, t stands for the number of samples a polynomial-time adversary or distinguisher can obtain. Related to Def 3.2, $a_i s + e_i$ is sometimes said to be from a *pseudorandom distribution* as it is difficult to distinguish it from a uniformly randomly chosen b_i.

The hardness of the RLWE assumptions depends on the choice of ring dimension n, the size of q and a bound Δ for the coefficients of e_i, which is determined by the distribution parameter of $D_{\mathcal{R},\sigma}$.

For the RLWE hardness assumptions to hold, the values of n and q can be selected using the inequality derived in [36], namely,

$$n \geq \frac{\log_2(q/\sigma)}{4\log_2(\delta)}. \qquad (1)$$

Here, σ refers to the distribution parameter used in $D_{\mathcal{R},\sigma}$ and δ is the root Hermite factor: a measure of the lattice security that can be mapped to the number of bits of security. In a seminal work by Chen and Nguyen [37], it is claimed that the lattice security is largely determined by the root Hermite factor. In another work by Lindner and Peikert [38], a formulation is given for the lattice security based on δ as

$$t_{\texttt{BKZ}} = 1.8/\log_2(\delta) - 110, \qquad (2)$$

where $t_{\texttt{BKZ}}$ is the estimated running time of the BKZ algorithm [39]. For instance, the value of $\delta = 1.006$ corresponds to about 100 bits of security.

The smoothing (distribution) parameter σ can be estimated as

$$\sigma \approx \sqrt{\ln(2n_m/\epsilon)/\pi},$$

where n_m is the maximum ring dimension and ϵ is the bound on the statistical error introduced by each randomized-rounding operation [30]. For $n_m \leq 2^{14}$ and $\epsilon \geq 2^{-80}$, the value of $\sigma \approx 4.578$.

C. Gaussian Sampling for Lattice Trapdoors

A trapdoor is an extra piece of information that enables the computation of a solution to an otherwise hard problem. In this paper, we rely on the lattice trapdoors introduced in [30]. Let $\mathbf{A} \in \mathcal{R}_q^{1\times m}$ be a row vector of ring elements generated using uniformly random distribution, where m is a parameter specific to the chosen trapdoor construction. Informally speaking, for an arbitrarily chosen $\beta \in \mathcal{R}_q$, it is computationally hard to find a vector of short polynomials $\boldsymbol{\alpha} \in \mathcal{R}_q^{m\times 1}$ that satisfies $\mathbf{A}\boldsymbol{\alpha} = \beta$. Furthermore, the vectors in the solution must be spherically distributed with a Gaussian function and a distribution parameter s; namely, we should have $\boldsymbol{\alpha} \leftarrow D_{\Lambda,s}$.

Finding such short vectors is usually referred to as a *preimage (Gaussian) sampling* operation for an arbitrary *syndrome* β. The hardness assumption can be based on the hardness of the approximate shortest independent vector problem, namely $\texttt{SIVP}_\gamma$. On the other hand, a trapdoor $\mathbf{T_A}$ for $\mathbf{A}$ can be used to compute such short vectors efficiently.

We use a ring-based trapdoor construction proposed in [23] (depicted in Algorithm 1). The trapdoor consists of two row vectors of short ring elements sampled using a Gaussian distribution with the distribution parameter σ, $\mathbf{T_A} = (\boldsymbol{\rho}, \boldsymbol{\upsilon})$. While the trapdoor $\mathbf{T}_\mathbf{A}$ is secret, the public key $\mathbf{A}$ is pseudorandom and enjoys the RLWE hardness assumptions. All parameters needed for trapdoor generation, namely, σ, q, k, and n, are selected based on the security parameter λ.

The vector $\mathbf{g}^T = (2^0, 2^1, \ldots, 2^{k-1})$ is introduced in [30] and is referred to as *the primitive vector*. Using a primitive vector $\mathbf{g}$ we can generate a $\mathbf{G}$*-lattice*, for which preimage sampling can be efficiently computed. The work in [30] provides a very efficient preimage sampling algorithm for a modulus q that is a power of two. However, for many cryptographic schemes such as IBE and ABE a prime modulus is more common. Therefore, the preimage sampling algorithm for $\mathbf{G}$-lattices with arbitrary modulus proposed in [32] is used in this work.

Algorithm 1 Trapdoor generation using RLWE [23]

function TRAPGEN(λ)
  Determine σ, q, k and n
  $a \leftarrow_U \mathcal{R}_q$
  $\boldsymbol{\rho} \leftarrow [\rho_1, \ldots, \rho_k]$ where $\rho_i \leftarrow D_{\mathcal{R},\sigma}$ for $i = 1, \ldots, k$
  $\boldsymbol{\upsilon} \leftarrow [\upsilon_1, \ldots, \upsilon_k]$ where $\upsilon_i \leftarrow D_{\mathcal{R},\sigma}$ for $i = 1, \ldots, k$
  $\mathbf{A} \leftarrow [1, a, g_1 - (a\rho_1 + \upsilon_1), \ldots, g_k - (a\rho_k + \upsilon_k)]$
  return $(\mathbf{A}, \mathbf{T}_\mathbf{A} = (\boldsymbol{\rho}, \boldsymbol{\upsilon}))$
end function

If preimage sampling is efficiently computable for $\mathbf{G}$-lattice, we can show that it is also efficiently computable for the public key $\mathbf{A}$ given the trapdoor $\mathbf{T}_\mathbf{A}$. For an arbitrary syndrome $\beta \in \mathcal{R}_q$, it is easy to see that $\mathbf{y} = (\mathbf{x}^T\boldsymbol{\upsilon}, \mathbf{x}^T\boldsymbol{\rho}, x_1, x_2, \ldots, x_k)$ is a short solution to $\mathbf{A}\mathbf{y} = \beta$, where $\mathbf{g}^T\mathbf{x} = \beta$ and $\mathbf{x}$ is a short solution.

On the other hand, the framework in [14] requires the preimage sampling algorithm to produce a spherically distributed solution for a given syndrome $\beta \in \mathcal{R}_q$. It is shown in [23] that solutions in the form of $\mathbf{y} = (\mathbf{x}^T\boldsymbol{\upsilon}, \mathbf{x}^T\boldsymbol{\rho}, x_1, x_2, \ldots, x_k)$ are not spherically distributed but ellipsoidal, and therefore leak information about the trapdoor. To this end, a perturbation is added to $\mathbf{y}$. Algorithm 2 gives a high level description of our secure preimage sampling algorithm, which is called Gaussian preimage sampling. The algorithm relies on the preimage sampling on $\mathbf{G}$-lattices, but it perturbs the preimage sampled via the primitive vector $\mathbf{g}$. This requires a perturbation generation function PERTURB to produce a perturbation vector $\mathbf{p}$, which is then added to the solution $\boldsymbol{\alpha}$ to ensure spherical Gaussian distribution.

Algorithm 2 Gaussian preimage sampling [30]

function GAUSSSAMP($\mathbf{A}, (\boldsymbol{\rho}, \boldsymbol{\upsilon}), \beta, \sigma, s$)
  $\mathbf{p} \leftarrow$ PERTURB$(n, q, s, 2\sigma, (\boldsymbol{\rho}, \boldsymbol{\upsilon}))$
  $\mathbf{z} \leftarrow$ SAMPLEG$(\sigma, \beta - \mathbf{A}\mathbf{p}, q)$
  $\boldsymbol{\alpha} \leftarrow [p_1 + \boldsymbol{\upsilon}\mathbf{z}, p_2 + \boldsymbol{\rho}\mathbf{z}, p_3 + z_1, \ldots, p_{k+2} + z_k]$
  return $\boldsymbol{\alpha}$
end function

To summarize, we have $\mathbf{A}\boldsymbol{\alpha} = \beta$, where $\boldsymbol{\alpha}$ follows a zero-centered Gaussian distribution with distribution parameter s, where $\mathbf{p} \in \mathcal{R}^{m\times 1}$, $\mathbf{z} \in \mathcal{R}^{k\times 1}$, and $m = k + 2$.

The parameter s in PERTURB operation is referred to as the *spectral norm*, which quantifies the norm of the solution and is computed as

$$s > C \cdot \sigma^2 \cdot \left(\sqrt{nk} + \sqrt{2n} + 4.7\right),$$

where C is a constant that can be found empirically. In our experiments we used $C = 1.80$. For more insight into the trapdoor construction used in this paper, one can profitably refer to [32] for theoretical explanation and to [33] for specific construction details.

D. Efficient Polynomial Multiplications

As our construction necessitates thousands of multiplications in R_q, where q can be a multiprecision integer, we take advantage of number-theoretic transform (NTT) and the Chinese Remainder Theorem (CRT) to accelerate polynomial multiplications in our implementation.

NTT is obtained by performing the discrete Fourier transform over a finite field $\mathbb{F}_P$, where P is a prime number. Let $w_N \in \mathbb{F}_P$ be a primitive N-th root of unity, which exists under the condition that N divides $P - 1$. The N-point NTT/INTT conversions with w_N are defined as: $\hat{\mathbf{a}} = \mathsf{NTT}^N_{w_N}(\mathbf{a})$, where $\hat{a}_i = \sum_{j=0}^{N-1} a_j w_N^{ij}$ and $\mathbf{a} = \mathsf{INTT}^N_{w_N}(\hat{\mathbf{a}})$, where $a_i = \frac{1}{N}\sum_{j=0}^{N-1} \hat{a}_j w_N^{-ij}$.

A polynomial multiplication $c(x) = a(x)b(x)$ in $\mathcal{R}$ that requires modulo reduction with $x^n + 1$, can be achieved by the negative wrapped convolution [40] that directly computes $c_i = \sum_{j=0}^{i} a_j b_{i-j} - \sum_{j=i+1}^{n-1} a_j b_{n+i-j}$. The method utilizes a primitive $2n$-th root of unity w_{2n} that exists if $2n$ divides $P-1$. Then $w_n = w_{2n}^2$ is a primitive n-th root of unity. Let $\mathbf{v} = (1, w_{2n}, \ldots w_{2n}^{n-1})$ and $\mathbf{v}^{-1} = (1, w_{2n}^{-1}, \ldots w_{2n}^{-(n-1)})$. We use $\mathsf{NTT}(\mathbf{a})$ for $\mathsf{NTT}^n_{w_n}(\mathbf{a} \odot \mathbf{v})$ and $\mathsf{INTT}(\hat{\mathbf{a}})$ for $\mathsf{INTT}^n_{w_n}(\hat{\mathbf{a}}) \odot \mathbf{v}^{-1}$ for simplicity, where $\odot$ denotes the coefficient-wise dot product. A multiplication in $\mathcal{R}$ is computed by $\mathbf{c} = \mathsf{INTT}^n_{w_n}\left(\mathsf{NTT}^n_{w_n}(\mathbf{a} \odot \mathbf{v}) \odot \mathsf{NTT}^n_{w_n}(\mathbf{b} \odot \mathbf{v})\right) \odot \mathbf{v}^{-1}$. The negative wrapped convolution still supports additions in NTT domain.

The CRT is adopted to handle large integers. Given t pairwise coprime numbers $q_0, q_1, \ldots q_{t-1}$ and their product $Q = \prod_{i=0}^{t-1} q_i$, there exists an isomorphism $\mathsf{CRT}()$: $\mathbb{Z}_Q \rightarrow \mathbb{Z}_{q_0} \times \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_{t-1}}$. For an integer a, the CRT conversion and its inverse (ICRT) are defined as $\left(\tilde{a}^{(0)}, \tilde{a}^{(1)}, \ldots \tilde{a}^{(t-1)}\right) = \mathsf{CRT}(a) \rightarrow \tilde{a}^{(j)} = [a]_{p_j}$ and $a = \mathsf{ICRT}_{j\in\mathbb{Z}_t}(\tilde{a}^{(j)}) = \left[\sum_{j=0}^{t-1}\left[\tilde{a}^{(j)}\frac{q_j}{Q}\right]_{q_j}\frac{Q}{q_j}\right]_Q$.

The combination of NTT and the CRT has been commonly adopted to compute multiplications of polynomials with large coefficients, for example in [41] and [42].

When a ring element $a \in \mathcal{R}_q$ (in polynomial representation) is transformed into $\tilde{\mathbf{a}}$ using NTT, the latter is referred as *the evaluation* representation, whereby the multiplication is extremely efficient as it is performed component-wise. The transformation operations themselves (NTT and INTT) are usually the computational bottlenecks. Therefore, provided that the cryptographic computations permit, it is better to keep operands in the evaluation representation as long as possible.

We use this approach to accelerate cryptographic computations in this paper.

E. CUDA GPU Background

CUDA abstracts the hardware architecture for developers. A CUDA-enabled GPU partitions thousands of cores into an array of streaming multiprocessors (SMMs). Each SMM is built with function units, registers, private memory, data cache and instruction buffers. An SMM executes threads in groups of 32 parallel CUDA threads (*warps*) in an SIMT (Single-Instruction, Multiple-Thread) style.

CUDA programs offload intensive computation to the device (i.e., a GPU), while the rest of application remains on the host (i.e., a CPU). The host launches a kernel which is a C function executed n times in parallel by n threads on the device. A kernel is executed by a grid of thread blocks: threads are grouped into blocks; blocks are organized into a grid. Threads within a block can cooperate through shared memory and synchronize their execution. The dimension of a block is preferred to be a multiple of 32 (warp size). Grid configuration is set at kernel launches and is important for the better utilization of computing resources. Both the host and the device maintain their own separate memory spaces in DRAM, referred to as host memory and device memory, respectively. Hence, a program transfers data between host and device memory for computation on the device.

Three types of device memory are listed from fast to slow:

- Constant memory is read-only to all grids, cached and broadcast to all involved threads when requested. However, its size cannot exceed 64 KB.
- Shared Memory is shared by threads in the same block. It may have bank conflicts that cause accesses to be serialized. Bank conflicts should be minimized. Shared memory size cannot exceed 48 KB per block.
- Global Memory is slow and visible to all threads. It favors a coalesced access pattern that avoids or minimizes overfetch. As it is adequate in size (several gigabytes) the global memory is where data are stored in general.

IV. KP-ABE Basics

In a KP-ABE scheme, plaintext is encrypted under a set of attribute values, which serves as the public key of the system whereas a private key corresponds to a specific access policy. For sake of simplicity, we work with binary attributes, thus a set of attributes is $\mathcal{X} = \{x_1, x_2, \dots, x_\ell\}$ where $x_i \in \{0, 1\}$.

An access policy, usually expressed as a circuit over a set of attributes, defines the rules as to who can decrypt the ciphertexts in the ABE scheme.

