Riding on Asymmetry: Efficient ABE for Branching Programs

Sergey Gorbunov* and Dhinakaran Vinayagamurthy**

Abstract. In an Attribute-Based Encryption (ABE) scheme the ciphertext encrypting a message μ, is associated with a public attribute vector $\mathbf{x}$ and a secret key sk_P is associated with a predicate P. The decryption returns μ if and only if $P(\mathbf{x}) = 1$. ABE provides efficient and simple mechanism for data sharing supporting fine-grained access control. Moreover, it is used as a critical component in constructions of succinct functional encryption, reusable garbled circuits, token-based obfuscation and more.

In this work, we describe a new efficient ABE scheme for a family of branching programs with short secret keys and from a mild assumption. In particular, in our construction the size of the secret key for a branching program P is $|P| + \mathrm{poly}(\lambda)$, where λ is the security parameter. Our construction is secure assuming the standard Learning With Errors (LWE) problem with approximation factors $n^{\omega(1)}$. Previous constructions relied on $n^{O(\log n)}$ approximation factors of LWE (resulting in less efficient parameters instantiation) or had large secret keys of size $|P| \times \mathrm{poly}(\lambda)$. We rely on techniques developed by Boneh et al. (EUROCRYPT'14) and Brakerski et al. (ITCS'14) in the context of ABE for circuits and fully-homomorphic encryption.

1 Introduction

Attribute-Based Encryption (ABE) was introduced by Sahai and Waters [SW05] in order to realize the vision of fine-grained access control to encrypted data. Using ABE, a user can encrypt a message μ with respect to a public attribute-vector $\mathbf{x}$ to obtain a ciphertext $\mathsf{ct}_{\mathbf{x}}$. Anyone holding a secret key sk_P, associated with an access policy P, can decrypt the message μ if $P(\mathbf{x}) = 1$. Moreover, the security notion guarantees that no collusion of adversaries holding secret keys $\mathsf{sk}_{P_1}, \ldots, \mathsf{sk}_{P_t}$ can learn anything about the message μ if none of the individual keys allow to decrypt it. Until recently, candidate constructions of ABE were limited to restricted classes of access policies that test for equality (IBE), boolean formulas and inner-products: [Coc01,BF03,GPSW06,Wat09,LW10,LOS+10,ABB10,CHKP12,AFV11,Boy13].

In recent breakthroughs Gorbunov, Vaikuntanathan and Wee [GVW13] and Garg, Gentry, Halevi, Sahai and Waters [GGH+13c] constructed ABE schemes for arbitrary boolean predicates. The GVW construction is based on the standard Learning With Errors (LWE) problem with sub-exponential approximation factors, whereas GGHSW relies on hardness of a (currently) stronger assumptions over existing multilinear map candidates [GGH13a,CLT13,GGH15]. But in both these ABE schemes, the size of the secret keys had a multiplicative dependence on the size of the predicate: $|P| \cdot \mathrm{poly}(\lambda, d)$ (where d is the depth of the circuit representation of the predicate). In a subsequent work, Boneh et al. [BGG+14] showed how to construct ABE for arithmetic predicates with short secret keys: $|P| + \mathrm{poly}(\lambda, d)$, also assuming hardness of LWE with sub-exponential approximation factors. However, in [GVW13], the authors also showed an additional construction for a family of branching programs under a milder and quantitatively better assumption: hardness of LWE with polynomial approximation factors. Basing the security on LWE with polynomial approximation factors, as opposed to sub-exponential, results in two main advantages. First, the security of the resulting construction relies on the hardness of a much milder LWE assumption. But moreover, the resulting instantiation has better parameter – small modulo q – leading directly to practical efficiency improvements.

In this work, we focus on constructing an ABE scheme under milder security assumptions and better performance guarantees. We concentrate on ABE for a family of branching programs which is sufficient for most existing applications such as medical and multimedia data sharing [APG+11,LYZ+13,PPM+14].

* Work done while at MIT, supported by Microsoft PhD fellowship. Email: sgorbunov100@gmail.com.
** University of Toronto. Email: dhinakaran5@cs.toronto.edu

First, we summarize the two most efficient results from learning with errors problem translated to the setting of branching programs (via standard Barrington's theorem [Bar86]). Let L be the length of a branching program P and let λ denote the security parameter. Then,

- [GVW13]: There exists an ABE scheme for length L branching programs with *large secret keys* based on the security of *LWE with polynomial approximation factors*. In particular, in the instantiation $|\mathsf{sk}_P| = |L| \times \mathrm{poly}(\lambda)$ and $q = \mathrm{poly}(L, \lambda)$.
- [BGG+14]: There exists an ABE scheme for length L branching programs with *small secret keys* based on the security of *LWE with quasi-polynomial approximation factors*. In particular, $|\mathsf{sk}_P| = |L| + \mathrm{poly}(\lambda, \log L)$, $q = \mathrm{poly}(\lambda)^{\log L}$.

To advance the state of the art for both theoretical and practical reasons, the natural question that arises is whether we can obtain the best of both worlds and:

Construct an ABE for branching programs with small secret keys based on the security of LWE with polynomial approximation factors?

1.1 Our Results

We present a new efficient construction of ABE for branching programs from a mild LWE assumption. Our result can be summarized in the following theorem.

Theorem 1 (informal). *There exists a selectively-secure Attribute-Based Encryption for a family of length-L branching programs with* small secret keys *based on the security of* LWE with polynomial approximation factors*. More formally, the size of the secret key sk_P is $L + \mathrm{poly}(\lambda, \log L)$ and modulo $q = \mathrm{poly}(L, \lambda)$, where λ is the security parameter.*

Furthermore, we can extend our construction to support arbitrary length branching programs by setting q to some small super-polynomial.

As an additional contribution, our techniques lead to a new efficient constructing of homomorphic signatures for branching programs. In particular, Gorbunov et al. [GVW15b] showed how to construct homomorphic signatures for circuits based on the simulation techniques of Boneh et al. [BGG+14] in the context of ABE. Their resulting construction is secure based on the short integer solution (SIS) problem with *sub-exponential approximation factors* (or quasi-polynomial in the setting of branching programs). Analogously, our simulation algorithm presented in Section 3.4 can be used directly to construct homomorphic signatures for branching programs based on SIS with *polynomial approximation factors*.

Theorem 2 (informal). *There exists a homomorphic signatures scheme for the family of length-L branching programs based on the security of* SIS with polynomial approximation factors.

High Level Overview. The starting point of our ABE construction is the ABE scheme for circuits with short secret keys by Boneh et al. [BGG+14]. At the heart of their construction is a fully key-homomorphic encoding scheme.

It encodes $a \in \{0, 1\}$ with respect to a public key $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$ in a "noisy" sample:

$$\psi_{\mathbf{A},a} = (\mathbf{A} + a \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{e}$$

where $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$ and $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ are fixed across all the encodings and $\mathbf{e} \xleftarrow{\$} \chi^m$ (for some noise distribution χ) is chosen independently every time. The authors show that one can turn such a key-homomorphic encoding scheme, where homomorphism is satisfied over the encoded values and over the public keys simultaneously, into an attribute based encryption scheme for circuits.

Our first key observation is the asymmetric noise growth in their homomorphic multiplication over the encodings. Consider ψ_1, ψ_2 to be the encodings of a_1, a_2 under public keys $\mathbf{A}_1, \mathbf{A}_2$. To achieve multiplicative homomorphism, their first step is to achieve homomorphism over a_1 and a_2 by computing

$$a_1 \cdot \psi_2 = (a_1 \cdot \mathbf{A}_2 + (a_1 a_2) \cdot \mathbf{G})^\intercal \mathbf{s} + a_1 \mathbf{e}_2 \tag{1}$$

Now, since homomorphism over the public key matrices must also be satisfied in the resulting encoding *independently* of a_1, a_2 we must replace $a_1 \cdot \mathbf{A}_2$ in Equation 1 with operations over $\mathbf{A}_1, \mathbf{A}_2$ only. To do this, we can use the first encoding $\psi_1 = (\mathbf{A}_1 + a_1 \cdot \mathbf{G})^\intercal + \mathbf{e}_1$ and replace $a_1 \cdot \mathbf{G}$ with $a_1 \cdot \mathbf{A}_2$ as follows. First, compute $\widetilde{\mathbf{A}}_2 \in \{0,1\}^{m \times m}$ such that $\mathbf{G} \cdot \widetilde{\mathbf{A}}_2 = \mathbf{A}_2$. (Finding such $\widetilde{\mathbf{A}}_2$ is possible since the "trapdoor" of $\mathbf{G}$ is known publicly). Then compute

$$\begin{aligned}(\widetilde{\mathbf{A}}_2)^\intercal \cdot \psi_1 &= \widetilde{\mathbf{A}}_2^\intercal \cdot ((\mathbf{A}_1 + a_1 \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{e}_1) \\ &= \left(\mathbf{A}_1 \widetilde{\mathbf{A}}_2 + a_1 \cdot \mathbf{G}\widetilde{\mathbf{A}}_2\right)^\intercal \mathbf{s} + \widetilde{\mathbf{A}}_2 \mathbf{e}_1 \\ &= \left(\mathbf{A}_1 \widetilde{\mathbf{A}}_2 + a_1 \cdot \mathbf{A}_2\right)^\intercal \mathbf{s} + \mathbf{e}_1' \end{aligned} \tag{2}$$

Subtracting Equation 2 from Equation 1, we get $\left(-\mathbf{A}_1 \widetilde{\mathbf{A}}_2 + (a_1 a_2) \cdot \mathbf{G}\right)^\intercal \mathbf{s} + \mathbf{e}'$ which is an encoding of $a_1 a_2$ under the public key $\mathbf{A}^\times := -\mathbf{A}_1 \widetilde{\mathbf{A}}_2$. Thus,

$$\psi_{\mathbf{A}^\times, a^\times} := a_1 \cdot \psi_2 - \widetilde{\mathbf{A}}_2^\intercal \cdot \psi_1$$

where $a^\times := a_1 a_2$. Here, $\mathbf{e}'$ remains small enough because $\widetilde{\mathbf{A}}_2$ has small (binary) entries. We observe that the new noise $\mathbf{e}' = a_1 \mathbf{e}_2 - \widetilde{\mathbf{A}}_2 \mathbf{e}_1$ grows asymmetrically. That is, the $\mathrm{poly}(n)$ multiplicative increase always occurs with respect to the first noise $\mathbf{e}_1$. Naïvely evaluating k levels of multiplicative homomorphism results in a noise of magnitude $\mathrm{poly}(n)^k$. Can we manage the noise growth by some careful design of the order of homomorphic operations?

To achieve this, comes our second idea: design evaluation algorithms for a "sequential" representation of a matrix branching program to carefully manage the noise growth following the Brakerski-Vaikuntanathan paradigm in the context of fully-homomorphic encryption [BV14].

First, to generate a ciphertext with respect to an attribute vector $\mathbf{x} = (x_1, \ldots, x_\ell)$ we publish encodings of its individual bits:

$$\psi_i \approx (\mathbf{A}_i + x_i \cdot \mathbf{G})^\intercal \mathbf{s}$$

We also publish encoding of an initial start state 0:[1]

$$\psi_0^v \approx (\mathbf{A}_0^v + v_0 \cdot \mathbf{G})^\intercal \mathbf{s}$$

The message μ is encrypted under encoding $\mathbf{u}^\intercal \mathbf{s} + e$ (where $\mathbf{u}$ is treated as the public key) and during decryption the user should obtain a value $\approx \mathbf{u}^\intercal \mathbf{s}$ from $\{\psi_i\}_{i \in [\ell]}, \psi_0^v$ iff $P(\mathbf{x}) = 1$.

Now, suppose the user wants to evaluate a branching program P on the attribute vector $\mathbf{x}$. Informally, the evaluation of a branching program proceeds in steps updating a special state vector. The next state is determined by the current state and one of the input bits (pertaining to this step). Viewing the sequential representation of the branching program allows us to update the state using only a *single* multiplication and a few additions. Suppose v_t represents the state of the program P at step t and the user holds its corresponding encoding ψ_t^v (under some public key). To obtain ψ_{t+1}^v the user needs to use ψ_i (for some i determined by the program). Leveraging on the asymmetry, the state can be updated by multiplying ψ_i with the matrix $\widetilde{\mathbf{A}}_t^v$ corresponding to the encoding ψ_t^v (and then following a few simple addition steps). Since ψ_i always contains a "fresh" noise (which is never increased as we progress evaluating the program), the noise in ψ_{t+1}^v increases from the noise in ψ_t^v only by a *constant additive* factor! As a result, after k steps in the evaluation procedure the noise will be bounded by $k \cdot \mathrm{poly}(n)$. Eventually, if $P(\mathbf{x}) = 1$, the user will learn $\approx \mathbf{u}^\intercal \mathbf{s}$ and be able to recover μ (we refer the reader to the main construction for details).

The main challenge in "riding on asymmetry" for attribute-based encryption is the requirement for satisfying parallel homomorphic properties: we must design separate homomorphic algorithms for operating over the public key matrices and over the encodings that allow for correct decryption. First, we define and

[1] Technically, we need to publish encodings of 5 states, but we simplify the notation in the introduction for conceptual clarify.

design an algorithm for public key homomorphic operations that works specially for branching programs. Second, we design a homomorphic algorithm that works over the encodings that *preserves the homomorphism over public key matrices and the bits*[2] and *carefully manages the noise growth* as illustrated above. To prove the security, we need to argue that no collusion of users is able to learn anything about the message given many secret keys for programs that do not allow for decryption individually. We design a separate public-key simulation algorithm to accomplish this.

1.2 Applications

We summarize some of the known applications of attribute-based encryption. Parno, Raykova and Vaikuntanathan [PRV12] showed how to use ABE to design (publicly) verifiable two-message delegation delegation scheme with a pre-processing phase. Goldwasser, Kalai, Popa, Vaikuntanathan and Zeldovich [GKP+13] showed how to use ABE as a critical building block to construct succinct one-query functional encryption, reusable garbled circuits, token-based obfuscation and homomorphic encryption for Turing machines. Our efficiency improvements for branching programs can be carried into all these applications.

1.3 Other Related Work

A number of works optimized attribute-based encryption for boolean formulas: Attrapadung et al. [ALdP11] and Emura et al. [EMN+09] designed ABE schemes with constant size ciphertext from bilinear assumptions. For arbitrary circuits, Boneh et al. [BGG+14] also showed an ABE with constant size ciphertext from multilinear assumptions. ABE can also be viewed as a special case of functional encryptions [BSW12]. Gorbunov et al. [GVW12] showed functional encryption for arbitrary functions in a bounded collusion model from standard public-key encryption scheme. Garg et al. [GGH+13b] presented a functional encryption for unbounded collusions for arbitrary functions under a weaker security model from multilinear assumptions. More recently, Gorbunov et al. exploited a the asymmetry in the noise growth in [BGG+14] in a different context of design of a predicate encryption scheme based on standard LWE [GVW15a].

1.4 Organization

In Section 2 we present the lattice preliminaries, definitions for ABE and branching programs. In Section 3 we present our main evaluation algorithms and build our ABE scheme in Section 4. We present a concrete instantiation of the parameters in Section 5. Finally, we outline the extensions in Section 6.

2 Preliminaries

Notation. Let PPT denote probabilistic polynomial-time. For any integer $q \geq 2$, we let $\mathbb{Z}_q$ denote the ring of integers modulo q and we represent $\mathbb{Z}_q$ as integers in $(-q/2, q/2]$. We let $\mathbb{Z}_q^{n \times m}$ denote the set of $n \times m$ matrices with entries in $\mathbb{Z}_q$. We use bold capital letters (e.g. $\mathbf{A}$) to denote matrices, bold lowercase letters (e.g. $\mathbf{x}$) to denote vectors. The notation $\mathbf{A}^\intercal$ denotes the transpose of the matrix $\mathbf{A}$. If $\mathbf{A}_1$ is an $n \times m$ matrix and $\mathbf{A}_2$ is an $n \times m'$ matrix, then $[\mathbf{A}_1 \| \mathbf{A}_2]$ denotes the $n \times (m + m')$ matrix formed by concatenating $\mathbf{A}_1$ and $\mathbf{A}_2$. A similar notation applies to vectors. When doing matrix-vector multiplication we always view vectors as column vectors. Also, $[n]$ denotes the set of numbers $1, \ldots, n$.

2.1 Lattice Preliminaries

Learning With Errors (LWE) Assumption The LWE problem was introduced by Regev [Reg09], who showed that solving it *on the average* is as hard as (quantumly) solving several standard lattice problems *in the worst case.*

[2] These bits represent the bits of the attribute vector in the ABE scheme

Definition 1 (LWE). *For an integer $q = q(n) \geq 2$ and an error distribution $\chi = \chi(n)$ over $\mathbb{Z}_q$, the learning with errors problem* $\mathsf{dLWE}_{n,m,q,\chi}$ *is to distinguish between the following pairs of distributions:*

$$\{\mathbf{A}, \mathbf{A}^\intercal \mathbf{s} + \mathbf{x}\} \quad \textit{and} \quad \{\mathbf{A}, \mathbf{u}\}$$

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

Connection to lattices. Let $B = B(n) \in \mathbb{N}$. A family of distributions $\chi = \{\chi_n\}_{n \in \mathbb{N}}$ is called B-bounded if

$$\Pr[\chi \in \{-B, \ldots, B-1, B\}] = 1.$$

There are known quantum [Reg09] and classical [Pei09] reductions between $\mathsf{dLWE}_{n,m,q,\chi}$ and approximating short vector problems in lattices in the worst case, where χ is a B-bounded (truncated) discretized Gaussian for some appropriate B. The state-of-the-art algorithms for these lattice problems run in time nearly exponential in the dimension n [AKS01,MV10]; more generally, we can get a 2^k-approximation in time $2^{\tilde{O}(n/k)}$. Throughout this paper, the parameter $m = \text{poly}(n)$, in which case we will shorten the notation slightly to $\mathsf{LWE}_{n,q,\chi}$.

Trapdoors for Lattices and LWE

Gaussian distributions. Let $D_{\mathbb{Z}^m,\sigma}$ be the truncated discrete Gaussian distribution over $\mathbb{Z}^m$ with parameter σ, that is, we replace the output by $\mathbf{0}$ whenever the $||\cdot||_\infty$ norm exceeds $\sqrt{m} \cdot \sigma$. Note that $D_{\mathbb{Z}^m,\sigma}$ is $\sqrt{m} \cdot \sigma$-bounded.

Lemma 1 (Lattice Trapdoors [Ajt99,GPV08,MP12]). *There is an efficient randomized algorithm* $\mathsf{TrapSamp}(1^n, 1^m, q)$ *that, given any integers $n \geq 1$, $q \geq 2$, and sufficiently large $m = \Omega(n \log q)$, outputs a parity check matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a 'trapdoor' matrix $\mathbf{T_A} \in \mathbb{Z}^{m \times m}$ such that the distribution of $\mathbf{A}$ is* $\text{negl}(n)$*-close to uniform.*

Moreover, there is an efficient algorithm $\mathsf{SampleD}$ *that with overwhelming probability over all random choices, does the following: For any $\mathbf{u} \in \mathbb{Z}_q^n$, and large enough $s = \Omega(\sqrt{n \log q})$, the randomized algorithm* $\mathsf{SampleD}(\mathbf{A}, \mathbf{T_A}, \mathbf{u}, s)$ *outputs a vector $\mathbf{r} \in \mathbb{Z}^m$ with norm $||\mathbf{r}||_\infty \leq ||\mathbf{r}||_2 \leq s\sqrt{n}$ (with probability 1). Furthermore, the following distributions of the tuple $(\mathbf{A}, \mathbf{T_A}, \mathbf{U}, \mathbf{R})$ are within* $\text{negl}(n)$ *statistical distance of each other for any polynomial $k \in \mathbb{N}$:*

- $(\mathbf{A}, \mathbf{T_A}) \leftarrow \mathsf{TrapSamp}(1^n, 1^m, q)$; $\mathbf{U} \leftarrow \mathbb{Z}_q^{n \times k}$; $\mathbf{R} \leftarrow \mathsf{SampleD}(\mathbf{A}, \mathbf{T_A}, \mathbf{U}, s)$.
- $(\mathbf{A}, \mathbf{T_A}) \leftarrow \mathsf{TrapSamp}(1^n, 1^m, q)$; $\mathbf{R} \leftarrow (D_{\mathbb{Z}^m,s})^k$; $\mathbf{U} := \mathbf{AR} \pmod q$.