*In our construction of KP-ABE, an access policy corresponds to a Boolean circuit, which outputs logical-*0 *when its input takes the required set of attribute values.* An example access policy and the corresponding circuit are given in Example 4.1.

Example 4.1: An employee in a software company can decrypt source files in a software development project if she meets the following requirements: If she is a developer and working on the project; or if she is an employee of the company and has power user capabilities. Here we can obtain four binary attributes from these requirements, namely

- x_1: Is the user a developer? (YES or NO)
- x_2: Is the user working on the project? (YES or NO)
- x_3: Is the user employee of the company? (YES or NO)
- x_4: Is the user a power user? (YES or NO)

Then, the Boolean expression for the corresponding policy circuit is

$f(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2) \vee (x_3 \wedge x_4)$.

In our KP-ABE construction, however, we search for a circuit that outputs logical-0 when these attributes take the required values; namely, only when $x_1 = x_2 = 1$ or $x_3 = x_4 = 1$. The Boolean expression for such circuit is then $f(x_1, x_2, x_3, x_4) = (1 - x_1x_2)(1 - x_3x_4)$. For simplicity and generality, we adopt an arithmetic notation for Boolean expressions.

A KP-ABE scheme requires a trusted third party, PKG, that generates private keys corresponding to access policies. For this, PKG knows some master secret, or more technically a trapdoor, to generate a private key for any access policy.

A KP-ABE scheme is a family of functions, namely `Setup`, `Encrypt`, `KeyGen`, and `Decrypt`, which are explained in the following.

- SETUP$(1^\lambda, \ell) \rightarrow \{\texttt{MPK}, \texttt{MSK}\}$: Given a security parameter λ and the number of attributes ℓ, PKG generates a master public key `MPK` and a master secret key `MSK`. `MPK` contains the ABE public parameters while `MSK` consists of the trapdoor, with which PKG generates secret keys for access policies.
- ENCRYPT$(\mu, \mathbf{x}, \texttt{MPK}) \rightarrow \mathbf{C}$: Using `MPK` and attribute values $\mathbf{x} \in \{0, 1\}^\ell$, sender encrypts the message μ, outputs the ciphertext $\mathbf{C}$.
- KEYGEN$(\texttt{MSK}, \texttt{MPK}, f) \rightarrow \boldsymbol{\alpha}_f$: Given `MSK` and a policy (implemented by a Boolean circuit $f : \{0, 1\}^\ell \rightarrow \{0, 1\}$), PKG generates the secret key $\boldsymbol{\alpha}_f$ corresponding to f. PKG sends $\boldsymbol{\alpha}_f$ to the receiver that is authorized to decrypt ciphertexts encrypted under f.
- DECRYPT$(\mathbf{C}, \boldsymbol{\alpha}_f, \tilde{\mathbf{x}}) \rightarrow \bar{\mu}$: The decryption process consists of two phases: i) the homomorphic evaluation process that transforms ciphertext $\mathbf{C}$ to $\mathbf{C}_f$ so that the latter can be decrypted by $\boldsymbol{\alpha}_f$ and ii) the actual decryption operation that results in $\bar{\mu}$, which is equal to the original message μ if receiver has $\boldsymbol{\alpha}_f$.

Naturally, decryption succeeds only if the same attribute values are used in ENCRYPT and DECRYPT operations, namely $\mathbf{x} = \tilde{\mathbf{x}}$.

A distinctive and powerful property of KP-ABE is that the policy can be determined after the encryption operation. This requires, on the other hand, two technically challenging operations: homomorphic evaluation of the public keys and ciphertext. Our work demonstrates that they can be efficiently performed via our state-of-the-art GPU implementation.

V. Our Construction

In this section we present our construction of KP-ABE and explain KP-ABE operations in detail. We omit some public parameters such as standard deviation σ, the modulus q, its bit length k etc. from the algorithms for sake of simplicity.

A. Setup

In the setup phase, PKG generates a master public key (MPK) and the corresponding master secret key MSK using Algorithm 3. The master public key contains the ring vectors $\mathbf{B}_i$ corresponding to the attributes. As previously explained, $\mathbf{T_A}$ is a trapdoor associated with the vector $\mathbf{A}$ used to find a short solution $\boldsymbol{\alpha}$ to $\mathbf{A}\boldsymbol{\alpha} = \beta$ for an arbitrary $\beta \in \mathcal{R}_q$, where $\boldsymbol{\alpha}$ is short in the sense that it follows a zero-centered Gaussian distribution with a relatively small distribution parameter. Consequently, $\mathbf{T_A}$ enables PKG to generate a secret key for a given access policy as explained in Section V-D.

Algorithm 3 KP-ABE Setup Operation

function $\text{SETUP}(\lambda, \ell)$
  $(\mathbf{A}, \mathbf{T_A}) \leftarrow \text{TRAPGEN}(\lambda)$
  $\mathbf{B}_i \leftarrow_U \mathcal{R}_q^{1\times m}$ for $i = 0, 1, \ldots, l$
  $\beta \leftarrow_U \mathcal{R}_q$
  $\mathtt{MPK} \leftarrow \{\mathbf{A}, (\mathbf{B}_i)_{i=0}^{\ell}, \beta\}$
  $\mathtt{MSK} \leftarrow \{\mathbf{T_A}\}$
  return MPK, MSK
end function

Here, the public key $\mathbf{A}$ is pseudorandom and enjoys the hardness of RLWE as demonstrated in [23]. This basically means that it is computationally infeasible to obtain $\mathbf{T_A}$ given $\mathbf{A}$ and therefore only PKG can generate private keys.

B. Encryption

In our KP-ABE construction, the encryption operation, as described in Algorithm 4, takes as input MPK, the attribute values $\mathbf{x} \in \{0,1\}^{\ell}$, and the plaintext message $\mu \in \mathcal{R}_2$ and outputs the ciphertext pair $\mathbf{C_{in}} \in \mathcal{R}_q^{(\ell+2)m}$ and $c_1 \in \mathcal{R}_q$. The encryption algorithm is a variation of the dual Regev encryption algorithm, which is originally proposed for IBE schemes in [14] and adapted to ring setting in [24]. In the dual Regev algorithm, the security is based on RLWE hardness assumptions given in Definitions 3.1 and 3.2. The search RLWE hardness assumption prevents adversary from computing $s \in \mathcal{R}_q$ used in the encryption whereas the ciphertext components are pseudorandom due to the decision RLWE assumption.

Algorithm 4 ABE Encryption Algorithm

function $\text{ENCRYPT}(\mu, \mathbf{x}, \mathtt{MPK})$
  $s \leftarrow_U R_q$; $e_1 \leftarrow D_{\mathcal{R},\sigma}$; $\mathbf{e_A} \leftarrow D_{\mathcal{R}^{1\times m},\sigma}$
  $\mathbf{S}_i \leftarrow_U \{\pm 1\}^{m\times m}$ for $i = 0, \ldots, \ell$
  $\mathbf{e}_0 \leftarrow (\mathbf{e}_A^T | \mathbf{e}_A^T \mathbf{S}_0 | \mathbf{e}_A^T \mathbf{S}_1 | \ldots | \mathbf{e}_A^T \mathbf{S}_\ell)^T$
  $\mathbf{C_{in}} \leftarrow (\mathbf{A} | (\mathbf{G} + \mathbf{B}_0) | (x_1 \mathbf{G} + \mathbf{B}_1) | \ldots | (x_\ell \mathbf{G} + \mathbf{B}_\ell))^T s + \mathbf{e}_0$
  $c_1 \leftarrow \beta s + e_1 + \mu \lceil \frac{q}{2} \rceil$
  return $(\mathbf{C_{in}}, c_1)$
end function

In Algorithm 4, $\mathbf{G} = (1, 2, 2^2, \ldots, 2^{k-1}, 0, 0)$ is the primitive row vector of constant polynomials extended by two 0s to match the dimension of other vectors of polynomials since $m = k + 2$. The ring element e_1 and the vector of polynomials $\mathbf{e}_A$ are both sampled from the same discrete Gaussian distribution and they are occasionally referred as *error* or *noise* components in the ciphertext, making the decryption impossible when exceeding a certain threshold.

For easy reference, we adopt the notation $\mathbf{C_A} = \mathbf{A}^T s + \mathbf{e}_{0,A}$ and $\mathbf{C}_i = (x_i \mathbf{G} + \mathbf{B}_i)^T s + \mathbf{e}_{0,i}$ for $i = 0, 1, \ldots, \ell$, where the latter encrypts the attribute vector $\mathbf{x}$. Here, $x_0 = 1$ is not an attribute itself, but a necessary component to implement logical gates in the policy circuit.

C. Evaluation of Public Keys and Ciphertext

Both public keys ($\mathbf{B}_i$) and the ciphertexts ($\mathbf{C}_i$) corresponding to attributes are homomorphically evaluated over an access policy circuit f. This way, we can obtain a public key $\mathbf{B}_f$ and a ciphertext $\mathbf{C}_f$ that correspond to f. Hence, we have two functions:

- **Public Key Evaluation**: $\text{EVALPK}(\mathbf{x}, \mathbf{B}_i, f) \to \mathbf{B}_f$,
- **Ciphertext Evaluation**: $\text{EVALCT}(\mathbf{x}, \mathbf{C}_i, f) \to \mathbf{C}_f$,

where $\mathbf{B}_i, \mathbf{B}_f \in \mathcal{R}_q^{1\times m}$, $\mathbf{C}_i, \mathbf{C}_f \in \mathcal{R}_q^m$ for $i = 0, 1, \ldots, \ell$ and $\mathbf{x} \in \{0,1\}^{\ell}$.

Considering Example 4.1, we can visualize the policy f as a Boolean circuit with two NAND gates whose outputs are connected to an AND gate. Evaluation of ciphertext leads to noise increase in the error vectors (i.e., $\mathbf{e}_{0,i}$), and the noise level should not exceed the threshold for the chosen ciphertext modulus q to ensure correct decryption. All the details of homomorphic evaluation for our benchmark circuit are explained in Section VI.

D. Key Generation

The vector $\mathbf{C}_f$ obtained after homomorphic evaluation in the previous section can be considered as a ciphertext encrypted under the public key $\mathbf{B}_f$. Since both $\mathbf{C}_f$ and $\mathbf{B}_f$ correspond to the access policy f, we can write $\mathbf{C}_f = \mathbf{B}_f^T s + \mathbf{e}_f$, where $||\mathbf{e}_f|| > ||\mathbf{e}_{0,i}||$.

PKG uses Algorithm 5 to generate a secret key $\boldsymbol{\alpha}_f$ corresponding to $(\mathbf{A}|\mathbf{B}_f)$. Note that $(\mathbf{A}|\mathbf{B}_f)\boldsymbol{\alpha}_f = \beta$, where $\boldsymbol{\alpha}_f \in \mathcal{R}^{2m}$ is a vector of short ring elements. Algorithm 5 is the ring version of the algorithm in [19].

E. ABE Decryption

Decryption operation can be perfomed using $\bar{\mu} = c_1 - \boldsymbol{\alpha}_f^T (\mathbf{C_A}^T | \mathbf{C}_f^T)^T$. We can prove the correctness as follows:

$$\begin{aligned}
\bar{\mu} =& c_1 - \boldsymbol{\alpha}_f^T (\mathbf{C_A}^T | \mathbf{C}_f^T)^T \\
=& c_1 - (\boldsymbol{\alpha_A}^T (\mathbf{A}^T s + \mathbf{e}_{0,A}) + \boldsymbol{\alpha_B}^T (\mathbf{B}_f^T s + \mathbf{e_f})) \\
=& c_1 - ((\mathbf{A}\boldsymbol{\alpha_A} + \mathbf{B}_f \boldsymbol{\alpha_B})^T s + \boldsymbol{\alpha_A}^T \mathbf{e}_{0,A} + \boldsymbol{\alpha_B}^T \mathbf{e}_f) \\
=& \beta s + e_1 + \mu \lceil q/2 \rceil - (\beta s + \boldsymbol{\alpha_A}^T \mathbf{e}_{0,A} + \boldsymbol{\alpha_B}^T \mathbf{e}_f) \\
=& \mu \lceil q/2 \rceil + e_1 - \boldsymbol{\alpha_A}^T \mathbf{e}_{0,A} - \boldsymbol{\alpha_B}^T \mathbf{e}_f
\end{aligned}$$

If the terms $e_1 - \boldsymbol{\alpha_A}^T \mathbf{e}_{0,A} - \boldsymbol{\alpha_B}^T \mathbf{e}_f$ have sufficiently small norms, a simple thresholding operation yields the correct plaintext: if the i-th coefficient $\bar{\mu}_i < q/4$, the decryption operation outputs 1; otherwise 0, for $i = 0, \ldots, n-1$. Therefore, correctness of the decryption operation is determined by the norm of the secret key generated by GAUSSSAMP and of the

Algorithm 5 ABE Key Generation Algorithm [19]

function KEYGEN($\mathbf{A}, \mathbf{B}_f, \beta$, MSK)
  $\boldsymbol{\alpha_B} \leftarrow D_{\mathcal{R}^m,\sigma}$
  $t \leftarrow \beta - \mathbf{B}_f \boldsymbol{\alpha_B}$
  $\boldsymbol{\alpha_A} \leftarrow$ GAUSSSAMP$(\mathbf{A}, \mathbf{T_A}, t)$
  $\boldsymbol{\alpha}_f^T \leftarrow (\boldsymbol{\alpha_A}^T | \boldsymbol{\alpha_B}^T)$
  return $\boldsymbol{\alpha}_f$
end function

error term $\mathbf{e}_f$ in $\mathbf{C}_f$. The latter error term $\mathbf{e}_f$ is the result of homomorphic EVALCT process.

In Section VI, we explain homomorphic evaluation of public keys and ciphertext over simple Boolean circuits and show how it increases the noise of ciphertexts. The noise analysis is important to determine the system parameters such as ring dimension and modulus size to ensure correctness and targeted security level.

The proposed KP-ABE scheme is selectively secure, where the definition of the *selective security* is introduced in [3]. Informally speaking, in selective security a polynomial time adversary first commits to a challenge attribute value x^*, then receives KP-ABE public key MPK and has access to a key generation oracle that returns a secret key $\boldsymbol{\alpha}_f$ corresponding to any access policy f provided that $f(\mathbf{x}^*) \neq 0$. The selective security requires that the adversary cannot distinguish, with a non-negligible advantage, between the ciphertexts of two different messages encrypted under the challenge attribute x^*. For selective security proof, an essential hardness assumption is the decisional RLWE problem. The formal security proof is provided in Section IX-B of Appendix.