Sampling algorithms We will use the following algorithms to sample short vectors from specific lattices. Looking ahead, the algorithm $\mathsf{SampleLeft}$ [ABB10,CHKP12] will be used to sample keys in the real system, while the algorithm $\mathsf{SampleRight}$ [ABB10] will be used to sample keys in the simulation.

Algorithm $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{T_A}, \mathbf{u}, \alpha)$*:*

> *Inputs:* a full rank matrix $\mathbf{A}$ in $\mathbb{Z}_q^{n \times m}$, a "short" basis $\mathbf{T}_A$ of $\Lambda_q^\perp(\mathbf{A})$, a matrix $\mathbf{B}$ in $\mathbb{Z}_q^{n \times m_1}$, a vector $\mathbf{u} \in \mathbb{Z}_q^n$, and a Gaussian parameter α.
>
> *Output:* Let $\mathbf{F} := (\mathbf{A} \parallel \mathbf{B})$. The algorithm outputs a vector $\mathbf{e} \in \mathbb{Z}^{m+m_1}$ in the coset $\Lambda_{\mathbf{F}+\mathbf{u}}$.

Theorem 3 ([ABB10, Theorem 17], [CHKP12, Lemma 3.2]). *Let $q > 2$, $m > n$ and $\alpha > \|\mathbf{T_A}\|_{\mathsf{GS}} \cdot \omega(\sqrt{\log(m + m_1)})$. Then* $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{T_A}, \mathbf{u}, \alpha)$ *taking inputs as in (3) outputs a vector $\mathbf{e} \in \mathbb{Z}^{m+m_1}$ distributed statistically close to $D_{\Lambda_{\mathbf{F}+\mathbf{u}},\alpha}$, where $\mathbf{F} := (\mathbf{A} \parallel \mathbf{B})$.*

where $\|\mathbf{T}\|_{\mathsf{GS}}$ refers to the norm of Gram-Schmidt orthogonalisation of $\mathbf{T}$. We refer the readers to [ABB10] for more details.

Algorithm $\mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{R}, \mathbf{T_G}, \mathbf{u}, \alpha)$*:*

> *Inputs:* matrices $\mathbf{A}$ in $\mathbb{Z}_q^{n\times k}$ and $\mathbf{R}$ in $\mathbb{Z}^{k\times m}$, a full rank matrix $\mathbf{G}$ in $\mathbb{Z}_q^{n\times m}$, a "short" basis $\mathbf{T_G}$ of $\Lambda_q^{\perp}(\mathbf{G})$, a vector $\mathbf{u} \in \mathbb{Z}_q^n$, and a Gaussian parameter α.
> *Output:* Let $\mathbf{F} := (\mathbf{A} \parallel \mathbf{AR} + \mathbf{G})$. The algorithm outputs a vector $\mathbf{e} \in \mathbb{Z}^{m+k}$ in the coset $\Lambda_{\mathbf{F}+\mathbf{u}}$.

Often the matrix $\mathbf{R}$ given to the algorithm as input will be a random matrix in $\{1,-1\}^{m\times m}$. Let S^m be the m-sphere $\{\mathbf{x} \in \mathbb{R}^{m+1} : \|\mathbf{x}\| = 1\}$. We define $s_R := \|\mathbf{R}\| := \sup_{\mathbf{x}\in S^{m-1}} \|\mathbf{R}\cdot\mathbf{x}\|$.

Theorem 4 ([ABB10, Theorem 19]). *Let $q > 2, m > n$ and $\alpha > \|\mathbf{T_G}\|_{\mathsf{GS}} \cdot s_R \cdot \omega(\sqrt{\log m})$. Then $\mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{R}, \mathbf{T_G}, \mathbf{u}, \alpha)$ taking inputs as in (3) outputs a vector $\mathbf{e} \in \mathbb{Z}^{m+k}$ distributed statistically close to $D_{\Lambda_{\mathbf{F}+\mathbf{u}},\alpha}$, where $\mathbf{F} := (\mathbf{A} \parallel \mathbf{AR} + \mathbf{G})$.*

Primitive matrix We use the primitive matrix $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$ defined in [MP12]. This matrix has a trapdoor $\mathbf{T_G}$ such that $\|\mathbf{T_G}\|_\infty = 2$.

We also define an algorithm $\mathsf{invG} : \mathbb{Z}_q^{n\times m} \to \mathbb{Z}_q^{m\times m}$ which deterministically derives a pre-image $\widetilde{\mathbf{A}}$ satisfying $\mathbf{G}\cdot\widetilde{\mathbf{A}} = \mathbf{A}$. From [MP12], there exists a way to get $\widetilde{\mathbf{A}}$ such that $\widetilde{\mathbf{A}} \in \{0,1\}^{m\times m}$.

2.2 Attribute-Based Encryption

An attribute-based encryption scheme $\mathcal{ABE}$ [GPSW06] for a class of circuits $\mathcal{C}$ with ℓ bit inputs and message space $\mathcal{M}$ consists of a tuple of p.p.t. algorithms $(\mathsf{Params}, \mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$:

$\mathsf{Params}(1^\lambda) \to \mathsf{pp}$: The parameter generation algorithm takes the security parameter 1^λ and outputs a public parameter pp which is implicitly given to all the other algorithms of the scheme.

$\mathsf{Setup}(1^\ell) \to (\mathsf{mpk}, \mathsf{msk})$: The setup algorithm gets as input the length ℓ of the input index, and outputs the master public key mpk, and the master key msk.

$\mathsf{Enc}(\mathsf{mpk}, x, \mu) \to \mathsf{ct_x}$: The encryption algorithm gets as input mpk, an index $x \in \{0,1\}^\ell$ and a message $\mu \in \mathcal{M}$. It outputs a ciphertext $\mathsf{ct_x}$.

$\mathsf{KeyGen}(\mathsf{msk}, C) \to \mathsf{sk}_C$: The key generation algorithm gets as input msk and a predicate specified by $C \in \mathcal{C}$. It outputs a secret key sk_C.

$\mathsf{Dec}(\mathsf{ct_x}, \mathsf{sk}_C) \to \mu$: The decryption algorithm gets as input $\mathsf{ct_x}$ and sk_C, and outputs either $\perp$ or a message $\mu \in \mathcal{M}$.

Definition 2 (Correctness). *We require that for all $(\mathbf{x}, C)$ such that $C(\mathbf{x}) = 1$ and for all $\mu \in \mathcal{M}$, we have $\Pr[\mathsf{ct_x} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, \mu); \mathsf{Dec}(\mathsf{ct_x}, \mathsf{sk}_C) = \mu)] = 1$ where the probability is taken over $\mathsf{pp} \leftarrow \mathsf{Params}(1^\lambda), (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\ell)$ and the coins of all the algorithms in the expression above.*

Definition 3 (Security).
For a stateful adversary $\mathcal{A}$, we define the advantage function $\mathsf{Adv}^{\mathrm{ABE}}_{\mathcal{A}}(\lambda)$ to be

$$\Pr\left[b = b' : \begin{array}{l} \mathbf{x}, d_{\max} \leftarrow \mathcal{A}(1^\lambda, 1^\ell); \\ \mathsf{pp} \leftarrow \mathsf{Params}(1^\lambda, 1^{d_{\max}}); \\ (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, \mathbf{x}^*); \\ (\mu_0, \mu_1) \leftarrow \mathcal{A}^{\mathsf{Keygen}(\mathsf{msk},\cdot)}(\mathsf{mpk}), \\ \quad |\mu_0| = |\mu_1|; \\ b \xleftarrow{\$} \{0,1\}; \\ \mathsf{ct_x} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, \mu_b); \\ b' \leftarrow \mathcal{A}^{\mathsf{Keygen}(\mathsf{msk},\cdot)}(\mathsf{ct_x}) \end{array}\right] - \frac{1}{2}$$

with the restriction that all queries y that $\mathcal{A}$ makes to $\mathsf{Keygen}(\mathsf{msk}, \cdot)$ satisfies $C(\mathbf{x}^*) = 0$ (that is, sk_C does not decrypt $\mathsf{ct_x}$). An attribute-based encryption scheme is selectively secure *if for all PPT adversaries $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathrm{ABE}}_{\mathcal{A}}(\lambda)$ is a negligible function in λ.*

2.3 Branching Programs

We define branching programs similar to [BV14]. A width-w branching program BP of length L with input space $\{0,1\}^\ell$ and s states (represented by $[s]$) is a sequence of L tuples of the form $(\mathsf{var}(t), \sigma_{t,0}, \sigma_{t,1})$ where

- $\sigma_{t,0}$ and $\sigma_{t,1}$ are injective functions from $[s]$ to itself.
- $\mathsf{var} : [L] \to [\ell]$ is a function that associates the t-th tuple $\sigma_{t,0}, \sigma_{t,1}$ with the input bit $x_{\mathsf{var}(t)}$.

The branching program BP on input $\mathbf{x} = (x_1, \ldots, x_\ell)$ computes its output as follows. At step t, we denote the state of the computation by $\eta_t \in [s]$. The initial state is $\eta_0 = 1$. In general, η_t can be computed recursively as

$$\eta_t = \sigma_{t,x_{\mathsf{var}(t)}}(\eta_{t-1})$$

Finally, after L steps, the output of the computation $\mathsf{BP}(\mathbf{x}) = 1$ if $\eta_L = 1$ and 0 otherwise.

As done in [BV14], we represent states with bits rather than numbers to bound the noise growth. In particular, we represent the state $\eta_t \in [s]$ by a unit vector $\mathbf{v}_t \in \{0,1\}^s$. The idea is that $\mathbf{v}_t[i] = 1$ if and only if $\sigma_{t,x_{\mathsf{var}(t)}}(\eta_{t-1}) = i$. Note that we can also write the above expression as $\mathbf{v}_t[i] = 1$ if and only if either:

- $\mathbf{v}_{t-1}\left[\sigma_{t,0}^{-1}(i)\right] = 1$ and $x_{\mathsf{var}(t)} = 0$
- $\mathbf{v}_{t-1}\left[\sigma_{t,1}^{-1}(i)\right] = 1$ and $x_{\mathsf{var}(t)} = 1$

This latter form will be useful for us since it can be captured by the following formula. For $t \in [L]$ and $i \in [s]$,

$$\begin{aligned}\mathbf{v}_t[i] &:= \mathbf{v}_{t-1}\left[\sigma_{t,0}^{-1}(i)\right] \cdot (1 - x_{\mathsf{var}(t)}) + \mathbf{v}_{t-1}\left[\sigma_{t,1}^{-1}(i)\right] \cdot x_{\mathsf{var}(t)} \\ &= \mathbf{v}_{t-1}\left[\gamma_{t,i,0}\right] \cdot (1 - x_{\mathsf{var}(t)}) + \mathbf{v}_{t-1}\left[\gamma_{t,i,1}\right] \cdot x_{\mathsf{var}(t)}\end{aligned}$$

where $\gamma_{t,i,0} := \sigma_{t,0}^{-1}(i)$ and $\gamma_{t,i,1} = \sigma_{t,1}^{-1}(i)$ can be publicly computed from the description of the branching program. Hence, $\left\{\mathsf{var}(t), \{\gamma_{t,i,0}, \gamma_{t,i,1}\}_{i\in[s]}\right\}_{t\in[L]}$ is also valid representation of a branching program BP.

For clarity of presentation, we will deal with width-5 permutation branching programs, which is shown to be equivalent to the circuit class $\mathcal{NC}^1$ [Bar86]. Hence, we have $s = w = 5$ and the functions σ_0, σ_1 are permutations on $[5]$.

3 Our Evaluation Algorithms

In this section we describe the key evaluation and encoding (ciphertext) evaluation algorithms that will be used in our ABE construction. The algorithms are carefully designed to manage the noise growth in the LWE encodings *and* to preserve parallel homomorphism over the public keys and the encoded values.

3.1 Basic Homomorphic Operations

We first describe basic homomorphic addition and multiplication algorithms over the public keys and encodings (ciphertexts) based on the techniques developed by Boneh et al. [BGG+14].

Definition 4 (LWE Encoding). *For any matrix $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, we define an LWE encoding of a bit $a \in \{0,1\}$ with respect to a (public) key $\mathbf{A}$ and randomness $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$ as*

$$\psi_{\mathbf{A},\mathbf{s},a} = (\mathbf{A} + a \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{e} \in \mathbb{Z}_q^m$$

for error vector $\mathbf{e} \xleftarrow{\$} \chi^m$ and an (extended) primitive matrix $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$.

In our construction, however, all encodings will be under the same LWE secret $\mathbf{s}$, hence for simplicity we will simply refer to such an encoding as $\psi_{\mathbf{A},a}$.

Definition 5 (Noise Function). *For every* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}, \mathbf{s} \in \mathbb{Z}_q^n$ *and encoding* $\psi_{\mathbf{A},a} \in \mathbb{Z}_q^m$ *of a bit* $a \in \{0,1\}$ *we define a noise function as*

$$\mathsf{Noise}_{\mathbf{s}}(\psi_{\mathbf{A},a}) := ||\psi_{\mathbf{A},a} - (\mathbf{A} + a \cdot \mathbf{G})^\intercal \mathbf{s} \mod q||_\infty$$

Looking ahead, in Lemma 8 we show that if the noise obtained after applying homomorphic evaluation is $\leq q/4$, then our ABE scheme will decrypt the message correctly. Now we define the basic additive and multiplicative operations on the encodings of this form, as per [BGG+14]. In their context, they refer to a matrix $\mathbf{A}$ as the "public key" and $\psi_{\mathbf{A},a}$ as a ciphertext.

Homomorphic addition This algorithm takes as input two encodings $\psi_{\mathbf{A},a}, \psi_{\mathbf{A}',a'}$ and outputs the sum of them. Let $\mathbf{A}^+ = \mathbf{A} + \mathbf{A}'$ and $a^+ = a + a'$.

$$\mathsf{Add}_{\mathsf{en}}(\psi_{\mathbf{A},a}, \psi_{\mathbf{A}',a'}) : \text{ Output } \psi_{\mathbf{A}^+,a^+} := \psi_{\mathbf{A},a} + \psi_{\mathbf{A}',a'} \mod q$$

Lemma 2 (Noise Growth in $\mathsf{Add}_{\mathsf{en}}$). *For any two valid encodings* $\psi_{\mathbf{A},a}, \psi_{\mathbf{A}',a'} \in \mathbb{Z}_q^m$*, let* $\mathbf{A}^+ = \mathbf{A} + \mathbf{A}'$ *and* $a^+ = a + a'$ *and* $\psi_{\mathbf{A}^+,a^+} = \mathsf{Add}_{\mathsf{en}}(\psi_{\mathbf{A},a}, \psi_{\mathbf{A}',a'})$*, then we have*

$$\mathsf{Noise}_{\mathbf{A}^+,a^+}(\psi_{\mathbf{A}^+,a^+}) \leq \mathsf{Noise}_{\mathbf{A},a}(\psi_{\mathbf{A},a}) + \mathsf{Noise}_{\mathbf{A}',a'}(\psi_{\mathbf{A}',a'})$$

Proof. Given two encodings we have,

$$\begin{aligned}
\psi_{\mathbf{A}^+,a^+} &= \psi_{\mathbf{A},a} + \psi_{\mathbf{A}',a'} \\
&= ((\mathbf{A} + a \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{e}) + ((\mathbf{A}' + a' \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{e}') \\
&= ((\mathbf{A} + \mathbf{A}') + (a + a') \cdot \mathbf{G})^\intercal \mathbf{s} + (\mathbf{e} + \mathbf{e}') \\
&= (\mathbf{A}^+ + a^+ \cdot \mathbf{G})^\intercal \mathbf{s} + (\mathbf{e} + \mathbf{e}')
\end{aligned}$$

Thus, from the definition of the noise function, it follows that

$$\mathsf{Noise}_{\mathbf{A}^+,a^+}(\psi_{\mathbf{A},a} + \psi_{\mathbf{A}',a'}) \leq \mathsf{Noise}_{\mathbf{A},a}(\psi_{\mathbf{A},a}) + \mathsf{Noise}_{\mathbf{A}',a'}(\psi_{\mathbf{A}',a'})$$

Homomorphic multiplication This algorithm takes in two encodings $\psi_{\mathbf{A},a} = (\mathbf{A} + a \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{e}_1$ and $\psi_{\mathbf{A}',a'} = (\mathbf{A}' + a' \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{e}_2$ and outputs an encoding $\psi_{\mathbf{A}^\times,a^\times}$ where $\mathbf{A}^\times = -\mathbf{A}\widetilde{\mathbf{A}'}$ and $a^\times = aa'$ as follows:

$$\mathsf{Multiply}_{\mathsf{en}}(\psi_{\mathbf{A},a}, \psi_{\mathbf{A}',a'}) : \text{ Output } \psi_{\mathbf{A}^\times,a^\times} := -\widetilde{\mathbf{A}'}^\intercal \cdot \psi + a \cdot \psi'.$$

Note that this process requires the knowledge of the attribute a in clear.

Lemma 3 (Noise Growth in $\mathsf{Multiply}_{\mathsf{en}}$). *For any two valid encodings* $\psi_{\mathbf{A},a}, \psi_{\mathbf{A}',a'} \in \mathbb{Z}_q^m$*, let* $\mathbf{A}^\times = -\mathbf{A}\widetilde{\mathbf{A}'}$ *and* $a^\times = aa'$ *and* $\psi_{\mathbf{A}^\times,a^\times} = \mathsf{Multiply}_{\mathsf{en}}(\psi_{\mathbf{A},a}, \psi_{\mathbf{A}',a'})$ *then we have*

$$\mathsf{Noise}_{\mathbf{A}^\times,a^\times}(\psi_{\mathbf{A}^\times,a^\times}) \leq m \cdot \mathsf{Noise}_{\mathbf{A},a}(\psi_{\mathbf{A},a}) + a \cdot \mathsf{Noise}_{\mathbf{A}',a'}(\psi_{\mathbf{A}',a'})$$

Proof. Given two valid encodings, we have

$$\begin{aligned}
\psi_{\mathbf{A}^\times,a^\times} &= -\widetilde{\mathbf{A}'}^\intercal \cdot \psi + a \cdot \psi' \\
&= -\widetilde{\mathbf{A}'}^\intercal \big((\mathbf{A} + a \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{e}\big) + a \cdot \big((\mathbf{A}' + a' \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{e}'\big) \\
&= \left((-\mathbf{A}\widetilde{\mathbf{A}'} - a \cdot \mathbf{A}')^\intercal \mathbf{s} - \widetilde{\mathbf{A}'}^\intercal \mathbf{e}\right) + \left((a \cdot \mathbf{A}' + aa' \cdot \mathbf{G})^\intercal \mathbf{s} + a \cdot \mathbf{e}'\right) \\
&= \big(\underbrace{(-\mathbf{A}\widetilde{\mathbf{A}}_2)}_{\mathbf{A}^\times} + \underbrace{aa'}_{a^\times} \cdot \mathbf{G}\big)^\intercal \mathbf{s} + \big(\underbrace{-\widetilde{\mathbf{A}'}^\intercal \mathbf{e} + a \cdot \mathbf{e}'}_{\mathbf{e}^\times}\big)
\end{aligned}$$

Thus, from the definition of the noise function, we must bound the noise $\mathbf{e}^\times$. Hence,

$$\left\|\mathbf{e}^\times\right\|_\infty \leq \left\|\widetilde{\mathbf{A}'}^\intercal \mathbf{e}\right\|_\infty + a \cdot \|\mathbf{e}'\|_\infty \leq m \cdot \|\mathbf{e}\|_\infty + a \cdot \|\mathbf{e}'\|_\infty$$

where the last inequality holds since $\widetilde{\mathbf{A}'} \in \{0,1\}^{m \times m}$.