VI. EVALUATION OF PUBLIC KEYS AND CIPHERTEXT ON GATES

When a (arithmetic or logic) gate is evaluated for an access policy f we obtain a new public key $\mathbf{B}_f$ and a new ciphertext $\mathbf{C}_f$ for the output of the gate. As a result, the noise level in the resulting ciphertext becomes larger than the noise level in the input ciphertext. We need to keep the noise growth under control for correct decryption, whereby the noise level in the resulting ciphertext must remain under the threshold $q/4$, as shown in the preceding section.

For the sake of simplicity, we deal with only the first part of the ciphertext $\mathbf{C}_{\text{in}}$ as the other part c_1 is not affected by the evaluation process.

A. Arithmetic Addition/Subtraction

Suppose we have two attributes x_1 and x_2, then the KP-ABE encryption of message $\mu \in R_2$ is computed as $\mathbf{C}_{\text{in}} = (\mathbf{A}|(\mathbf{G}+\mathbf{B}_0)|(x_1\mathbf{G}+\mathbf{B}_1)|(x_2\mathbf{G}+\mathbf{B}_2))^T s + \mathbf{e}_0$, where $\mathbf{e}_0 = \left(\mathbf{e}_{0,A}^T|\mathbf{e}_{0,0}^T|\mathbf{e}_{0,1}^T|\mathbf{e}_{0,2}^T\right)^T \in R_q^{4m}$. We can also partition the ciphertext as $\mathbf{C_A} = \mathbf{A}^T s + \mathbf{e}_{0,A}$ and $\mathbf{C}_i = (x_i\mathbf{G} + \mathbf{B}_i)^T s + \mathbf{e}_{0,i}$ for $i = 0, 1, 2$ and $x_0 = 1$. If the access policy is a single addition or subtraction operation, the circuit evaluation is straightforward $\mathbf{C}_\pm = \mathbf{C}_1 \pm \mathbf{C}_2$, $\mathbf{B}_\pm = \mathbf{B}_1 \pm \mathbf{B}_2$. We can also formulate the increase in noise level in error vectors as $\mathbf{e}_{0,\pm} = \mathbf{e}_{0,1} \pm \mathbf{e}_{0,2}$. As easily observed, the evaluation is inexpensive and the increase in noise level is additive (very limited).

B. Multiplication or Logical AND Operation

As we work with binary attributes, the multiplication and logical AND operations are identical. Using the ciphertext inputs in Section VI-A, the multiplication operation is performed homomorphically and we obtain $\mathbf{C}_\times = x_2\mathbf{C}_1 + \Psi^T\mathbf{C}_2$ and $\mathbf{B}_\times = \mathbf{B}_2\boldsymbol{\Psi}$, where $\boldsymbol{\Psi} =$ BITDECOMP$(-\mathbf{B}_1)$ and BITDECOMP stands for the bit decomposition operation over the polynomials of $-\mathbf{B}_1$ such that $-\mathbf{B}_1 = \mathbf{G}\boldsymbol{\Psi}$.

Suppose that $b_i = b_{i,0} + b_{i,1}x + \ldots + b_{i,n-1}x^{n-1}$ with $b_{i,j} \in \mathbb{Z}_q$ is the i-th polynomial in $-\mathbf{B}_1$ and $b_{i,j,h}$ is h-th bit of the j-th coefficient of b_i. Then the binary polynomial $\psi_{h,i}$ in the i-th column and h-th row of $\boldsymbol{\Psi}$ can be computed as

$$\psi_{h,i} = b_{i,0,h} + b_{i,1,h}x + \ldots + b_{i,n-1,h}x^{n-1},$$

where $0 \leq i, h \leq m-1$. Therefore, $\boldsymbol{\Psi}$ is a matrix of dimension $m \times m$, where elements are binary polynomials of degree $n-1$ or less, namely $\boldsymbol{\Psi} \in R_2^{m\times m}$.

The noise in the output ciphertext $\mathbf{C}_\times$ has the following form $\mathbf{e}_{0,\times} = x_2\mathbf{e}_{0,1} + \boldsymbol{\Psi}^T\mathbf{e}_{0,2}$. The dominating factor in noise growth is due to the term $\boldsymbol{\Psi}^T\mathbf{e}_{0,2}$, which is a vector-matrix product of ring elements. The statistical properties of the binary decomposition matrix result in fast increase in the noise. As $\boldsymbol{\Psi}$ consists of vectors in $\mathcal{R}_2$ and recalling $\mathbf{B}_1$ is a uniformly randomly generated vector of polynomialsthe coefficients of the polynomials in $\boldsymbol{\Psi}$ are distributed with the mean of 0.5 and standard deviation of 0.5. Therefore, a small non-zero mean in $\mathbf{e}_{0,2}$ will contribute to a considerable increase in the mean and standard deviation of $\mathbf{e}_{0,\times}$.

To limit the noise growth, we use binary non-adjacent form (NAF) of integers in the construction of bit decomposition matrix as explained in Algorithm 6. Binary NAF, which uses -1 in addition to 0 and 1 to represent integers, produces a $\boldsymbol{\Psi}$ in which coefficients of the polynomials are distributed with zero mean, since binary non-adjacent form is a balanced representation. For a better understanding of how binary NAF helps control the noise growth one can profitably refer to Section IX in Appendix.

NAND gates are universal in the sense that any Boolean functions can be realized using only NAND gates. Homomorphic evaluation of a NAND gate can be performed in a very similar way to NAND gates, which is shown in Section VIII-A in Appendix. Thus, we are interested in a benchmark circuit consisting of only NAND gates. More specifically, our benchmark circuit has a topology of a binary tree, an example of which is given with four binary attributes in Section VIII-B in Appendix. A generic algorithm of public key and ciphertext evaluation of binary NAND tree circuits is illustrated in Algorithm 7. For the noise analysis and correctness and security constraints of the benchmark circuit, one can refer to Section IX in Appendix. See also Table V for our estimates of modulus sizes and ring dimensions for some given numbers of attributes.

Our NAND gate circuit provides an ultimate benchmark as only the depth of the policy circuit determines the complexity

Algorithm 6 NAF Bit Decomposition Operation

```
function NAFDecomp(B)
    for i = 0 to m − 1 do
        for j = 0 to n − 1 do
            y ← b_{i,j}; h ← 0
            while y > 0 do
                if y is odd then
                    z ← 2 − (y mod 4); y ← y − z
                else
                    z ← 0
                end if
                ψ_{h,i,j} ← z; y ← y/2; h ← h + 1
            end while
        end for
    end for
    return Ψ
end function
```

Algorithm 7 Public Key and Ciphertext Evaluation for NAND Trees

```
function EvalBenchmark(C_A, C_i, B_i, x̄, ℓ)
    for i = 1 to ℓ − 1 do
        x̄_{ℓ+i} ← (1 − x̄_{2i−1} x̄_{2i})
        Ψ_i ← NAFDecomp(−B_{2i−1})
        B_{ℓ+i} ← B_0 − B_{2i} Ψ_i
        C_{ℓ+i} ← C_0 − x̄_{2i} C_{2i−1} − Ψ_i^T C_{2i}
    end for
    B_f = B_{2ℓ−1}
    C_f = C_{2ℓ−1}
    return B_f, C_f
end function
```

of the KP-ABE scheme implemented here. To assess the performance of the KP-ABE scheme for any other policy circuit, all that one needs to do is to consider its depth and check the implementation results of the benchmark circuit of the same depth provided in this paper.

VII. Implementation Details and Results

In this section we explain our GPU implementation of the RLWE KP-ABE scheme and provide performance measurements. As no other similar ABE implementation is found in the literature, we only compare the throughput of our ring multiplication implementation with previous state-of-the-art implementations.

Table I shows the CPU execution times of KP-ABE operations obtained using the PALISADE library [33], [43], [44]. The timing figures suggest that the homomorphic ciphertext and public key evaluation (Algorithm 7) and encryption (Algorithm 4) operations do not scale well on a CPU. Decryption operation is already fast. Key generation can be made faster using a GPU, but as it is only performed occasionally per policy it is not a performance bottleneck in the KP-ABE implementation. Therefore, we focus on implementing only homomorphic evaluation and encryption operations on a GPU.

Before delving into technical details, we start with a high-level outline. ABE encryption and evaluation operations both perform many multiplication operations in $\mathcal{R}_q$. A GPU is the ideal platform since ABE requires thousands of multiplications, which are amenable to parallelization. We decide to follow the approach in [45], [42] and [46] rather than several other floating-point (fp) based GPU implementations such as [47], [48], [41], for the former relies on integer arithmetic and is faster than nearly all methods based on fp arithmetic. Although the performance reported in [45] seems to be slower than [41], our implementation is faster than [45] and outperforms any other implementations reported in the literature including [41]. A more detailed comparison of this work to cuFFT-based algorithms and [41] is provided in Section VII-D.

TABLE I: Execution times (in ms) of KP-ABE operations on a computer with Intel(R) Core(TM) i7-4720HQ CPU @2.6 GHz running Ubuntu 16.04 TLS using the PALISADE library in [33], [43], [44]

ℓ	KeyGen	Encrypt	EvalCT + EvalPK	Decrypt
2	94	69	311	3.6
4	156	123	1,081	5.48
8	166	259	3,365	6.19

We implemented an NTT-based fast negative wrapped convolution algorithm for multiplication in $\mathcal{R}_q$ that is customized for our scheme, based on the code in the CUDA Homomorphic Encryption Library (cuHE) [42] on GitHub. However, as the method has a general limitation on coefficient size we adopted the CRT to break any high-norm polynomial into t parallel low-norm polynomials. Any arithmetic computation in $\mathcal{R}_q$ is now mapped to corresponding operations over integer vectors in $\mathbb{Z}^n_{p_0}, \mathbb{Z}^n_{p_1}, \dots \mathbb{Z}^n_{p_{t-1}}$. The result is later converted to a single vector by ICRT and then reduced back to $\mathcal{R}_q$.

For a polynomial addition/subtraction, rather than performing it in CRT or NTT domain, it is more efficient to use polynomial representation in $\mathcal{R}_q$, if the arithmetic circuit does not involve multiplications. And an addition/subtraction can be embedded as a simple step in other functions. NAF bit decomposition yields polynomials in R_3. We skip the CRT conversions by mapping -1 coefficients to $P-1$ and feeding them directly to NTT.

We instantiate the Box-Muller method on GPU that samples $\mathbf{e}_0$ in Algorithm 4 from a discrete Gaussian distribution much more efficiently than on a CPU. All these modules are assembled to implement ABE encryption and evaluation operations.

A. Fast NTT for Negative Wrapped Convolution

For arithmetic in ring $\mathcal{R}_q$, we use an NTT-based approach and represent polynomials as vector of integers, namely in $\mathbb{Z}^n$. NTT conversions are performed in $\mathbb{F}_P$, where we choose $P = 2^{64} - 2^{32} + 1$ for fast modulo arithmetic as in [49]. As 8 is a primitive 64-th root of unity modulo P, any NTT with size $N \leq 64$ uses $w_N = 8^{\frac{64}{N}}$ as a primitive root of unity. This way, the multiplications of the input coefficients with the powers of root of unity (twiddle factors) are replaced with much cheaper bit-wise left shifts, e.g. $a \cdot w_{16}^3 = (a \ll 12)$. The reduction of the result (a large integer) modulo P can be achieved with 32-bit additions and subtractions. An example is given in Table VIII in Appendix.

Eliminating Conditional Branches. Suppose that we perform modulo P reduction on a (less than) 224-bit integer e.g. $x = x' \times 8^{49} = x' \ll 147$ where x_i denotes the i-th least significant 32-bit word of x and $x' \in \mathbb{F}_P$. We observe that $x_0 = x_1 = 0$ and compute it as $x \equiv (-x_2 - x_3) + (x_2 - x_4) \cdot 2^{32} \mod P$. Carry-out or borrow-in occurring at the 65-th bit will corrupt the result. Handling them normally involves conditional branches.

If threads in a single warp branch to different instructions, these threads will execute in sequence, which is inefficient. Thus, we predict the conditions where carry-out or borrow-in occurs, for every possible left shift offset, and handle them without branches. Since we know $x_4 \in [0, 255]$, we may compute the formula as $r = x_2 \cdot 2^{32} - (x_2 + x_3 + x_4 \cdot 2^{32})$ without any carry-out. And the integer value of the Boolean type element $r > x_2 \cdot 2^{32}$ is used to handle the borrow-in case. Then we apply a similar trick with the Boolean type element to obtain $x \mod P = r - P$ when $r \geq P$. Pseudo code for this is provided in Table VIII in Appendix. This effort eliminates conditional branches.

Parallelizing NTT/INTT on Multiple Threads. The main idea in our particular method for NTT operation is to (recursively) apply the four-step Cooley-Tukey algorithm [50] until NTT size drops below 64. Similar to [49] in the first level of recursion, we arrange an integer vector of $n = 2048$ (or $n = 4096$) as a two-dimensional vector of 64×32 (or 64×64). In $\mathsf{NTT}()$, we multiply each vector element with a power of w_{2n} for the negative wrapped convolution. Then we perform $\mathsf{NTT}_8^{64}()$ on each column, transpose the 2D-vector, multiply them with twiddle factors (powers of w_n) and eventually perform $\mathsf{NTT}_{8^2}^{32}()$ (or $\mathsf{NTT}_8^{64}()$) on each row. In $\mathsf{INTT}()$, we perform $\mathsf{NTT}_{8^{-2}}^{32}()$ (or $\mathsf{NTT}_{8^{-1}}^{64}()$) on each row first, multiply them with twiddle factors (powers of w_n^{-1}), transpose the 2D-vector and then perform $\mathsf{NTT}_{8^{-1}}^{64}()$ on each column. Finally, we multiply each vector element with a power of w_{2n}^{-1} and $\frac{1}{n}$. We exclude transposes in both forward and backward conversions. The column-wise and row-wise conversions are computed separately with two kernels. Each kernel uses $\frac{n}{8}$ threads which are divided into at most $\frac{n}{512}$ blocks, whereby each thread reads/writes 8 vector elements.

Minimized Thread Communication Overhead. 64-point or 32-point conversions are handled by 8 or 4 threads and require only a single synchronization of threads. Each thread is assigned 8 column elements to perform $\mathsf{NTT}_{8^8}^{8}()$ recursively and multiplies them with powers of 8 as twiddle factors (implemented as simple left shifts). Then it transposes the vector elements of 8×8, reads column elements and performs $\mathsf{NTT}_{8^8}^{8}()$ again. Note that the transpose operation benefits from the shared memory as mapping functions are optimized to minimize bank conflicts. A single synchronization of all threads is required only after writing to the shared memory.