Note: This type of homomorphism is different from a standard fully homomorphic encryption (FHE) mainly for the following two reasons.

- To perform multiplicative homomorphism, here we need one of the input values *in clear* but the FHE homomorphic operations are performed without the knowledge of the input values.
- The other big difference is that, here we require the output public key matrices $\mathbf{A}^+, \mathbf{A}^\times$ to be independent of the input values a_1, a_2. More generally, when given an arbitrary circuit with AND and OR gates along with the matrices corresponding to its input wires, one should be able to determine the matrix corresponding to the output wire without the knowledge of the values of the input wires. But, this property is not present in any of the existing FHE schemes.

3.2 Our Public Key Evaluation Algorithm

We define a (public) key evaluation algorithm $\mathsf{Eval}_{\mathsf{pk}}$. The algorithm takes as input a description of the branching program BP, a collection of public keys $\{\mathbf{A}_i\}_{i\in[\ell]}$ (one for each attribute bit $\mathbf{x}_i$), a collection of public keys $\mathbf{V}_{0,i}$ for initial state vector and an auxiliary matrix $\mathbf{A}^{\mathsf{c}}$. The algorithm outputs an "evaluated" public key corresponding to the branching program:

$$\mathsf{Eval}_{\mathsf{pk}}(\mathsf{BP}, \{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^{\mathsf{c}}) \to \mathbf{V}_{\mathsf{BP}}$$

The auxiliary matrix $\mathbf{A}^{\mathsf{c}}$ can be thought of as the public key we use to encode a constant 1. We also define $\mathbf{A}'_i := \mathbf{A}^{\mathsf{c}} - \mathbf{A}_i$, as a public key that will encode $1 - \mathbf{x}_i$. The output $\mathbf{V}_{\mathsf{BP}} \in \mathbb{Z}_q^{n\times m}$ is the homomorphically defined public key $\mathbf{V}_{L,1}$ at position 1 of the state vector at the Lth step of the branching program evaluation.

The algorithm proceeds as follows. Recall the description of the branching program BP represented by tuples $\big(\mathsf{var}(t), \{\gamma_{t,i,0}, \gamma_{t,i,1}\}_{i\in[5]}\big)$ for $t \in [L]$. The initial state vector is always taken to be $\mathbf{v}_0 := [1, 0, 0, 0, 0]$. And for $t \in [L]$,

$$\mathbf{v}_t[i] = \mathbf{v}_{t-1}\left[\gamma_{t,i,0}\right] \cdot (1 - x_{\mathsf{var}(t)}) + \mathbf{v}_{t-1}\left[\gamma_{t,i,1}\right] \cdot x_{\mathsf{var}(t)}$$

Our algorithm calculates $\mathbf{V}_{\mathsf{BP}}$ inductively as follows. Assume at time $t-1 \in [L]$, the state public keys $\{\mathbf{V}_{t-1,i}\}_{i\in[5]}$ are already assigned. We assign state public keys $\{\mathbf{V}_{t,i}\}_{i\in[5]}$ at time t as follows.

1. Let $\gamma_0 := \gamma_{t,i,0}$ and $\gamma_1 := \gamma_{t,i,1}$.
2. Let $\mathbf{V}_{t,i} = -\mathbf{A}'_{\mathsf{var}(t)}\widetilde{\mathbf{V}}_{t-1,\gamma_0} - \mathbf{A}_{\mathsf{var}(t)}\widetilde{\mathbf{V}}_{t-1,\gamma_1}$.

It is important to note that the public key defined at each step of the state vector is *independent of any input attribute vector*. Now, let $\mathbf{V}_{L,1}$ be the public key assigned at position 1 at step L of the branching program. We simply output $\mathbf{V}_{\mathsf{BP}} := \mathbf{V}_{L,1}$.

3.3 Our Encoding Evaluation Algorithm

We also define an encoding evaluation algorithm $\mathsf{Eval}_{\mathsf{en}}$ which we will use in the decryption algorithm of our ABE scheme. The algorithm takes as input the description of a branching program BP, an attribute vector $\mathbf{x}$, a set of encodings for the attribute (with corresponding public keys) $\{\mathbf{A}_i, \psi_i := \psi_{\mathbf{A}_i,x_i}\}_{i\in[\ell]}$, encodings of the initial state vector $\{\mathbf{V}_{0,i}, \psi_{0,i} := \psi_{\mathbf{V}_{0,i},\mathbf{v}_0[i]}\}_{i\in[5]}$ and an encoding of a constant "1" $\psi^{\mathsf{c}} := \psi_{\mathbf{A}^{\mathsf{c}},1}$. (From now on, we will use the simplified notations $\psi_i, \psi_{0,i}, \psi^{\mathsf{c}}$ for the encodings). $\mathsf{Eval}_{\mathsf{en}}$ outputs an encoding of the result $y := \mathsf{BP}(\mathbf{x})$ with respect to a homomorphically derived public key $\mathbf{V}_{\mathsf{BP}} := \mathbf{V}_{L,1}$.

$$\mathsf{Eval}_{\mathsf{en}}\big(\mathsf{BP}, \mathbf{x}, \{\mathbf{A}_i, \psi_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}, \psi_{0,i}\}_{i\in[5]}, \mathbf{A}^{\mathsf{c}}, \psi^{\mathsf{c}}\big) \to \psi_{\mathsf{BP}}$$

Recall that for $t \in [L]$, we have for all $i \in [5]$:

$$\mathbf{v}_t[i] = \mathbf{v}_{t-1}\left[\gamma_{t,i,0}\right] \cdot (1 - x_{\mathsf{var}(t)}) + \mathbf{v}_{t-1}\left[\gamma_{t,i,1}\right] \cdot x_{\mathsf{var}(t)}$$

The evaluation algorithm proceeds inductively to update the encoding of the state vector for each step of the branching program. The key idea to obtain the desired noise growth is that we only multiply the *fresh*

encodings of the attribute bits with the binary decomposition of the public keys. The result is then be added to update the encoding of the state vector. Hence, at each step of the computation the noise in the encodings of the state will only grow by some fixed additive factor.

The algorithm proceeds as follows. We define $\psi'_i := \psi_{\mathbf{A}'_i,(1-x_i)} = (\mathbf{A}'_i + (1-x_i)\cdot\mathbf{G})^\intercal\mathbf{s} + \mathbf{e}'_i$ to denote the encoding of $1-x_i$ with respect to $\mathbf{A}'_i = \mathbf{A}^{\mathsf{c}} - \mathbf{A}_i$. Note that it can be computed using $\mathsf{Add}_{\mathsf{en}}(\psi_{\mathbf{A}^{\mathsf{c}},1}, -\psi_{\mathbf{A}_i,x_i})$. Assume at time $t-1 \in [L]$ we hold encodings of the state vector $\{\psi_{\mathbf{V}_{t-1,i},\mathbf{v}_t[i]}\}_{i\in[5]}$. Now, we compute the encodings of the new state values:

$$\psi_{t,i} = \mathsf{Add}_{\mathsf{en}}\left(\mathsf{Multiply}_{\mathsf{en}}(\psi'_{\mathsf{var}(t)}, \psi_{t-1,\gamma_0}), \mathsf{Multiply}_{\mathsf{en}}(\psi_{\mathsf{var}(t)}, \psi_{t-1,\gamma_1})\right)$$

where $\gamma_0 := \gamma_{t,i,0}$ and $\gamma_1 := \gamma_{t,i,1}$. As we show below (in Lemma 4), this new encoding has the form $\left(\mathbf{V}_{t,i} + \mathbf{v}_t[i]\cdot\mathbf{G}\right)^\intercal\mathbf{s} + \mathbf{e}_{t,i}$ (for a small enough noise term $\mathbf{e}_{t,i}$).

Finally, let $\psi_{L,1}$ be the encoding obtained at the Lth step corresponding to state value at position "1" by this process. As we show in Lemma 5, noise term $\mathbf{e}_{\mathsf{BP}}$ has "low" infinity norm enabling correct decryption (Lemma 8). The algorithm outputs $\psi_{\mathsf{BP}} := \psi_{L,1}$.

Correctness and Analysis

Lemma 4. *For any valid set of encodings $\psi_{\mathsf{var}(t)}, \psi'_{\mathsf{var}(t)}$ for the bits $x_{\mathsf{var}(t)}, (1-x_{\mathsf{var}(t)})$ and $\{\psi_{t-1,i}\}_{i\in[5]}$ for the state vector $\mathbf{v}_{t-1}$ at step $t-1$, the output of the function*

$$\mathsf{Add}_{\mathsf{en}}\left(\mathsf{Multiply}_{\mathsf{en}}(\psi'_{\mathsf{var}(t)}, \psi_{t-1,\gamma_0}), \mathsf{Multiply}_{\mathsf{en}}(\psi_{\mathsf{var}(t)}, \psi_{t-1,\gamma_1})\right) \to \psi_{t,i}$$

where $\psi_{t,i} = \left(\mathbf{V}_{t,i} + \mathbf{v}_t[i]\cdot\mathbf{G}\right)^\intercal\mathbf{s} + \mathbf{e}_{t,i}$, for some noise term $\mathbf{e}_{t,i}$.

Proof. Given valid encodings $\psi_{\mathsf{var}(t)}, \psi'_{\mathsf{var}(t)}$ and $\{\psi_{t-1,i}\}_{i\in[5]}$, we have:

$$\begin{aligned}
\psi_{t,i} &= \mathsf{Add}_{\mathsf{en}}\left(\mathsf{Multiply}_{\mathsf{en}}(\psi'_{\mathsf{var}(t)}, \psi_{t-1,\gamma_0}), \mathsf{Multiply}_{\mathsf{en}}(\psi_{\mathsf{var}(t)}, \psi_{t-1,\gamma_1})\right) \\
&= \mathsf{Add}_{\mathsf{en}}\Big(\left[(-\mathbf{A}'_{\mathsf{var}(t)}\widetilde{\mathbf{V}}_{t-1,\gamma_0} + (\mathbf{v}_t[\gamma_0]\cdot(1-x_{\mathsf{var}(t)}))\cdot\mathbf{G})^\intercal\mathbf{s} + \mathbf{e}_1)\right], \\
&\qquad \left[(-\mathbf{A}_{\mathsf{var}(t)}\widetilde{\mathbf{V}}_{t-1,\gamma_1} + (\mathbf{v}_t[\gamma_1]\cdot x_{\mathsf{var}(t)})\cdot\mathbf{G})^\intercal\mathbf{s} + \mathbf{e}_2)\right]\Big) \\
&= \Big[\underbrace{\left(-\mathbf{A}'_{\mathsf{var}(t)}\widetilde{\mathbf{V}}_{t-1,\gamma_0} - \mathbf{A}_{\mathsf{var}(t)}\widetilde{\mathbf{V}}_{t-1,\gamma_1}\right)}_{\mathbf{V}_{t,i}} + \underbrace{\left(\mathbf{v}_t[\gamma_0]\cdot(1-x_{\mathsf{var}(t)}) + \mathbf{v}_t[\gamma_1]\cdot x_{\mathsf{var}(t)}\right)}_{\mathbf{v}_t[i]}\cdot\mathbf{G}\Big]^\intercal\mathbf{s} + \mathbf{e}_{t,i}
\end{aligned}$$

where the first step follows from the correctness of $\mathsf{Multiply}_{\mathsf{en}}$ algorithm and last step from that of $\mathsf{Add}_{\mathsf{en}}$ with $\mathbf{e}_{t,i} = \mathbf{e}_1 + \mathbf{e}_2$ where $\mathbf{e}_1 = -\left(\widetilde{\mathbf{V}}_{t-1,\gamma_0}\right)^\intercal\mathbf{e}'_{\mathsf{var}(t)} - (1-x_{\mathsf{var}(t)})\cdot\mathbf{e}_{t-1,\gamma_0}$ and $\mathbf{e}_2 = -\left(\widetilde{\mathbf{V}}_{t-1,\gamma_1}\right)^\intercal\mathbf{e}_{\mathsf{var}(t)} - x_{\mathsf{var}(t)}\cdot\mathbf{e}_{t-1,\gamma_1}$.

Lemma 5. *Let $\mathsf{Eval}_{\mathsf{en}}\left(\mathsf{BP}, \mathbf{x}, \{\mathbf{A}_i, \psi_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}, \psi_{0,i}\}_{i\in[5]}, \mathbf{A}^{\mathsf{c}}, \psi^{\mathsf{c}}\right) \to \psi_{\mathsf{BP}}$ such that all the noise terms, $\left\{\mathsf{Noise}_{\mathbf{A}_i,x_i}(\psi_i)\right\}_{i\in[\ell]}, \mathsf{Noise}_{\mathbf{A}^{\mathsf{c}},1}(\psi^{\mathsf{c}}), \left\{\mathsf{Noise}_{\mathbf{V}_{0,i},\mathbf{v}_0[i]}(\psi_{0,i})\right\}_{i\in[5]}$ are bounded by B, then*

$$\mathsf{Noise}_{\mathbf{V}_{\mathsf{BP}},y}(\psi_{\mathsf{BP}}) \le 3m\cdot L\cdot B + B$$

Proof. We will prove this lemma by induction. That is, we will prove that at any step t,

$$\mathsf{Noise}_{\mathbf{V}_{t,i},\mathbf{v}_t[i]}(\psi_{t,i}) \le 3m\cdot t\cdot B + B$$

for $i \in [5]$. For the base case, $t = 0$, we operate on *fresh* encodings for the initial state vector $\mathbf{v}_0$. Hence, we have that, $\mathsf{Noise}_{\mathbf{V}_{0,i},\mathbf{v}_0[i]}(\psi_{0,i}) \le B$, for all $i \in [5]$. Let $\{\psi_{t-1,i}\}_{i\in[5]}$ be the encodings of the state vector $\mathbf{v}_{t-1}$ at step $t-1$ such that

$$\mathsf{Noise}_{\mathbf{V}_{t-1,i},\mathbf{v}_{t-1}[i]}(\psi_{t-1,i}) \le 3m\cdot(t-1)\cdot B + B$$

for $i \in [5]$. We know that $\psi_{t,i} = \mathsf{Add}_{\mathsf{en}}\left(\mathsf{Multiply}_{\mathsf{en}}(\psi'_{\mathsf{var}(t)}, \psi_{t-1,\gamma_0}), \mathsf{Multiply}_{\mathsf{en}}(\psi_{\mathsf{var}(t)}, \psi_{t-1,\gamma_1})\right)$. Hence, from Lemma 2 and Lemma 3, we get:

$$\begin{aligned}
\mathsf{Noise}_{\mathbf{V}_{t,i},\mathbf{v}_t[i]}(\psi_{t,i}) &\leq \left(m \cdot \mathsf{Noise}_{\mathbf{A}'_{\mathsf{var}(t)},(1-x_{\mathsf{var}(t)})}(\psi'_{\mathsf{var}(t)}) + (1 - x_{\mathsf{var}(t)}) \cdot \mathsf{Noise}_{\mathbf{V}_{t-1,\gamma_0},\mathbf{v}_{t-1}[\gamma_0]}\right) \\
&\quad + \left(m \cdot \mathsf{Noise}_{\mathbf{A}_{\mathsf{var}(t)},x_{\mathsf{var}(t)}}(\psi_{\mathsf{var}(t)}) + x_{\mathsf{var}(t)} \cdot \mathsf{Noise}_{\mathbf{V}_{t-1,\gamma_1},\mathbf{v}_{t-1}[\gamma_1]}\right) \\
&= \left(m \cdot 2B + (1 - x_{\mathsf{var}(t)}) \cdot (3m(t-1)B + B)\right) \\
&\quad + \left(m \cdot B + x_{\mathsf{var}(t)} \cdot (3m(t-1)B + B)\right) \\
&= 3m \cdot t \cdot B + B
\end{aligned}$$

where

$$\mathsf{Noise}_{\mathbf{A}'_{\mathsf{var}(t)},(1-x_{\mathsf{var}(t)})}(\psi'_{\mathsf{var}(t)}) \leq \mathsf{Noise}_{\mathbf{A}^{\mathsf{c}},1}(\psi^{\mathsf{c}}) + \mathsf{Noise}_{-\mathbf{A}_{\mathsf{var}(t)},-x_{\mathsf{var}(t)}}(-\psi_{\mathsf{var}(t)}) \leq B + B = 2B$$

by Lemma 2. With ψ_{BP} being an encoding at step L, we have $\mathsf{Noise}_{\mathbf{V}_{\mathsf{BP}},y}(\psi_{\mathsf{BP}}) \leq 3m \cdot L \cdot B + B$. Thus, $\mathsf{Noise}_{\mathbf{V}_{\mathsf{BP}},y}(\psi_{\mathsf{BP}}) = O(m \cdot L \cdot B)$.

3.4 Our Simulated Public Key Evaluation Algorithm

During simulation, we will use a different procedure for assigning public keys to each wire of the input and the state vector. In particular, $\mathbf{A}_i = \mathbf{A} \cdot \mathbf{R}_i - x_i \cdot \mathbf{G}$ for some *shared* public key $\mathbf{A}$ and some low norm matrix $\mathbf{R}_i$. Similarly, the state public keys $\mathbf{V}_{t,i} = \mathbf{A} \cdot \mathbf{R}_{t,i} - \mathbf{v}_t[i] \cdot \mathbf{G}$. The algorithm thus takes as input the description of the branching program BP, the attribute vector $\mathbf{x}$, two collection of low norm matrices $\{\mathbf{R}_i\}, \{\mathbf{R}_{0,i}\}$ corresponding to the input public keys and initial state vector, a low norm matrix $\mathbf{R}^{\mathsf{c}}$ for the public key of constant 1 and a shared matrix $\mathbf{A}$. It outputs a homomorphically derived low norm matrix $\mathbf{R}_{\mathsf{BP}}$.

$$\mathsf{Eval}_{\mathsf{SIM}}(\mathsf{BP}, \mathbf{x}, \{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \mathbf{R}^{\mathsf{c}}, \mathbf{A}) \to \mathbf{R}_{\mathsf{BP}}$$

The algorithm will ensure that the output $\mathbf{R}_{\mathsf{BP}}$ satisfies $\mathbf{A} \cdot \mathbf{R}_{\mathsf{BP}} - \mathsf{BP}(\mathbf{x}) \cdot \mathbf{G} = \mathbf{V}_{\mathsf{BP}}$, where $\mathbf{V}_{\mathsf{BP}}$ is the homomorphically derived public key.