Further Optimizations. We pre-compute $(w_n)^i$, $(w_n^{-1})^i$, $(w_{2n})^i$ and $(w_{2n}^{-1})^i$ for $i = 0, \ldots, n-1$ and store them as four separate vectors in the global memory. Although the same vector is stored twice in different order since $w_n^{-i} = w_n^{n-i}\ \forall i \in \mathbb{Z}_n$, it ensures coalesced global memory access. Also, we store $\frac{1}{n} w_{2n}^{-1}$ instead of w_{2n}^{-1} to save 8 integer multiplications per thread in every INTT conversions.

B. CRT Configurations

The NTT method defined above only applies on polynomials whose norm is smaller than P. We utilize CRT to break down an integer in $\mathbb{Z}_q$ into a vector of smaller integers smaller. We generate t CRT primes, namely $\{p_0,\ p_1,\ \ldots,\ p_{t-1}\}$, to convert a vector $\mathbf{f} \in \mathbb{Z}_q^n$ into its CRT domain value $\left(\tilde{\mathbf{f}}^{(0)},\ \tilde{\mathbf{f}}^{(1)},\ \ldots,\ \tilde{\mathbf{f}}^{(t-1)}\right)$ where $\tilde{\mathbf{f}}^{(j)} = [\mathbf{f}]_{p_j} \in \mathbb{Z}_{p_j}^n$. Suppose we compute $h(x) = \sum_{i=0}^{\tau-1} f_i(x) g_i(x)$ in R_q for any τ. We lift polynomials to their vector forms $\mathbf{h}, \mathbf{f}_i, \mathbf{g}_i \in \mathbb{Z}_q^n$. $\mathbf{h}$ is reduced modulo q from ICRT result $\mathbf{h}'$, where $\mathbf{h}'$ is computed as:

$$\mathbf{h}' = \mathsf{ICRT}_{j \in \mathbb{Z}_t}\left(\tilde{\mathbf{h}'}^{(j)}\right) = \mathsf{ICRT}_{j \in \mathbb{Z}_t}\left(\sum_{i=0}^{\tau-1} \tilde{\mathbf{f}}_i^{(j)} \tilde{\mathbf{g}}_i^{(j)}\right) \tag{3}$$

$$= \mathsf{ICRT}_{j \in \mathbb{Z}_t}\left(\mathsf{INTT}\left(\sum_{i=0}^{\tau-1} \mathsf{NTT}\left(\tilde{\mathbf{f}}_i^{(j)}\right) \odot \mathsf{NTT}\left(\tilde{\mathbf{g}}_i^{(j)}\right)\right)\right). \tag{4}$$

Constraints on the Size and Number of CRT Primes. Since the method utilizes different mathematical objects, namely $\mathbb{Z}_q$, $\mathbb{F}_{p_j}$'s and $\mathbb{F}_P$, it works correctly only if the following two constraints are satisfied:

$$P > \left\| \sum_{i=0}^{\tau-1} \tilde{\mathbf{f}}_i^{(j)} \tilde{\mathbf{g}}_i^{(j)} \right\|,\ \forall i \in \mathbb{Z}_\tau,\ \forall j \in \mathbb{Z}_t; \tag{5}$$

$$\prod_{j=0}^{t-1} p_j > \left\| \sum_{i=0}^{\tau-1} f_i(x) g_i(x) \right\|,\ \forall i \in \mathbb{Z}_\tau. \tag{6}$$

ABE encryption and evaluation operations impose different constraints on CRT parameter selection. A summary is presented in Table II. The constant factor 2 exists in all inequalities due to the negative wrapped convolution. For example in Encrypt (Algorithm 4), the coefficients of a product of two CRT domain polynomials fall within the interval $[-n(p_j - 1)^2, n(p_j - 1)^2]$, or that of a product of two $\mathcal{R}_q$ polynomials fall within the interval $[-n(q-1)^2, n(q-1)^2]$ when a prime modulus is in use. EvalBenchmark (Algorithm 7), on the other hand, consist of vector-matrix multiplication in the form $\mathcal{R}_q^m \leftarrow \mathcal{R}_q^m \times \mathcal{R}_3^{m \times m}$ (e.g., $\mathbf{B}_{2i} \mathbf{\Psi}_i$). Here, $\mathcal{R}_3$ is due to non-adjacent form decomposition in Algorithm 6. Consequently, EvalCT or EvalPK (EvalXX) provides a smaller upper-bound for CRT prime sizes (see Table II) and decreases the number of CRT primes (see Table V), compared to Encrypt.

Impact of Using Fewer CRT Primes. We assume that p_j's have similar sizes: $\lceil \log_2 p_j \rceil$. For each set of ABE parameters, Eq. 5 can be used to determine an upper bound on $\lceil \log_2 p_j \rceil$. Then after choosing p_j's as large as possible, we determine a minimum t with Eq. 6. Compared to the value of t, $\lceil \log_2 p_j \rceil$ has a very limited influence on performance as it affects the speed of $\mathsf{NTT}()$ (modulo p_j is performed at the end of $\mathsf{NTT}()$) only to a certain extent. However, as a multiplication requires $2t$ $\mathsf{NTT}()$'s and t $\mathsf{INTT}()$'s, increasing t from 3 to 4 reduces

TABLE II: Constraints on the CRT Primes' Size and Count Are Given by p_j and $\prod_{j=0}^{t-1} p_j$ (a.k.a. $\prod p_j$), Respectively.

	Prime q		Composite q	
	p_j	$\prod p_j$	p_j	$\prod p_j$
ENCRYPT	$\le \sqrt{P/2n}$	$> 2n(q-1)^2$	$\cancel{\le \sqrt{P/2n}}$	$= q$
EVALXX	$\le \sqrt{P/2mn}$	$> 2mn(q-1)$	$\le \sqrt{P/2mn}$	

performance by 33%. Therefore, we first pick as a small value as possible for t, then set $\lceil \log_2 p_j \rceil \approx \frac{m}{t}$.

To further take advantage of this, when a prime modulus is chosen and fixed, we generate different sets of CRT primes for encryption and evaluation, since evaluation obviously requires fewer number of primes. On the contrary, when generating a composite modulus as the product of CRT primes, we have to pick the prime size under the tighter bound given in the evaluation stage.

Using a Composite Modulus. We can alternatively select $q = \prod_{j=0}^{t-1} p_j$ to eliminate Eq. 6.

As shown in Table II, for a composite q, we have fewer number of CRT primes[1]. As a result, using fewer CRT primes (see Table V) simplifies the computation. Besides, in our benchmark circuit for ABE evaluation operation, we are able to avoid unnecessary CRT and ICRT operations, since we apply CRT only on circuit inputs and apply ICRT only to recover the final result. For both cases where q is either prime or composite, we compute the values of t and take timing results, which are listed in Table V. The timing results confirm our expectation.

Further Optimizations. We use the NTL library[2] to perform pre-computations needed for CRT conversions for both ABE encryption and evaluation, including p_j's, $M = \prod_{j=0}^{t-1} p_j$, $M_j = \frac{M}{p_j}$ and $[M_j^{-1}]_{p_j}$ for all $j \in \mathbb{Z}_t$. The precomputed values are, then, transferred to GPU and stored in its constant memory. Employing n threads, $\mathsf{CRT}()$ converts each vector in $\mathbb{Z}_q^n$ to t vectors by reducing them modulo p_j's. An ICRT kernel obtains vectors in $\mathbb{Z}_M^n$ and then reduces them to $\mathbb{Z}_q^N$ (if $M > q$) using Barrett reduction. All accesses to vectors in global memory are coalesced. Large integer arithmetic operations are specifically optimized for our parameters to provide the best performance.

The outputs of NAFDECOMP() skip the CRT conversions. We could perform CRT on each of the m polynomials in R_3 by mapping -1 coefficients to $p_j - 1$ for all $j \in \mathbb{Z}_t$, as in Eq. 4. But that would require space for $m \times t$ polynomials and perform $m \times t$ NTT conversions. Instead, we map -1 to $P - 1$ once for all $j \in \mathbb{Z}_t$ and perform NTT directly:

$$\mathbf{h}' = \mathsf{ICRT}_{j \in \mathbb{Z}_t} \left(\mathsf{INTT} \left(\sum_{i=0}^{\tau-1} \mathsf{NTT}\left(\tilde{\mathbf{f}}_i^{(j)}\right) \odot \mathsf{NTT}\left(\tilde{\mathbf{g}}_i\right) \right) \right), \tag{7}$$

where $\mathbf{g}_i \in R_3$. This requires only m polynomials and m NTT conversions.

[1]The size of CRT primes is determined by the Evaluate step rather than the Encrypt step, although the latter can use larger and fewer CRT primes, because q remains the same in both steps.

[2]http://www.shoup.net/ntl/

C. Minimizing Memory Consumption

Although we allocate linear memory on the host and the device to store polynomials, we create a data structure `Array3D_t` to access coefficients virtually as a 3D array. It contains a pointer `uint64_t *ptr` to the starting address and keeps its dimensions in a 3-tuple `uint3 dim`. Mathematically speaking, an `Array3D_t` element is in $\mathbb{Z}^{\texttt{dim.z}\times\texttt{dim.y}\times\texttt{dim.x}}$. Coefficients are addressed consecutively in memory by first `x`, then `y`, and at last `z` dimension, i.e. `ptr[idx.z][idx.y][idx.x]`. As `dim.x` is fixed to the ring dimension n, `idx.x` gives the index of a coefficient within the vector. Also, for `dim.z` vectors are packed, `idx.z`-th is the index of a vector in the pack. The data domains are described in Table III.

The particular design of data structures in our implementation ensures coalesced access to global memory, enables the launch of a huge kernel to keep all CUDA cores busy and is then able to overlap global memory accesses with GPU computation. For example, to perform bit-decomposition and NTT conversions, i.e. $\hat{\mathbf{\Psi}} \leftarrow$ NAFDECOMP_NTT$(-\mathbf{B}_{2i-1})$ in Algorithm 7, we launch a grid of dimension $(m, m, \frac{n}{512})$ with 64 threads per block ($m^2 \frac{n}{512}$ threads in total) to perform m^2 NTT conversions. Since the ring dimension is 1024, 2048 or 4096, the data structure naturally aligns global memory addresses. If we create an element for CRT+NTT domain, it has sufficient space for other domains (except NAFDECOMP+NTT) as well. CRT and NTT conversions read and write in the same `Array3D_t` element. This way we reduce memory consumption significantly.

D. Performance

We experimented with our implementation using an Nvidia GeForce Titan X graphics card with the Maxwell architecture ("Titan X" in short) as well as an Nvidia GeForce Titan Xp that

TABLE III: Usage of `Array3D_t` for `dim.z` $= \tau$ full vectors of length `dim.x` $= n$ in various domains. Integers are stored in `uint64_t` words. The word-length of q is then $s = \lceil \frac{k}{64} \rceil$ with word-base $b = 2^{64}$.

Domain	`dim.y`	Form	`ptr[z][y][x]`
Regular $\mathbb{Z}_q^N$	s	$\mathbb{Z}_b^{\tau\times s\times n}$	The `y`-th 64-bit word of the `x`-th coefficient of the `z`-th full vector.
CRT	t	$\mathbb{Z}^{\tau\times t\times N}$	The `x`-th coefficient of the `y`-th CRT vector from the `z`-th full vector (reduced modulo $p_{\texttt{y}}$).
CRT+NTT	t	$\mathbb{F}_P^{\tau\times t\times N}$	The `x`-th coefficient of the NTT vector from the `y`-th CRT vector from the `z`-th full vector.
BD+NTT	m	$\mathbb{F}_P^{\tau\times m\times n}$	The `x`-th coefficient of the NTT vector converted from the `y`-th bit-decomposed vector of the `z`-th plain vector (the `y`-th bit of each coefficient in $\mathbb{Z}_q$).

is built with the Pascal architecture ("Titan Xp" in short). Titan X was priced for about $1000 in early 2016 and Titan Xp has a similar price in 2017. The host computer also has an Intel Core i7-3770k processor with 4 cores running at 3.50 GHz and system memory (host memory) of 32 GB. The host runs Ubuntu 16.04 LTS and the programs are compiled with g++ 5.4.0 and CUDA compilation tools v8.0.44 (See Table IV).

TABLE IV: Experimental Environment

	Titan X	Titan Xp
CUDA cores	3072	3840
Base Clock	1.22 GHz	1.58 GHz
Memory Bandwidth	336.5 GB/s	547.7 GB/s
Memory Size	12 GB	12 GB

A Comparison of Polynomial Multiplication Speed. We approximate the latency of a full polynomial multiplication to the latency of two NTTs and one INTT to provide a fair comparison. We developed programs for both our method and the cuFFT-based method in [47], and measure the average latency of $256 \times 128 \times 4$ multiplications, with respect to [47]. We fix ring dimension as 2048 and assume coefficients are less than 24-bit. Our multiplication takes $0.34 \times 2 + 0.26 = 0.94$ μs on Titan X.

A cuFFT-based method requires 4096-point FFTs for such a ring dimension. The work in [47] compared several methods and concluded that a cuFFT-based multiplication is more efficient for $n \geq 2048$. Our sample cuFFT-based code shows that each multiplication takes 7.62 μs on Titan X. It turns out that a cuFFT based-method is 68 times slower than our integer based method.

The method in [41] does not involve negative wrapped convolution. For ring dimension 2048, it requires 4096-point FFTs whose performance was not reported. Based on its complexity $n \log_2 n$, we estimate that 4096-point FFTs are 2.18 slower than 2048-point FFTs. Using the fourth bar of the third subplot of Figure V in [41], we can conclude that each multiplication would take $2.18 \times 40\ ms \div (256 \times 128) \div 4 \approx 0.665\ \mu s$. They adopt an Nvidia GeForce GTX 280 graphics card which has a peak performance of 933 GFlops/s. According to their calculation, they reach 444 GFlops/s. If their code ran on out device and (not likely) reached the peak 192 GFlops/s performance, it would take $\frac{444}{192} \approx 2.31$ times longer, that is, $0.665 \times 2.31 \approx 1.54$ μs, which is 64% slower than ours. If we consider their GPU to be $\frac{933}{192} \approx 4.86$ times more powerful than ours in term of peak GFlops/s, their code would take $0.665 \times 4.86 \approx 3.23$ μs, which is 3.44 times slower than ours. Based on these comparisons, our GPU implementation for multiplication in $\mathcal{R}_q$ compares favorably with all other implementations reported in the literature.

Performance of ABE Encryption and Evaluation. We enumerated timing results together with parameter selections in Table V. These parameters are selected based on our security and correctness analysis in Section IX in Appendix. We provide two sets of measurements based on the configuration whereby either a prime or a composite q is used. The number of CRT primes generated is listed under column "t" for each scenario. "Time", "Fixed" and "Changing" show timing results in milliseconds. "Fixed" assumes that the circuit has a fixed policy (EVALCT only) while "Changing" assumes a changing policy (EVALCT+EVALPK). From these two tables, Titan Xp yields roughly 1.6 times speedup over Titan X, due to the fact that Titan Xp has 1.6 times faster memory bandwidth.

The measurements of ABE encryption ("Enc") include all steps but the sampling of $s \leftarrow_U \mathcal{R}_q$ in Algorithm 4. Discrete Gaussian noise is sampled on the device while s is sampled on the host and transfered to the device. The output $\mathbf{C}_{\text{in}}$ is transfered from the device to the host.

The measurements of ABE evaluation ("Fixed") assume that $\mathbf{B}_i$'s for all $i \in [l{+}1,\ 2l{-}1]$ are pre-computed. In other words, we skip the step, where $\mathbf{B}_{\ell+i} \leftarrow \mathbf{B}_0 - \mathbf{B}_{2i}\mathbf{\Psi}_i$ is computed in Algorithm 7. This is a reasonable assumption when the policy circuit is fixed. Memory consumption on the device or on the host is similar, since we either store $\{\mathbf{B}_0, \mathbf{B}_1, \ldots \mathbf{B}_l\}$ or pre-compute and store $\{\mathbf{B}_1, \mathbf{B}_3, \ldots, \mathbf{B}_{2l-1}\}$. Computing $\mathbf{B}_i$'s has the same cost as computing $\mathbf{C}_i$'s. To include the computation of $\mathbf{B}_i$'s for a changing policy circuit, we present the time cost under column "Changing". In each gate of ABE evaluation, we transfer $\mathbf{B}_i$'s and $\mathbf{C}_i$'s required by this gate from the host, perform arithmetic operations in the device and send back a single vector $\mathbf{C}_i$ to the host.

Although memory transfers between host memory and device memory occur at run time, their latency is hidden behind computation. This is achieved by creating CUDA streams to pipeline data transfers and computation tasks. Timing results in Table V has no overhead caused by data transfer.

We also consider the implementation scenario where not only $\mathbf{B}_i$'s are pre-computed for each gate, but also are their bit-decomposed forms $\mathbf{\Psi}_i$'s converted to NTT domain as $\hat{\mathbf{\Psi}}_i$'s. By doing this, we exclude m^2 forward NTT conversions in each gate. However, memory consumption increases by roughly 64 times since each bit in $\mathbf{B}_i$ is now a 64-bit integer in $\mathbb{F}_P$. As storing $\hat{\mathbf{\Psi}}_i$ in host memory requires transferring them to the device memory, which in fact is slower than m^2 NTT conversions, they are therefore stored on device memory. As we run out of device memory for a small number of attributes (i.e. 32), we exclude the timing results in this scenario. But its performance can be accurately estimated as $2 \times T_{\text{Fixed}} - T_{\text{Changing}}$ from the timings listed under those two columns. For example, potentially the evaluation for 16 or 32 attributes would take only 13.6 ms or 91.8 ms, respectively.

To give an idea of the magnitude and complexity of computations, one can consider that an ABE evaluation operation requires about $(l-1)m^2$ polynomial multiplications and additions in $\mathcal{R}_q$. Our GPU implementation achieves a very high throughput. For instance, for 1024 attributes (ring dimension is 4096) with a prime q, we achieve less than 2.21 μs (Titan X) or 1.33 μs (Titan Xp) per multiplication and accumulation in $\mathcal{R}_q$; for 16 attributes (ring dimension is 2048) with a prime q, it takes less than 0.82 μs (on Titan X) or 0.51 μs (on Titan Xp).

The impact of choosing a composite q can be captured by comparing the measurements under "Prime q" and "Composite q" sections of Table V. CRT and ICRT do not weigh much in the computation of ABE evaluation. Although we eliminate $(\frac{l}{2} - 1)$ CRTs and $(l-2)$ ICRTs by choosing a composite

TABLE V: Performance of selected parameters on Titan X / Titan Xp.

Parameters			Prime q					Composite q				
l	k	n	ENCRYPT		EVALCT + EVALPK			ENCRYPT		EVALCT + EVALPK		
			t	Time (ms)	t	Changing (ms)	Fixed (ms)	t	Time (ms)	t	Changing (ms)	Fixed (ms)
2	36	1024	4	1.10 / 0.76	3	0.62 /0.41	0.34 /0.23	2	0.77 / 0.57	2	0.53 / 0.36	0.26 / 0.19
4	51	2048	5	4.28 / 2.85	3	6.48 /3.76	2.76 /1.70	3	2.60 / 1.71	3	6.33 / 3.94	2.62 / 1.62
8	60	2048	6	9.31 / 6.28	4	22.3 /13.6	10.5 / 6.55	3	4.98 /3.51	3	19.5 / 12.0	7.90 / 4.91
16	69	2048	6	20.1 /13.3	4	61.9 /38.2	25.8 / 17.6	3	11.2 / 7.79	3	55.5 / 35.1	21.3 / 13.6
32	82	4096	7	106 / 67.5	5	419 / 264	188 /112	4	60.6 /39.7	4	386 / 245	152 / 91.8
64	92	4096	8	232 / 137	6	1,113 / 642	461 / 271	4	119 /72.0	4	957 / 594	377 / 224
128	102	4096	9	614 / 354	6	2,681 /1,668	1,253 / 749	5	339 / 196	5	2,521 / 1,555	1,078 / 643
256	112	4096	10	1,459 / 834	6	6,477 / 3,860	2,961 /1,697	5	745 / 424	5	6,015 / 3,579	2,495 / 1,437
512	122	4096	11	3,627 / 2.027	7	16,762 /10,183	8,472 /4,928	6	2,014 / 1,084	6	15,177 / 9,149	6,879 / 3,971
1024	132	4096	11	8,387 / 4,705	7	40,570 / 24,503	19,948 / 11,455	6	4,749 / 2,565	6	36,671 / 22,396	16,301 / 9,411

q, for most parameter sets we gain little benefits. However, a composite q requires a smaller t compared to a prime q (i.e. performance of all ABE evaluations and all ABE encryptions), which provides some improvement.

A scheme with up to 1024 attributes is supported by our implementation and this number will be larger with more system memory. The performance results are very promising considering the fact that our GPU now costs around or below \$1000. A more advanced GPU definitely will yield better performance.

A quick comparison of the execution times in Table I (CPU) and Table V (GPU) shows that our GPU implementation of KP-ABE encryption operation at least $259/9.31 = 27.8$ times faster for 8 attributes than CPU implementation whereas the acceleration ratio can be as high as 73.8. For homomorphic evaluation operations with 8 attributes the acceleration ratios will be at least 151 and as high as 685.

VIII. CONCLUSION

We present a construction and implementation of the first RLWE KP-ABE scheme and experimentally demonstrate that it can be efficiently implemented by leveraging commercial-off-the-shelf compute resources, notably a moderately priced GPU. Since the key ABE operations require numerous polynomial multiplications amenable to parallel computations, we focus on improving their throughput. To this end, we develop special-purpose algorithms and data structures to optimize memory access. A comparison with previous works shows our polynomial multiplication with ring dimension $N = 2048$ is at least 64% faster than the fastest implementation reported in the literature.

Our KP-ABE scheme requires highly expensive homomorphic operations over public keys and ciphertext. However, despite these perceived challenges, our implementation yields highly favorable timing results. We show that the most time-consuming ABE evaluation operation can be performed in as low as 13.6 ms and 91.8 ms, for 16 and 32 attributes, respectively. These runtime results would be even smaller, and scale to a larger number of attributes, with newer, increasingly more capable GPUs. The fact that our implementation supports up to 1024 attributes is very promising for the efficient implementation of more advanced cryptographic algorithms, which require ABE as a building block, such as functional encryption and token-based program obfuscation.

ACKNOWLEDGMENT

We would like to gratefully acknowledge the input and feedback from Vinod Vaikuntanathan.

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research.

Dai, Doröz and Sunar's work was in part provided by the US National Science Foundation CNS Award #1561536.

Polyakov, Rohloff, Sajjadpour and Savaş's sponsorships are as follows: Sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Army Research Laboratory (ARL) under Contract Numbers W911NF-15-C-0226 and W911NF-15-C-0233. The views expressed are those of the authors and do not necessarily reflect the official policy or position of the Department of Defense or the U.S. Government. Project sponsored by the National Security Agency under Grant H98230-15-1-0274. This research is based upon work supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either express or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

REFERENCES

[1] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Advances in Cryptology - EUROCRYPT 2005, Aarhus, Denmark, May 22-26, 2005, Proceedings*, ser. LNCS, R. Cramer, Ed., vol. 3494. Springer, 2005, pp. 457–473.

[2] D. Boneh and M. K. Franklin, "Identity-based encryption from the weil pairing," *SIAM J. Comput.*, vol. 32, no. 3, pp. 586–615, 2003.

[3] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of CCS'06*. New York, NY, USA: ACM, 2006, pp. 89–98.

[4] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE Symposium on Security and Privacy (SP '07)*, May 2007, pp. 321–334.

[5] J. Zhang, Z. Zhang, and A. Ge, "Ciphertext policy attribute-based encryption from lattices," in *ASIACCS '12, Seoul, Korea, May 2-4, 2012*, H. Y. Youm and Y. Won, Eds. ACM, 2012, pp. 16–17. [Online]. Available: http://dl.acm.org/citation.cfm?id=2414456

[6] B. Waters, “Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization,” in *PKC 2011 - Taormina, Italy, March 6-9, 2011. Proceedings*, ser. LNCS, D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, Eds., vol. 6571. Springer, 2011, pp. 53–70.

[7] H. Deng, Q. Wu, B. Qin, J. Domingo-Ferrer, L. Zhang, J. Liu, and W. Shi, “Ciphertext-policy hierarchical attribute-based encryption with short ciphertexts,” *Inf. Sci.*, vol. 275, pp. 370–384, 2014.

[8] E. Zavattoni, L. J. D. Perez, S. Mitsunari, A. H. Sánchez-Ramírez, T. Teruya, and F. Rodríguez-Henríquez, “Software implementation of an attribute-based encryption scheme,” *IEEE Trans. Computers*, vol. 64, no. 5, pp. 1429–1441, 2015.

[9] R. Ostrovsky, A. Sahai, and B. Waters, “Attribute-based encryption with non-monotonic access structures,” in *Proceedings of CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, P. Ning, S. D. C. di Vimercati, and P. F. Syverson, Eds. ACM, 2007, pp. 195–203.

[10] V. Goyal, A. Jain, O. Pandey, and A. Sahai, “Bounded ciphertext policy attribute based encryption,” in *ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings,*, ser. LNCS, L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, Eds., vol. 5126. Springer, 2008, pp. 579–591.

[11] A. B. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, “Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption,” in *EUROCRYPT 2010, French Riviera, May 30 - June 3, 2010. Proceedings*, ser. LNCS, H. Gilbert, Ed., vol. 6110. Springer, 2010, pp. 62–91.

[12] A. B. Lewko and B. Waters, “Decentralizing attribute-based encryption,” in *EUROCRYPT 2011, Tallinn, Estonia, May 15-19, 2011. Proceedings*, ser. LNCS, K. G. Paterson, Ed., vol. 6632. Springer, 2011, pp. 568–588.

[13] A. H. Sánchez and F. Rodríguez-Henríquez, “NEON implementation of an attribute-based encryption scheme,” in *ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings*, ser. LNCS, M. J. J. Jr., M. E. Locasto, P. Mohassel, and R. Safavi-Naini, Eds., vol. 7954. Springer, 2013, pp. 322–338.

[14] C. Gentry, C. Peikert, and V. Vaikuntanathan, “Trapdoors for hard lattices and new cryptographic constructions,” in *STOC*, 2008, pp. 197–206.

[15] M. Ajtai, “Generating hard instances of lattice problems,” *Quaderni di Matematica*, vol. 13, pp. 1–32, 2004, preliminary version in STOC 1996.

[16] ——, “Generating hard instances of the short basis problem,” in *ICALP*, 1999, pp. 1–9.

[17] O. Goldreich, S. Goldwasser, and S. Halevi, “Public-key cryptosystems from lattice reduction problems,” in *Advances in Cryptology–CRYPTO’97*. Springer, 1997, pp. 112–131.

[18] J. Zhang and Z. Zhang, “A ciphertext policy attribute-based encryption scheme without pairings,” in *Inscrypt 2011, Beijing, China, November 30 - December 3, 2011*, ser. LNCS, C. Wu, M. Yung, and D. Lin, Eds., vol. 7537. Springer, 2011, pp. 324–340.

[19] D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy, “Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits,” in *EUROCRYPT 2014, Denmark, May 11-15, 2014. Proceedings*, ser. LNCS, P. Q. Nguyen and E. Oswald, Eds., vol. 8441. Springer, 2014, pp. 533–556.

[20] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich, “Reusable garbled circuits and succinct functional encryption,” in *STOC’13, Palo Alto, CA, USA, June 1-4, 2013*, D. Boneh, T. Roughgarden, and J. Feigenbaum, Eds. ACM, 2013, pp. 555–564.

[21] N. Bitansky and V. Vaikuntanathan, “Indistinguishability obfuscation from functional encryption,” in *IEEE FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, V. Guruswami, Ed. IEEE Computer Society, 2015, pp. 171–190.

[22] O. Regev, “On lattices, learning with errors, random linear codes, and cryptography,” *J. ACM*, vol. 56, no. 6, pp. 1–40, 2009, preliminary version in STOC 2005.

[23] R. E. Bansarkhani and J. A. Buchmann, “Improvement and efficient implementation of a lattice-based signature scheme,” in *- SAC 2013, Burnaby, BC, Canada, August 14-16, 2013*, ser. LNCS, T. Lange, K. E. Lauter, and P. Lisonek, Eds., vol. 8282. Springer, 2013, pp. 48–67.

[24] V. Lyubashevsky, C. Peikert, and O. Regev, “On ideal lattices and learning with errors over rings,” in *EUROCRYPT*, 2010, pp. 1–23.