The algorithm proceeds inductively as follows. Assume at time $t-1 \in [L]$, the we hold a collection of low norm matrices $\mathbf{R}_{t-1,i}$ and public keys $\mathbf{V}_{t-1,i} = \mathbf{A} \cdot \mathbf{R}_{t-1,i} - \mathbf{v}_t[i] \cdot \mathbf{G}$ for $i \in [5]$ corresponding to the state vector. Let $\mathbf{R}'_i = \mathbf{R}^{\mathsf{c}} - \mathbf{R}_i$ for all $i \in [\ell]$. We show how to derive the low norm matrices $\mathbf{R}_{t,i}$ for all $i \in [5]$:

1. Let $\gamma_0 := \gamma_{t,i,0}$ and $\gamma_1 := \gamma_{t,i,1}$.
2. Compute

$$\mathbf{R}_{t,i} = \left(-\mathbf{R}'_{\mathsf{var}(t)}\widetilde{\mathbf{V}}_{t-1,\gamma_0} + (1 - x_{\mathsf{var}(t)}) \cdot \mathbf{R}_{t-1,\gamma_0}\right) + \left(-\mathbf{R}_{\mathsf{var}(t)}\widetilde{\mathbf{V}}_{t-1,\gamma_1} + x_{\mathsf{var}(t)} \cdot \mathbf{R}_{t-1,\gamma_1})\right)$$

Finally, let $\mathbf{R}_{L,1}$ be the matrix obtained at the Lth step corresponding to state value "1" by the above algorithm. Output $\mathbf{R}_{\mathsf{BP}} := \mathbf{R}_{L,1}$. Below, we show that the norm of $\mathbf{R}_{\mathsf{BP}}$ remains small and that homomorphically computed public key $\mathbf{V}_{\mathsf{BP}}$ using $\mathsf{Eval}_{\mathsf{pk}}$ satisfies that $\mathbf{V}_{\mathsf{BP}} = \mathbf{A} \cdot \mathbf{R}_{\mathsf{BP}} - \mathsf{BP}(\mathbf{x}) \cdot \mathbf{G}$.

Lemma 6 (Correctness of $\mathsf{Eval}_{\mathsf{SIM}}$). *For any set of valid inputs to $\mathsf{Eval}_{\mathsf{SIM}}$, we have*

$$\mathsf{Eval}_{\mathsf{SIM}}(\mathsf{BP}, \mathbf{x}, \{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \mathbf{R}^{\mathsf{c}}, \mathbf{A}) \to \mathbf{R}_{\mathsf{BP}}$$

where $\mathbf{V}_{\mathsf{BP}} = \mathbf{A}\mathbf{R}_{\mathsf{BP}} - \mathsf{BP}(\mathbf{x}) \cdot \mathbf{G}$.

Proof. We will prove this lemma by induction. That is, we will prove that at any step t,

$$\mathbf{V}_{t,i} = \mathbf{A}\mathbf{R}_{t,i} - \mathbf{v}_t[i] \cdot \mathbf{G}$$

for any $i \in [5]$. For the base case $t = 0$, since the inputs are valid, we have that $\mathbf{V}_{0,i} = \mathbf{AR}_{0,i} - \mathbf{v}_0[i] \cdot \mathbf{G}$, for all $i \in [5]$. Let $\mathbf{V}_{t-1,i} = \mathbf{AR}_{t-1,i} - \mathbf{v}_{t-1}[i] \cdot \mathbf{G}$ for $i \in [5]$. Hence, we get:

$$\begin{aligned}
\mathbf{AR}_{t,i} &= \Big(-\mathbf{AR}'_{\mathsf{var}(t)}\widetilde{\mathbf{V}}_{t-1,\gamma_0} + (1 - x_{\mathsf{var}(t)}) \cdot \mathbf{AR}_{t-1,\gamma_0}\Big) + \Big(-\mathbf{AR}_{\mathsf{var}(t)}\widetilde{\mathbf{V}}_{t-1,\gamma_1} + x_{\mathsf{var}(t)} \cdot \mathbf{AR}_{t-1,\gamma_1})\Big) \\
&= \Big(- \big(\mathbf{A}'_{\mathsf{var}(t)} + (1 - x_{\mathsf{var}(t)}) \cdot \mathbf{G}\big)\widetilde{\mathbf{V}}_{t-1,\gamma_0} + (1 - x_{\mathsf{var}(t)}) \cdot \big(\mathbf{V}_{t-1,\gamma_0} + \mathbf{v}_{t-1}[\gamma_0] \cdot \mathbf{G}\big)\Big) \\
&\quad + \Big(- \big(\mathbf{A}_{\mathsf{var}(t)} + x_{\mathsf{var}(t)} \cdot \mathbf{G}\big)\widetilde{\mathbf{V}}_{t-1,\gamma_1} + x_{\mathsf{var}(t)} \cdot \big(\mathbf{V}_{t-1,\gamma_1} + \mathbf{v}_{t-1}[\gamma_1] \cdot \mathbf{G}\big)\Big) \\
&= \Big(- \mathbf{A}'_{\mathsf{var}(t)}\widetilde{\mathbf{V}}_{t-1,\gamma_0} - (1 - x_{\mathsf{var}(t)}) \cdot \mathbf{V}_{t-1,\gamma_0} + (1 - x_{\mathsf{var}(t)}) \cdot \mathbf{V}_{t-1,\gamma_0} + \big((1 - x_{\mathsf{var}(t)})\mathbf{v}_{t-1}[\gamma_0]\big) \cdot \mathbf{G}\Big) \\
&\quad + \Big(- \mathbf{A}_{\mathsf{var}(t)}\widetilde{\mathbf{V}}_{t-1,\gamma_1} - x_{\mathsf{var}(t)} \cdot \mathbf{V}_{t-1,\gamma_1} + x_{\mathsf{var}(t)} \cdot \mathbf{V}_{t-1,\gamma_1} + \big(x_{\mathsf{var}(t)}\mathbf{v}_{t-1}[\gamma_1]\big) \cdot \mathbf{G}\Big) \\
&= \big(\underbrace{-\mathbf{A}'_{\mathsf{var}(t)}\widetilde{\mathbf{V}}_{t-1,\gamma_0} - \mathbf{A}_{\mathsf{var}(t)}\widetilde{\mathbf{V}}_{t-1,\gamma_1}}_{\mathbf{V}_{t,i}} \big) + \big(\underbrace{(1 - x_{\mathsf{var}(t)})\mathbf{v}_{t-1}[\gamma_0] + \big(x_{\mathsf{var}(t)}\mathbf{v}_{t-1}[\gamma_1]\big)}_{\mathbf{v}_t[i]} \big) \cdot \mathbf{G}
\end{aligned}$$

Hence, we have $\mathbf{V}_{t,i} = \mathbf{AR}_{t,i} - \mathbf{v}_t[i] \cdot \mathbf{G}$. Thus, at the Lth step, we have by induction that

$$\mathbf{V}_{\mathsf{BP}} = \mathbf{V}_{L,1} = \mathbf{AR}_{L,1-\mathbf{v}_t[i]\cdot\mathbf{G}} = \mathbf{AR}_{\mathsf{BP}} - \mathbf{v}_t[i] \cdot \mathbf{G}$$

Lemma 7. *Let* $\mathsf{Eval}_{\mathsf{SIM}}\big(\mathsf{BP}, \mathbf{x}, \{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \mathbf{R}^{\mathsf{c}}, \mathbf{A}\big) \to \mathbf{R}_{\mathsf{BP}}$ *such that all the "*$\mathbf{R}$*" matrices are sampled from* $\{-1, 1\}^{m\times m}$*, then*

$$\|\mathbf{R}_{\mathsf{BP}}\|_\infty \leq 3m \cdot L + 1$$

Proof. This proof is very similar to that of Lemma 5. We will prove this lemma also by induction. That is, we will prove that at any step t,

$$\|\mathbf{R}_{t,i}\|_\infty \leq 3m \cdot t + 1$$

for $i \in [5]$. For the base case, $t = 0$, the input $\mathbf{R}_{0,i}$s are such that, $\|\mathbf{R}_{t,0}\|_\infty = 1$, for all $i \in [5]$. Let $\|\mathbf{R}_{t-1,i}\|_\infty \leq 3m \cdot (t-1) + 1$ for $i \in [5]$. We know that

$$\mathbf{R}_{t,i} = \Big(-\mathbf{R}'_{\mathsf{var}(t)}\widetilde{\mathbf{V}}_{t-1,\gamma_0} + (1 - x_{\mathsf{var}(t)}) \cdot \mathbf{R}_{t-1,\gamma_0}\Big) + \Big(-\mathbf{R}_{\mathsf{var}(t)}\widetilde{\mathbf{V}}_{t-1,\gamma_1} + x_{\mathsf{var}(t)} \cdot \mathbf{R}_{t-1,\gamma_1})\Big)$$

Hence, we have:

$$\begin{aligned}
\|\mathbf{R}_{t,i}\|_\infty &\leq \Big(m \cdot \Big\|\widetilde{\mathbf{V}}_{t-1,\gamma_0}\Big\|_\infty \cdot \Big\|\mathbf{R}'_{\mathsf{var}(t)}\Big\|_\infty + (1 - x_{\mathsf{var}(t)}) \cdot \|\mathbf{R}_{t-1,\gamma_0}\|_\infty\Big) \\
&\quad + \Big(m \cdot \Big\|\widetilde{\mathbf{V}}_{t-1,\gamma_0}\Big\|_\infty \cdot \big\|\mathbf{R}_{\mathsf{var}(t)}\big\|_\infty + x_{\mathsf{var}(t)} \cdot \|\mathbf{R}_{t-1,\gamma_1}\|_\infty\Big) \\
&= \big(m \cdot 1 \cdot 2 + (1 - x_{\mathsf{var}(t)}) \cdot 3m \cdot (t-1)\big) + \big(m \cdot 1 \cdot 1 + x_{\mathsf{var}(t)} \cdot 3m \cdot (t-1)\big) \\
&= 3m \cdot t + 1
\end{aligned}$$

where $\|\mathbf{R}'_i\|_\infty \leq \|\mathbf{R}^{\mathsf{c}} + \mathbf{R}_i\|_\infty \leq \|\mathbf{R}^{\mathsf{c}}\|_\infty + \|\mathbf{R}_i\|_\infty \leq 1 + 1 = 2$. With $\mathbf{R}_{\mathsf{BP}}$ being at step L, we have $\|\mathbf{R}_{\mathsf{BP}}\|_\infty \leq 3m \cdot L + 1$. Thus, $\|\mathbf{R}_{\mathsf{BP}}\|_\infty = O(m \cdot L)$.

4 Our Attribute-Based Encryption

In this section we describe our attribute-based encryption scheme for branching programs. We present the scheme for a bounded length branching programs, but note that we can trivially support unbounded length by setting modulo q to a small superpolynomial. For a family of branching programs of length bounded by L and input space $\{0,1\}^\ell$, we define the $\mathcal{ABE}$ algorithms $(\mathsf{Params}, \mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ as follows.