[25] ——, “A toolkit for ring-lwe cryptography.” in *EUROCRYPT*, vol. 7881. Springer, 2013, pp. 35–54.

[26] S. Gorbunov, V. Vaikuntanathan, and H. Wee, “Attribute-based encryption for circuits,” in *STOC’13, Palo Alto, CA, USA, June 1-4, 2013*, D. Boneh, T. Roughgarden, and J. Feigenbaum, Eds. ACM, 2013, pp. 545–554.

[27] S. Garg, C. Gentry, S. Halevi, A. Sahai, and B. Waters, “Attribute-based encryption for circuits from multilinear maps,” in *CRYPTO 2013, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings*, ser. LNCS, R. Canetti and J. A. Garay, Eds., vol. 8043. Springer, 2013, pp. 479–499.

[28] D. Boneh and A. Silverberg, “Applications of multilinear forms to cryptography,” *IACR Cryptology ePrint Archive*, vol. 2002, p. 80, 2002.

[29] M. Scott, “On the efficient implementation of pairing-based protocols,” in *IMACC 2011, Oxford, UK, December 12-15, 2011. Proceedings*, ser. LNCS, L. Chen, Ed., vol. 7089. Springer, 2011, pp. 296–308.

[30] D. Micciancio and C. Peikert, “Trapdoors for lattices: Simpler, tighter, faster, smaller,” in *EUROCRYPT*, 2012, pp. 700–718.

[31] L. Ducas and P. Q. Nguyen, “Faster Gaussian lattice sampling using lazy floating-point arithmetic,” in *ASIACRYPT 2012, Beijing, China, December 2-6, 2012. Proceedings*, ser. LNCS, X. Wang and K. Sako, Eds., vol. 7658. Springer, 2012, pp. 415–432.

[32] N. Genise and D. Micciancio, “Faster gaussian sampling for trapdoor lattices with arbitrary modulus,” *IACR Cryptology ePrint Archive*, vol. 2017, p. 308, 2017.

[33] K. D. Gur, Y. Polyakov, K. Rohloff, G. W. Ryan, and E. Savas, “Implementation and evaluation of improved gaussian sampling for lattice trapdoors,” *IACR Cryptology ePrint Archive*, vol. 2017, p. 285, 2017.

[34] S. Gorbunov, V. Vaikuntanathan, and H. Wee, “Attribute-based encryption for circuits,” *J. ACM*, vol. 62, no. 6, pp. 45:1–45:33, 2015.

[35] ——, “Predicate encryption for circuits from LWE,” in *CRYPTO 2015, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings*, ser. LNCS, R. Gennaro and M. Robshaw, Eds., vol. 9216. Springer, 2015, pp. 503–523.

[36] C. Gentry, S. Halevi, and N. P. Smart, “Homomorphic evaluation of the AES circuit,” in *CRYPTO*, 2012, pp. 850–867.

[37] Y. Chen and P. Q. Nguyen, “BKZ 2.0: Better lattice security estimates,” in *ASIACRYPT*, 2011, pp. 1–20.

[38] R. Lindner and C. Peikert, “Better key sizes (and attacks) for LWE-based encryption,” in *CT-RSA*, 2011, pp. 319–339.

[39] G. Hanrot and D. Stehlé, “Worst-case hermite-korkine-zolotarev reduced lattice bases,” *CoRR*, vol. abs/0801.3331, 2008. [Online]. Available: http://arxiv.org/abs/0801.3331

[40] D. D. Chen, N. Mentens, F. Vercauteren, S. S. Roy, R. C. C. Cheung, D. Pao, and I. Verbauwhede, “High-speed polynomial multiplication architecture for Ring-LWE and SHE cryptosystems,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 1, pp. 157–166, Jan 2015.

[41] P. Emeliyanenko, “Efficient multiplication of polynomials on graphics hardware,” in *Proceedings of the 8th International Symposium on Advanced Parallel Processing Technologies*, ser. APPT ’09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 134–149. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03644-6_11

[42] W. Dai and B. Sunar, “cuHE: A homomorphic encryption accelerator library,” in *Cryptography and Information Security in the Balkans - Second International Conference, BalkanCryptSec 2015, Koper, Slovenia, September 3-4, 2015, Revised Selected Papers*, ser. Lecture Notes in Computer Science, E. Pasalic and L. R. Knudsen, Eds., vol. 9540. Springer, 2015, pp. 169–186. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-29172-7_11

[43] Y. Polyakov, K. Rohloff, G. Sahu, and V. Vaikuntanthan, “Fast proxy re-encryption for publish/subscribe systems,” *IACR Cryptology ePrint Archive*, vol. 2017, p. 410, 2017. [Online]. Available: http://eprint.iacr.org/2017/410

[44] C. Borcea, A. D. Gupta, Y. Polyakov, K. Rohloff, and G. W. Ryan, “PICADOR: end-to-end encrypted publish-subscribe information distribution with proxy re-encryption,” *Future Generation Comp. Syst.*, vol. 71, pp. 177–191, 2017.

[45] G. S. Çetin, W. Dai, Y. Doröz, W. J. Martin, and B. Sunar, “Blind web search: How far are we from a privacy preserving search engine?” Cryptology ePrint Archive, Report 2016/801, 2016, http://eprint.iacr.org/2016/801. [Online]. Available: http://eprint.iacr.org/2016/801

[46] W. Dai, Y. Doröz, and B. Sunar, “Accelerating NTRU based homomorphic encryption using GPUs,” in *High Performance Extreme Computing Conference (HPEC), 2014 IEEE*, Sept 2014, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/7041001/?arnumber=7041001&tag=1

[47] S. Akleylek, Ö. Dağdelen, and Z. Yüce Tok, *On the Efficiency of Polynomial Multiplication for Lattice-Based Cryptography on GPUs Using CUDA*. Cham: Springer International Publishing, 2016, pp. 155–168. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-29172-7_10

[48] M. S. Lee, Y. Lee, J. H. Cheon, and Y. Paek, "Accelerating bootstrapping in FHEW using GPUs," in *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, July 2015, pp. 128–135.
[49] N. Emmart and C. C. Weems, "High precision integer multiplication with a GPU using Strassen's algorithm with multiple FFT sizes," *Parallel Processing Letters*, vol. 21, no. 03, pp. 359–375, 2011. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S0129626411000266
[50] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.
[51] M. R. Albrecht, "On dual lattice attacks against small-secret LWE and parameter choices in helib and SEAL," in *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, ser. LNCS, J. Coron and J. B. Nielsen, Eds., vol. 10211, 2017, pp. 103–129. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-56614-6_4
[52] D. Micciancio, "Generalized compact knapsacks, cyclic lattices, and efficient one-way functions," *Computational Complexity*, vol. 16, no. 4, pp. 365–411, 2007, preliminary version in FOCS 2002.
[53] D. Stehlé, R. Steinfeld, K. Tanaka, and K. Xagawa, "Efficient public key encryption based on ideal lattices," in *ASIACRYPT*, 2009, pp. 617–635.
[54] D. Boneh, T. Roughgarden, and J. Feigenbaum, Eds., *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013.* ACM, 2013.

APPENDIX

A. Evaluating NAND Gate

Logical NAND gate is a universal gate in the sense that any logic circuit can be realized using only NAND gates. Analyzing NAND gates, therefore, is extremely important for our benchmarks as we use NAND gates-only circuits with a tree structure.

NAND gate can be obtained using one AND gate and an arithmetic subtraction operation, namely $(xy)' = 1 - xy$. Consequently, the circuit evaluation is only slightly different than that of AND gate

$$\begin{aligned}\mathbf{C}_{NAND} &= \mathbf{C}_0 - x_2\mathbf{C}_1 - \mathbf{\Psi}_1^T\mathbf{C}_2 \\ \mathbf{B}_{NAND} &= \mathbf{B}_0 - \mathbf{B}_2\mathbf{\Psi}_1,\end{aligned} \tag{8}$$

where $\mathbf{C}_0$ is the encryption of logical-1, i.e., $\mathbf{C}_0 = (\mathbf{G} + \mathbf{B}_0)^T s + \mathbf{e}_{0,0}$. The operations on the noise term is similar $\mathbf{e}_{0,NAND} = \mathbf{e}_{0,0} - x_2\mathbf{e}_{0,1} - \mathbf{\Psi}_1^T\mathbf{e}_{0,2}$. Any other logical gate can be obtained in a similar manner. For instance, $x \oplus y = x + y - 2xy$ and $x \vee y = x + y - xy$ for binary variables x and y.

B. Evaluating a Simple Benchmark Circuit of NAND Gates

Suppose a tree-like circuit of NAND gates in Figure 2 with four attributes, x_i for $i = 1, 2, 3, 4$. Then the ciphertext for a message $\mu \in \mathcal{R}_2$ is as follows

$$\mathbf{C}_{\text{in}} = (\mathbf{A}|(\mathbf{G}+\mathbf{B}_0)|(x_1\mathbf{G}+\mathbf{B}_1)|\dots|(x_4\mathbf{G}+\mathbf{B}_4))^T s + \mathbf{e}_0,$$

where c_1 is the same as before (henceforth we will omit it from our discussion for it is not affected by the evaluation process). We now explain how the evaluation is performed using a simple circuit illustrated in Fig.2.

The ciphertexts and public keys to be evaluated are $\mathbf{C}_i = (x_i\mathbf{G} + \mathbf{B}_i)^T s + \mathbf{e}_{0,i}$ and B_i for $i = 1, 2, 3, 4$, respectively, where the corresponding inputs in the circuit

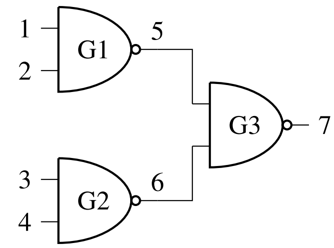

Fig. 2: A tree like NAND circuit with four attributes

(Fig.2) are labeled with the index number i. The following are the steps of the evaluation process:

1) Evaluation of gate G1
$$\begin{aligned}\mathbf{\Psi}_1 &= \text{NAFD}\textsc{ecomp}(-\mathbf{B}_1) \\ \mathbf{C}_5 &= \mathbf{C}_0 - y_2\mathbf{C}_1 - \mathbf{\Psi}_1^T\mathbf{C}_2 \\ \mathbf{B}_5 &= \mathbf{B}_0 - \mathbf{B}_2\mathbf{\Psi}_1 \\ y_5 &= 1 - y_1y_2\end{aligned}$$
2) Evaluation of gate G2
$$\begin{aligned}\mathbf{\Psi}_2 &= \text{NAFD}\textsc{ecomp}(-\mathbf{B}_3) \\ \mathbf{C}_6 &= \mathbf{C}_0 - y_4\mathbf{C}_3 - \mathbf{\Psi}_2^T\mathbf{C}_4 \\ \mathbf{B}_6 &= \mathbf{B}_0 - \mathbf{B}_4\mathbf{\Psi}_2 \\ y_6 &= 1 - y_3y_4\end{aligned}$$
3) Evaluation of gate G3
$$\begin{aligned}\mathbf{\Psi}_3 &= \text{NAFD}\textsc{ecomp}(-\mathbf{B}_5) \\ \mathbf{C}_7 &= \mathbf{C}_0 - y_6\mathbf{C}_5 - \mathbf{\Psi}_3^T\mathbf{C}_6 \\ \mathbf{B}_7 &= \mathbf{B}_0 - \mathbf{B}_6\mathbf{\Psi}_3 \\ y_7 &= 1 - y_5y_6,\end{aligned}$$

where $\mathbf{\Psi}_i = \text{NAFD}\textsc{ecomp}(-\mathbf{B}_{2i-1})$ for $i = 1, 2, 3$. We can also write $\mathbf{B}_f = \mathbf{B}_7$. Note that y_is are binary values used during the evaluations and $y_7 = 0$ following the construction of the circuit for a given access policy. We change the notation for the attributes used during the evaluation phase from x_i to y_i for they may not be always identical. However, the decryption works only if $x_i = y_i$ for $i = 1, \dots, \ell$. For the noise terms we can obtain the following expressions:

$$\begin{aligned}\mathbf{e}_{0,5} &= \mathbf{e}_{0,0} - x_2\mathbf{e}_{0,1} - \mathbf{\Psi}_1^T\mathbf{e}_{0,2} \\ \mathbf{e}_{0,6} &= \mathbf{e}_{0,0} - x_4\mathbf{e}_{0,3} - \mathbf{\Psi}_2^T\mathbf{e}_{0,4} \\ \mathbf{e}_{0,7} &= \mathbf{e}_{0,0} - x_6\mathbf{e}_{0,5} - \mathbf{\Psi}_3^T\mathbf{e}_{0,6}\end{aligned}$$

IX. CORRECTNESS AND SECURITY CONSTRAINTS OF A POLICY CIRCUIT

The depth of a policy circuit can be defined as the number of AND gates (or NAND gates in our benchmark circuit) in cascade on its longest path from input to output. Therefore, as already discussed in Section VI-B the dominating contributor to the noise growth is the multiplication of noise vector $\mathbf{e}$ with bit decomposition matrix $\mathbf{\Psi} = \text{B}\textsc{it}\text{D}\textsc{ecomp}(-\mathbf{B})$, namely $\mathbf{\Psi}^T\mathbf{e}$, in every level of the circuit. The norm of the error vector at the output of the circuit must be less than $\frac{q}{4}$ for correct decryption. Before analyzing the effect of this multiplication on the noise growth, basic properties of arithmetic on random variables are recalled.

Suppose the mean and standard deviation of two independent random variables x and y are (ω_x, σ_x) and (ω_y, σ_y), respectively. We can write the following expressions for the mean and standard deviation of $z = x + y$ and $v = xy$

$$(\omega_z, \sigma_z) = \left(\omega_x + \omega_y, \sqrt{\sigma_x^2 + \sigma_y^2}\right)$$
$$(\omega_v, \sigma_v) = \left(\omega_x\omega_y, \sqrt{\sigma_x^2\sigma_y^2 + \sigma_x^2\omega_y^2 + \omega_x^2\sigma_y^2}\right). \tag{9}$$

Let (ω_e, σ_e) represent a Gaussian distribution, from which the polynomial coefficients in the error vectors are sampled in the encryption operation. Ideally, $\omega_e = 0$ as the random number generator (RNG) used in encryption implements a zero-centered Gaussian distribution. Notwithstanding, in an actual implementation of the Gaussian RNG this may not be the case, resulting in a relatively small, but nonzero, average value for a limited number of samples. As will be shown below, the noise growth in the error vector can be highly sensitive to this initial small non-zero mean. As the public vector $\mathbf{B}_i$ is sampled from a uniform distribution in ABE setup, we can easily assume that the polynomial coefficients in bit decomposition matrix $\mathbf{\Psi}$ are sampled from a binary uniform distribution with $(\omega_\psi, \sigma_\psi) = (0.5, 0.5)$.

The operation $\mathbf{e}_{new} = \mathbf{\Psi}^T\mathbf{e}$ consists of polynomial multiplications followed by polynomial additions. For instance, $e_{new,i} = \sum_{j=0}^{m-1} e_j\psi_{j,i}$ contains arithmetic operations on random variables in three levels. In the first level, the coefficients of the error and bit decomposition polynomials are multiplied. Consequently, the mean and standard deviation of the resulting integers, (ω_1, σ_1), can be computed using Eq. 9. As mentioned previously, since ω_e is non-zero in practical implementations, ω_1 is also non-zero, whose value will be amplified significantly by the subsequent addition operations.

In the second level, integers with (ω_1, σ_1) are summed to compute the coefficients of each polynomial, $e_j\psi_{j,i}$ for $i, j = 0, \ldots, m-1$. One coefficient of the resulting polynomial is the addition of n random integers with (ω_1, σ_1). Then the coefficients are distributed with $(\omega_2, \sigma_2) = (n\omega_1, \sqrt{n}\sigma_1)$ by Eq. 9. Finally in the third level, we sum m polynomials (recall $\sum_{j=0}^{m-1} e_j\psi_{j,i}$). Consequently, the coefficients in the resulting vector of polynomials are distributed with

$$(\omega_3, \sigma_3) = \left(mn\omega_e\omega_\psi, \sqrt{mn(\sigma_e^2\sigma_\psi^2 + \sigma_e^2\omega_\psi^2 + \omega_e^2\sigma_\psi^2)}\right) \tag{10}$$

Although ω_e can be very small, Eq. 10 clearly indicates that the mean can grow faster than the standard deviation of the error vector as demonstrated with the following example.

Example 9.1: Suppose our Gaussian random number generator has a small mean value $\omega_e = 2^{-8}$ with $\sigma_e = 4.57825$. Suppose also that $n = 4096$, $k = \lceil \log_2 q \rceil = 89$ and the depth of the policy circuit is 4. Table VI lists the estimated values of mean and standard deviation for each level of the circuit.

Example 9.1 clearly shows that the error vectors at the output of the circuit can have very large mean values that dominate the noise growth. Having a better Gaussian RNG would not greatly be useful for alleviating the mean growth problem. For instance, with a really small mean value such as $\omega_e = 2^{-20}$, the mean and standard deviation of the noise at the output of the circuit in Example 9.1 will be 50.03 and 40.78, respectively. Therefore, we need to accept a small nonzero mean in our Gaussian RNG as natural and perform our noise analysis accordingly.

TABLE VI: Estimates for the noise growth in a policy circuit of depth 4 using regular bit and NAF decomposition algorithm.

Level	$\lceil \log_2 \omega_e \rceil$	$\lceil \log_2 \sigma_e \rceil$
0	-8 / -8	2.19 / 2.19
1	9.51 / 2.51	10.95 / 10.66
2	27.02 / 13.02	19.75 / 19.13
3	44.52 / 23.52	35.27 / 27.60
4	62.03 / 34.03	52.78 / 36.07

There is, however, one method that can suppress the growth in noise significantly. Non-adjacent form (NAF) representation of integers proves to be extremely useful in reducing the increase in noise and consequently in improving the performance of all ABE operations by allowing to work with smaller moduli. In NAF representation of an integer, which includes -1 in addition to 0 and 1, only one third of the digits are non-zero, on average. Furthermore, the expected numbers of 1 and -1 are equal to each other (hence, NAF is *balanced*). As a result, if the bit decomposition operation is performed using NAF (i.e., using NAFDECOMP), the bit decomposition matrix $\mathbf{\Psi}$ will consist of polynomials whose coefficients are uniformly random in the set $\{-1, 0, 1\}$ with $\omega_\psi = 0, \sigma_\psi \approx 0.58$. Naturally, by the same argument ω_ψ is expected to be non-zero, in practice. The following example shows that using NAF decomposition helps limit the noise growth.

Example 9.2: Suppose our Gaussian random number generator has a small mean value $\omega_e = 2^{-8}$ with $\sigma_e = 4.57825$. Suppose also that $n = 4096$, $k = \lceil \log_2 q \rceil = 89$ and the depth of the policy circuit is 4. Assuming a small nonzero mean in our uniform RNG $\omega_\psi = 2^{-8}$, Table VI lists the estimated values of mean and standard deviation for each level of the circuit (to the right of symbol "/"). As can be observed from the table, the mean is no longer the dominant factor in the noise growth.

Since the noise growth in the ciphertext is now understood, we can find a practical upper bound for the coefficients in the error vectors. Assuming a Gaussian RNG with standard deviation σ, the probability of sampling a value larger than $\sigma\sqrt{\epsilon}$ is $2^{-\epsilon}$. In practice, this probability is considered to be negligible for $\epsilon = 128$. Consequently, assuming the error vector at the output of the policy circuit, $\mathbf{e}_f$, is distributed with (ω_f, σ_f), a practical upper bound for the norm of the output noise can be estimated as $\Delta_f = \omega_f + \sigma_f \cdot \sqrt{\epsilon}$

The above analysis accounts for only the noise growth in the ciphertext after the ciphertext is evaluated over the policy circuit. However, the ABE decryption operation (where dual Regev scheme is used) also increases the noise in the ciphertext. For the dual Regev scheme to decrypt correctly, absolute values of the coefficients of the polynomial $\bar{\mu}$ should be smaller than the modulus $\frac{q}{4}$, i.e. $||\bar{\mu}|| < \frac{q}{4}$.

By the dual Regev encryption scheme, we can write for the decrypted message

$$\bar{\mu} = \mu\lceil\frac{q}{2}\rceil + e_1 - \boldsymbol{\alpha}_f^T \mathbf{e}_f, \tag{11}$$

where e_1 is the error introduced in ABE encryption while $\mathbf{e}_f$ represents the error term in the ciphertext after the homomorphic evaluation over the policy circuit, whose norm is bounded by Δ_f as shown in the previous section. The term $\boldsymbol{\alpha}_f^T \mathbf{e}_f$ stands for ring multiplications followed by polynomial additions.

The secret key, $\boldsymbol{\alpha}_f$, a vector of ring elements, is generated as a result of the Gaussian sampling operation in Algorithm 2. Therefore, its norm is also determined by the same process, which yields a small norm solution to $(\mathbf{A}|\mathbf{B}_f)\boldsymbol{\alpha}_f = \beta_f$. An upper bound (*spectral bound* henceforth) for the small norm solution $\boldsymbol{\alpha}_f$ can be formulated as

$$\Delta_\alpha = c \cdot \chi, \tag{12}$$

where $\chi = \sigma^2(\sqrt{nk} + \sqrt{2n} + 4.7)$ and c stands for the empirically obtained constant (e.g., $c = 1.8$). For more information on trapdoor generation and Gaussian sampling operations, the interested reader is referred to [33].

Then, the final formula of a practical upper bound for the error term in the decrypted message can be given as

$$\Delta_\mu = \sqrt{mn} \cdot \Delta_f \cdot \Delta_\alpha. \tag{13}$$

Naturally as we must have $\Delta_\mu < \frac{q}{4}$ for correct decryption $q > 4\Delta_\mu$.

For security we adopt the following formula for the ring dimension n,

$$n > \frac{\log_2 \frac{q}{\sigma}}{4 \cdot \log_2 \delta}, \tag{14}$$

where δ is the root Hermite factor. For $\delta < 1.006$ we assume the underlying RLWE problem is hard providing sufficient level of security. As we do not use low norm or sparse secret keys, a case whose security is analyzed in [51], we use the security argument provided in [37], [38] that the root Hermite factor is the major factor to determine the security. The formula given in [38] for the running time of the BKZ algorithm [39]

$$t_{BKZ} = \frac{1.8}{\log_2(\delta)} - 110 \tag{15}$$

suggests that $\delta \approx 1.006$ provides about 100-bit security.

ABE parameters for various number of attributes satisfying both the correctness and security constraints are tabulated in Table VII. In Table VII, the columns under "Binary" and "NAF" list our estimates by Eq. 13 for modulus bit size and ring dimension when conventional bit decomposition and NAF decomposition are used, respectively. That the NAF decomposition method allows using much smaller modulus and ring dimension not only improves the execution timings but also the memory requirements. The rightmost two columns under "Experimental values" list the actual values used in our implementation. We tested our implementation with these values and found out that maximum error norm in the decrypted message is at least 8 bit smaller than the selected modulus, which is more than sufficient for correct decryption.

TABLE VII: ABE parameters for various number of attributes with $\sigma = 4.57825$, $\delta = 1.0059$, $\epsilon = 128$, $\omega_e = \omega_\psi = 2^{-8}$.

$\ell = 2^d$	d	Binary		NAF		Experimental	
		k	n	k	n	k	n
2	1	36	1024	36	1024	36	1024
4	2	48	2048	45	2048	51	2048
8	3	65	2048	53	2048	60	2048
16	4	88	4096	61	2048	69	2048
32	5	108	4096	74	4096	82	4096
64	6	127	4096	83	4096	92	4096
128	7	155	8192	95	4096	102	4096
256	8	176	8192	107	4096	111	4096
512	9	197	8192	119	4096	122	4096
1024	10	218	8192	131	4096	132	4096

Figure 3 illustrates the sensitivity of noise growth to different mean values of the discrete Gaussian generator. In conclusion, a high quality Gaussian RNG proves to be still important for the overall performance of the ABE scheme.

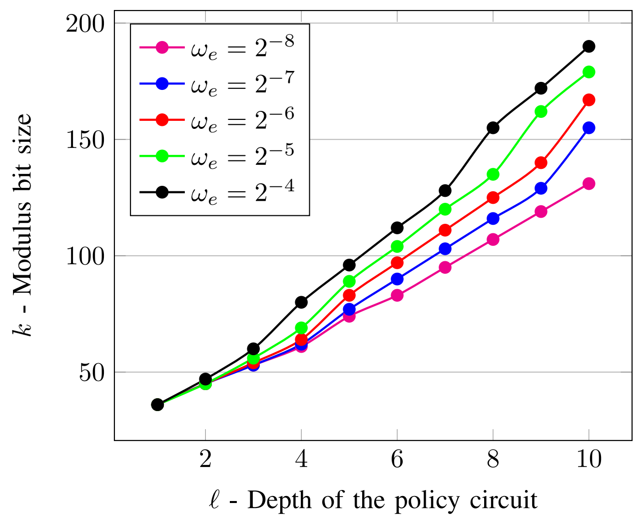

Fig. 3: Sensitivity of noise growth to non-zero mean value of Gaussian RNG.

A. Example of Modulo Reduction in $\mathbb{F}_P$ for NTT Convertions

Table VIII shows reduction operation with special form modulo $P = 2^{64} - 2^{32} + 1$ to accelerate NTT conversion operations as discussed in Section VII.

B. Security of the KP-ABE Scheme

The proposed scheme mainly relies on RLWE hardness assumptions as informally explained in Definitions 3.1 and 3.2, namely search and decision RLWE assumptions [24]. For the security of the trapdoor construction the reader is referred to [23] for the ring version of the trapdoor and to [32] for the specific instantiation of the trapdoor construction used in this paper. The reader is also referred to the seminal works [24], [25], [30] for deeper understanding of the ideal lattices and lattice trapdoors.

TABLE VIII: An example of modulo P reduction.

Powers of 2				224-bit x		$x = x' \ll 147$		Pseudo Code
2^1	$\equiv$		$+1$		$+x_0$			
2^{32}	$\equiv$	$+2^{32}$		$+x_1$				`add.cc x3, x3, x2;`
2^{64}	$\equiv$	$+2^{32}$	-1	$+x_2$	$-x_2$	$+x_2$	$-x_2$	`addc x4, x4, 0;`
2^{96}	$\equiv$		-1		$-x_3$		$-x_3$	`sub.cc r0, 0, x3;`
2^{128}	$\equiv$	-2^{32}		$-x_4$		$-x_4$		`subc r1, x2, x4;`
2^{160}	$\equiv$	-2^{32}	$+1$	$-x_5$	$+x_5$			`r -= (uint32_t)(-((r>>32)>x2));`
2^{192}	$\equiv$		$+1$		$+x_6$			`r += (uint32_t)(-(r>=P));`

We now demonstrate that the security proofs in [19] (i.e., selective security as defined in [3]) remain valid for our RLWE-based construction of the KP-ABE scheme. The security proofs are provided in a series of games played by a LWE solver $\mathcal{B}$ and an adversary $\mathcal{A}$, who has access to a key generation oracle. After $\mathcal{A}$ commits to a particular set of attribute values $x^* = (x_1^*, \ldots, x_\ell^*)$ (henceforth *the challenge attribute*) it can send queries for secret keys to the key generation oracle which can respond only for functions $f(x^*) = 1$.

The basic idea is that if $\mathcal{A}$ has a significant advantage in distinguishing between the ciphertexts of two different messages encrypted under the challenge attribute x^*, then we can show that $\mathcal{B}$ can break the decision RLWE hardness assumption in Definition 3.2. In the next section, we show how the oracle responds to queries for functions $f(x^*) = 1$ using SAMPLELEFT algorithm .

1) SAMPLELEFT Algorithm: This algorithm is fundamental to the ABE construction used in our work. In a nutshell, having $\mathbf{G}$ as a primitive vector one can obtain a small-norm solution, $\mathbf{y} \in \mathcal{R}_q^{2m\times 1}$, for $(\mathbf{A}|\mathbf{AS} - \mathbf{G})\mathbf{y} = u$, where $u \leftarrow_U \mathcal{R}_q$, $\mathbf{A} \leftarrow_U \mathcal{R}_q^m$, and $\mathbf{S} \leftarrow D_{\mathcal{R}^{m\times m},\sigma}$ is a matrix of small norm polynomials (e.g., following a Gaussian distribution). We can formulate the algorithm as follows:

$$\textsc{SampleLeft}(\mathbf{A}, \mathbf{G}, \mathbf{S}, u) \to \mathbf{y}.$$

Algorithm SAMPLELEFT relies on **Construction 2** described in [23] based on the findings in [52], [53]. In what follows, we provide a brief explanation for our version of the construction in [23].

The primitive vector $\mathbf{G} = (g_1, g_2, \ldots, g_k, 0, 0)$, where $g_i = 2^{i-1}$ and as $m = k + 2$, $\mathbf{G} \in \mathcal{R}_q^{1\times m}$. The vector $\mathbf{A} = (a_1, a_2, \ldots, a_m)$, where $a_i \leftarrow_U \leftarrow \mathcal{R}_q$. Also $\mathbf{s}_i \in R^{m\times 1}$ represents i-th column of $\mathbf{S}$. Consequently, $\mathbf{A}\mathbf{s}_j = \sum_{i=1}^m a_i s_{ji}$ is an element of $\mathcal{R}_q$.

Let $\mathbf{F} = (\mathbf{A}|\mathbf{AS} - \mathbf{G}) = (a_1, \ldots, a_m, \mathbf{A}\mathbf{s}_1 - g_1, \ldots, \mathbf{A}\mathbf{s}_k - g_k, \mathbf{A}\mathbf{s}_{k+1}, \mathbf{A}\mathbf{s}_{k+2})$, where the terms $\mathbf{A}\mathbf{s}_j$ for $j = 1, \ldots, m$ are uniformly distributed [52], [53]. It follows that $\mathbf{F}$ is uniformly distributed and the vectors $\mathbf{s}_j \in \mathcal{R}_q^{m\times 1}$ form a trapdoor $\mathbf{T}_\mathbf{F}$ for $\mathbf{F}$.

To generate a preimage of a uniformly randomly selected $u \leftarrow_U \mathcal{R}_q$, we first sample a vector $\mathbf{x} \in \Lambda_u^\perp(\mathbf{G})$ using the trapdoor $\mathbf{T}_\mathbf{G}$, where $\mathbf{x}$ is a vector of low norm polynomials and hence $\sum_{i=1}^m g_i x_i = u$. Then, one can easily verify that $\mathbf{y} = (y_1, \ldots, y_m, y_{m+1}, \ldots y_{2m})$ is a preimage of the syndrome u for $\mathbf{F}$, where $y_i = \sum_{j=1}^m x_j s_{ji}$ for $i = 1, \ldots, m$ and $y_i = -x_i$ for $i = m+1, \ldots, 2m$. Note that y_i are also polynomials with small norms. The following shows that $\mathbf{F}\mathbf{y} = u$.

$$\begin{aligned}
\mathbf{F}\mathbf{y} &= a_1 \sum_{j=1}^m x_j s_{j1} + \ldots + a_m \sum_{j=1}^m x_j s_{jm} + \\
&\ldots + x_1(g_1 - \mathbf{A}\mathbf{s}_1) + \ldots + x_m(-\mathbf{A}\mathbf{s}_m) \\
&= \sum_{i=1}^m a_i \sum_{j=1}^m x_j s_{ji} + \sum_{i=1}^{m-2} g_i x_i - \sum_{j=1}^m x_j \mathbf{A}\mathbf{s}_j \\
&= \sum_{j=1}^m x_j \sum_{i=1}^m a_i s_{ji} + u - \sum_{j=1}^m x_j \mathbf{A}\mathbf{s}_j \\
&= \sum_{j=1}^m x_j \mathbf{A}\mathbf{s}_j + u - \sum_{j=1}^m x_j \mathbf{A}\mathbf{s}_j \\
&= u.
\end{aligned} \tag{16}$$

However, since the distribution of $\mathbf{y}$ is ellipsoidal, not spherical as required in [14], and leaks information about the trapdoor, we need a spherically distributed preimage sample for u, which can be obtained using the techniques in [23].

In the next section, we demonstrate that how the key generation oracle responds to a secret key request for a circuit $f(x^*) = 1$ for the challenge attribute x^*. The oracle only needs to provide a small norm solution to a vector of the form $(\mathbf{A}|f(x^*)\mathbf{G} - \mathbf{A}\mathbf{S}_f)$, where $\mathbf{S}_f$ is a matrix of relatively small norm polynomials, which is possible to obtain using the method described in this section provided that $f(x^*) \neq 0$ ($f(x^*) = 1$ for binary attributes); with the only exception for $f(x^*) = 0$, which defines our access policy and results in a vector of the form $(\mathbf{A}|\mathbf{A}\mathbf{S}_f)$.

2) Simulated Circuit Evaluation: In some of the security games in [19], the public vector $\mathbf{A} \leftarrow_U \mathcal{R}_q^{1\times m}$ is chosen uniformly randomly, instead of using TRAPGEN function, which produces a pseudorandom public vector. Conversely, instead of selecting them uniformly randomly we use $\mathbf{B}_i = \mathbf{A}\mathbf{S}_i - x_i^*\mathbf{G}$ produced pseudorandomly, where x^* is the challenge attribute. And also $\mathbf{S}_i \in \{\pm 1\}^{m\times m}$ is chosen uniformly randomly for $i = 1, \ldots, \ell$. Without loss of generality, we assume that the policy circuit consists of only multiplication and addition/subtraction gates; and thus we do not use $\mathbf{B}_0$ henceforth.

The idea is to evaluate $\mathbf{S}_i$ matrices over the given circuit $f(x^*) \neq 0$, where x^* is committed to by adversary $\mathcal{A}$ before the security games start. Evaluation of the matrices $\mathbf{S}_i$ is indeed very similar to the evaluation of the public vectors $\mathbf{B}_i$ for $i = 1, \ldots, \ell$. The only difference is the fact that $\mathbf{S}_i$ is a matrix consisting of either $+1$ or -1. We can even consider that $\mathbf{S}_i$ is a matrix of constant polynomials in $\mathcal{R}_q^{m\times m}$. Then, we can write evaluation algorithms of addition/subtraction and

AND gates for two such matrices $\mathbf{S}_{i_1}$ and $\mathbf{S}_{i_2}$

$$\begin{aligned}\mathbf{S}_{\pm} &= \mathbf{S}_{i_1} + \mathbf{S}_{i_2}\\ \mathbf{S}_{AND} &= x^*_{i_2}\mathbf{S}_{i_2} + \mathbf{S}_{i_2}\textsc{BitDecomp}(-\mathbf{B}_{i_1}),\end{aligned} \tag{17}$$

respectively. Here, the results are also matrices of the same type, i.e., $\mathbf{S}_{\pm}, \mathbf{S}_{AND} \in \mathcal{R}_q^{m\times m}$, possibly with larger norms, for which one can provide upper bounds. To compute the evaluation of $\mathbf{S}_i$s over the circuit, we first call, $\mathbf{B}_f = \textsc{EvalPK}(f, (\mathbf{AS}_i - x^*_i\mathbf{G})_{i=1}^{\ell}))$, whereby we also store the $\mathbf{B}$ vectors calculated for each gate. After using either one of the formula in Eq. 17 for each gate in the circuit to perform evaluations, we obtain a matrix $\mathbf{S}_f \in \mathcal{R}_q^{m\times m}$ for the output of the circuit. As $\mathbf{S}_f$ is obtained as a result of the evaluation operation, we can write for the norm of $\mathbf{S}_f$, $||\mathbf{S}_f|| < \Delta_f$, where Δ_f measures the increase in the noise magnitude in a ciphertext $\mathbf{C}_f$ compared to the input ciphertexts $\mathbf{C}_i$. As we have $f(x^*) = 1$, we need to generate a low norm solution to $(\mathbf{A}|\mathbf{AS}_f - \mathbf{G})\boldsymbol{\alpha}_f$, which is possible using the technique described in Section IX-B1. Then the simulated circuit evaluation algorithm is described as

$$\textsc{EvalSim}(f, (x^*_i, \mathbf{S}_i)_{i=1}^{\ell}, \mathbf{A}) \rightarrow \mathbf{S}_f.$$

C. Security Games

In this section, we briefly explain **Game 2** and **Game 3** from [19], where the goal is to show that they are indistinguishable for a probabilistic polynomial time (PPT) adversary $\mathcal{A}$. **Game 2** proceeds as follows:

1) Adversary $\mathcal{A}$ commits to a set of attribute values x^*.
2) $\beta \leftarrow_U \mathcal{R}_q$, $\mathbf{A} \leftarrow_U \mathcal{R}_q^{1\times m}$, (i.e., uniformly randomly chosen)
3) $\mathcal{B}$ performs the following:
   a) $\mathbf{S}_i \leftarrow_U \{\pm 1\}^{m\times m}$ for $i = 1, \ldots, \ell$ (i.e., uniformly randomly chosen)
   b) $\mathbf{B}_i = x^*_i\mathbf{G} - \mathbf{AS}_i$ for $i = 1, \ldots, \ell$
   c) Public key $\texttt{MPK} = (\mathbf{A}, \mathbf{B}_i, \ldots, \mathbf{B}_\ell, \beta)$ is sent to $\mathcal{A}$
4) $\mathcal{A}$ cannot distinguish $\mathbf{B}_i$'s in MPK and uniformly randomly chosen $\mathbf{B}_i$'s in normal execution of the ABE algorithm.
5) $\mathcal{A}$ picks a plaintext pair (μ_0, μ_1) and sends it to $\mathcal{B}$.
6) $\mathcal{B}$ encrypts one of them μ_b at random ($b \in (0, 1)$) and sends the challenge ciphertext to $\mathcal{A}$.
7) $\mathcal{A}$ can query the oracle for any function f provided that $f(x^*) = 1$.
8) For any Boolean function $f(x^*) = 1$, the key generation oracle does
   - compute $\mathbf{B}_f = \textsc{EvalPK}((\mathbf{AS}_i, x^*_i\mathbf{G})_{i=1}^{\ell}, f)$
   - compute $\mathbf{S}_f = \textsc{EvalSim}(f, (x^*_i, \mathbf{S}_i)_{i=1}^{\ell}, A)$
   - return $\boldsymbol{\alpha}_f = \textsc{SampleLeft}(\mathbf{A}, \mathbf{G}, \mathbf{S}, \beta)$, where $(\mathbf{A}|\mathbf{AS}_f - \mathbf{G})\boldsymbol{\alpha}_f = \beta$
9) However, it cannot answer any query for $(\mathbf{A}|\mathbf{AS}_f)\boldsymbol{\alpha}_f = \beta$, which corresponds to the case $f(x^*) = 0$.
10) Thus, $\mathcal{A}$ has no significant advantage to tell whether $b = 0$ or $b = 1$.

In Step 6, the ciphertext will be

$$\begin{aligned}\mathbf{C}_{\text{in}} &= (\mathbf{A}|(x^*_1\mathbf{G} + \mathbf{B}_1)|\ldots|(x^*_\ell\mathbf{G} + \mathbf{B}_\ell))^T s + \mathbf{e}_0\\ &= (\mathbf{A}|(x^*_1\mathbf{G}+\mathbf{AS}_1-x^*_1\mathbf{G})|\ldots|(x^*_\ell\mathbf{G}+\mathbf{AS}_\ell-x^*_\ell\mathbf{G}))^T s+\mathbf{e}_0\\ &= (\mathbf{A}|\mathbf{AS}_1|\ldots|\mathbf{AS}_\ell)^T s + \mathbf{e}_0\\ c_1 &= \beta s + e_1 + \mu_b\lceil\frac{q}{2}\rceil\end{aligned}$$

In $\mathbf{C}_{\text{in}}$, $(\mathbf{A}, (\mathbf{AS}_1|\ldots|\mathbf{AS}_\ell), \mathbf{e}_0)$ is statistically close to $(\mathbf{A}, (\mathbf{A'}_1|\ldots|\mathbf{A'}_\ell), \mathbf{e}_0)$ for a uniformly randomly selected $\mathbf{A'}_i$. Therefore, $\mathcal{A}$ views all vectors $\mathbf{AS}_i$ statistically close to uniform.

Game 3 is identical to **Game 2** except that the challenge to $\mathcal{A}$ contains uniformly randomly selected pair and $\mathcal{A}$ cannot distinguish it from the valid ciphertext generated as in **Game 2**. More specifically, $\mathcal{B}$ is given the pair $(\mathbf{C}_A, c_1)$ which are either random, i.e., $\mathbf{C}_A \leftarrow_U R_q^{1\times m}$ and $c_1 \leftarrow_U R_q$ or

$$\begin{aligned}\mathbf{C}_A &= \mathbf{A}^T s + \mathbf{e}_0\\ c_1 &= \beta s + e_1,\end{aligned}$$

where $s \leftarrow_U \mathcal{R}_q$, $\mathbf{e}_0 \leftarrow D_{\mathcal{R}^m,\sigma}$, and $e_1 \leftarrow D_{\mathcal{R},\sigma}$. The game proceeds identically to **Game 2** until Step 6, which is performed by $\mathcal{B}$ in a slightly different manner. $\mathcal{B}$ picks one of the plaintext at random μ_b and performs the following:

$$\begin{aligned}\mathbf{C}^*_{\text{in}} &= (\mathbf{C}_A|\mathbf{S}_1^T\mathbf{C}_A|\ldots|\mathbf{S}_\ell^T\mathbf{C}_A)\\ c &= c_1 + \mu\lceil\frac{q}{2}\rceil\end{aligned}$$

$\mathcal{A}$ cannot distinguish whether it is **Game 2** or **Game 3** since we would have

$$\begin{aligned}\mathbf{C}^*_{\text{in}} &= ((\mathbf{A}^T s+\mathbf{e}_A)|((\mathbf{AS}_1)^T s+\mathbf{S}_1^T\mathbf{e}_A)|\ldots|((\mathbf{AS}_\ell)^T s+\mathbf{S}_\ell^T\mathbf{e}_A)\\ &= (\mathbf{A}|\mathbf{AS}_1|\ldots|(\mathbf{AS}_\ell)^T s + \mathbf{e}_0\\ c^*_1 &= \beta s + e_1 + \mu_b\lceil\frac{q}{2}\rceil,\end{aligned}$$

which were a valid ciphertext generated in **Game 2** if it were being played.

Conversely, suppose adversary $\mathcal{A}$ can guess b with ϵ advantage if it is given a valid ciphertext. This means $\mathcal{A}$ can win **Game 2** with ϵ advantage whereas its guess for b in **Game 3** can only be correct with 1/2 probability (indicating it has zero advantage in **Game 3**). In turn, $\mathcal{B}$ can distinguish between **Game 2** and **Game 3**, which indicates that $\mathcal{B}$ can solve the decision RLWE problem.

As we only demonstrate that the same security arguments are valid for our RLWE construction of KP-ABE, we deliberately refrain from explaining all security games here and refer the interested reader to [19] for a deeper insight.