– $\mathsf{Params}(1^\lambda, 1^L)$: For a security parameter λ and length bound L, let the LWE dimension be $n = n(\lambda)$ and let the LWE modulus be $q = q(n, L)$. Let χ be an error distribution over $\mathbb{Z}$ and let $B = B(n)$ be an error bound. We additionally choose two Gaussian parameters: a "small" Gaussian parameter $s = s(n)$ and a "large" Gaussian parameter $\alpha = \alpha(n)$. Both these parameters are polynomially bounded (in λ, L). The public parameters $\mathsf{pp} = (\lambda, L, n, q, m, \chi, B, s, \alpha)$ are implicitly given as input to all the algorithms below.
– $\mathsf{Setup}(1^\ell)$: The setup algorithm takes as input the length of the attribute vector ℓ.
  1. Sample a matrix with a trapdoor: $(\mathbf{A}, \mathbf{T_A}) \leftarrow \mathsf{TrapSamp}(1^n, 1^m, q)$.
  2. Let $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ be the primitive matrix with the public trapdoor basis $\mathbf{T_G}$.
  3. Choose $\ell + 6$ matrices $\{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,1}\}_{i\in[5]}, \mathbf{A}^{\mathsf{c}}$ at random from $\mathbb{Z}_q^{n\times m}$. First, ℓ matrices form the LWE "public keys" for the bits of attribute vector, next 5 form the "public keys" for the initial configuration of the state vector, and the last matrix as a "public key" for a constant 1.
  4. Choose a vector $\mathbf{u} \in \mathbb{Z}_q^n$ at random.
  5. Output the master public key
$$\mathsf{mpk} := \left(\mathbf{A}, \mathbf{A}^{\mathsf{c}}, \{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{G}, \mathbf{u}\right)$$
  and the master secret key $\mathsf{msk} := (\mathbf{T_A}, \mathsf{mpk})$.
– $\mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, \mu)$: The encryption algorithm takes as input the master public key mpk, the attribute vector $\mathbf{x} \in \{0,1\}^\ell$ and a message μ.
  1. Choose an LWE secret vector $\mathbf{s} \in \mathbb{Z}_q^n$ at random.
  2. Choose noise vector $\mathbf{e} \xleftarrow{\$} \chi^m$ and compute $\psi_0 = \mathbf{A}^\intercal \mathbf{s} + \mathbf{e}$.
  3. Choose a random matrix $\mathbf{R}^{\mathsf{c}} \leftarrow \{-1, 1\}^{m\times m}$ and let $\mathbf{e}^{\mathsf{c}} = (\mathbf{R}^{\mathsf{c}})^\intercal \mathbf{e}$. Now, compute an encoding of a constant 1:
$$\psi^{\mathsf{c}} = (\mathbf{A}^{\mathsf{c}} + \mathbf{G})^\intercal \mathbf{s} + \mathbf{e}^{\mathsf{c}}$$
  4. Encode each bit $i \in [\ell]$ of the attribute vector:
     (a) Choose random matrices $\mathbf{R}_i \leftarrow \{-1, 1\}^{m\times m}$ and let $\mathbf{e}_i = \mathbf{R}_i^\intercal \mathbf{e}$.
     (b) Compute $\psi_i = (\mathbf{A}_i + x_i \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{e}_i$.
  5. Encode the initial state configuration vector $\mathbf{v}_0 = [1, 0, 0, 0, 0]$: for all $i \in [5]$,
     (a) Choose a random matrix $\mathbf{R}_{0,i} \leftarrow \{-1, 1\}^{m\times m}$ and let $\mathbf{e}_{0,i} = \mathbf{R}_{0,i}^\intercal \mathbf{e}$.
     (b) Compute $\psi_{0,i} = (\mathbf{V}_{0,i} + \mathbf{v}_0[i] \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{e}_{0,i}$.
  6. Encrypt the message μ as $\tau = \mathbf{u}^\intercal \mathbf{s} + e + \lfloor q/2 \rceil \mu$, where $e \leftarrow \chi$.
  7. Output the ciphertext
$$\mathsf{ct}_{\mathbf{x}} = \left(\mathbf{x}, \psi_0, \psi^{\mathsf{c}}, \{\psi_i\}_{i\in[\ell]}, \{\psi_{0,i}\}_{i\in[5]}, \tau\right)$$
– $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{BP})$: The key-generation algorithm takes as input the master secret key msk and a description of a branching program:
$$\mathsf{BP} := \left(\mathbf{v}_0, \left\{\mathsf{var}(t), \{\gamma_{t,i,0}, \gamma_{t,i,1}\}_{i\in[5]}\right\}_{t\in[L]}\right)$$
  The secret key $\mathsf{sk}_{\mathsf{BP}}$ is computed as follows.
  1. Homomorphically compute a "public key" matrix associated with the branching program:
$$\mathbf{V}_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{pk}}(\mathsf{BP}, \{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^{\mathsf{c}})$$
  2. Let $\mathbf{F} = [\mathbf{A} || (\mathbf{V}_{\mathsf{BP}} + \mathbf{G})] \in \mathbb{Z}_q^{n\times 2m}$. Compute $\mathbf{r}_{\text{out}} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, (\mathbf{V}_{\mathsf{BP}} + \mathbf{G}), \mathbf{T_A}, \mathbf{u}, \alpha)$ such that $\mathbf{F} \cdot \mathbf{r}_{\text{out}} = \mathbf{u}$.
  3. Output the secret key for the branching program as
$$\mathsf{sk}_{\mathsf{BP}} := (\mathsf{BP}, \mathbf{r}_{\text{out}})$$
– $\mathsf{Dec}(\mathsf{sk}_{\mathsf{BP}}, \mathsf{ct}_{\mathbf{x}})$: The decryption algorithm takes as input the secret key for a branching program $\mathsf{sk}_{\mathsf{BP}}$ and a ciphertext $\mathsf{ct}_{\mathbf{x}}$. If $\mathsf{BP}(\mathbf{x}) = 0$, output $\perp$. Otherwise,

1. Homomorphically compute the encoding of the result $\mathsf{BP}(\mathbf{x})$ associated with the public key of the branching program:

$$\psi_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{en}}(\mathsf{BP}, \mathbf{x}, \{\mathbf{A}_i, \psi_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}, \psi_{0,i}\}_{i\in[5]}, (\mathbf{A}^{\mathsf{c}}, \psi^{\mathsf{c}}))$$

2. Finally, compute $\phi = \mathbf{r}_{\text{out}}^{\intercal} \cdot [\psi || \psi_{\mathsf{BP}}]$. As we show in Lemma 8, $\phi = \mathbf{u}^{\intercal}\mathbf{s} + \lfloor q/2 \rceil \, \mu + e_\phi \pmod q$, for a short e_ϕ.
3. Output $\mu = 0$ if $|\tau - \phi| < q/4$ and $\mu = 1$ otherwise.

4.1 Correctness

Lemma 8. *Let $\mathcal{BP}$ be a family of width-5 permutation branching programs with their length bounded by L and let $\mathcal{ABE} = (\mathsf{Params}, \mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be our attribute-based encryption scheme. For a LWE dimension $n = n(\lambda)$, the parameters for $\mathcal{ABE}$ are instantiated as follows (according to Section 5):*

$$\chi = D_{\mathbb{Z},\sqrt{n}} \qquad B = O(n) \qquad m = O(n \log q)$$
$$q = \tilde{O}(n^7 \cdot L^2) \qquad \alpha = \tilde{O}(n \log q)^2 \cdot L$$

then the scheme $\mathcal{ABE}$ is correct, according to the definition in Section 2.2.

Proof. We have to show that the decryption algorithm outputs the correct message μ, given a valid set of a secret key and a ciphertext.

From Lemma 4, we have that $\psi_{\mathsf{BP}} = (\mathbf{V}_{\mathsf{BP}} + \mathbf{G})^{\intercal}\mathbf{s} + \mathbf{e}_{\mathsf{BP}}$ since $\mathsf{BP}(\mathbf{x}) = 1$. Also, from Lemma 5, we know that $\|\mathbf{e}_{\mathsf{BP}}\|_\infty = O(m \cdot L \cdot (m \cdot B)) = O(m^2 \cdot L \cdot B)$ since our input encodings have noise terms bounded by $m \cdot B$. Thus, the noise term in ϕ is bounded by:

$$\begin{aligned}
\|e_\phi\|_\infty &= m \cdot \left(\mathsf{Noise}_{\mathbf{A},0}(\psi) + \mathsf{Noise}_{\mathbf{V}_{\mathsf{BP}},1}(\psi_{\mathsf{BP}})\right) \cdot \|\mathbf{r}_{\text{out}}\|_\infty \\
&= m \cdot (B + O(m^2 \cdot L \cdot B)) \cdot \tilde{O}(n \log q)^2 \cdot L\sqrt{m} \\
&= O\big((n \log q)^6 \cdot L^2 \cdot B\big)
\end{aligned}$$

where $m = O(n \log q)$ and $\|\mathbf{r}_{\text{out}}\|_\infty \le \alpha\sqrt{m} = \tilde{O}(n \log q)^2 \cdot L\sqrt{m}$ according to Section 5. Hence, we have

$$|\tau - \phi| \le \|e\|_\infty + \|e_\phi\|_\infty = O\big((n \log q)^6 \cdot L^2 \cdot B\big) \le q/4$$

Clearly, the last inequality is satisfied when $q = \tilde{O}(n^7 \cdot L^2)$. Hence, the decryption proceeds correctly outputting the correct μ.

4.2 Security Proof

Theorem 5. *For any ℓ and any length bound L, $\mathcal{ABE}$ scheme defined above satisfies selective security game 3 for any family of branching programs BP of length L with ℓ-bit inputs, assuming hardness of $\mathsf{dLWE}_{n,q,\chi}$ for sufficiently large $n = \text{poly}(\lambda), q = \tilde{O}(n^7 \cdot L^2)$ and $\text{poly}(n)$ bounded error distribution χ. Moreover, the size of the secret keys grows polynomially with L (and independent of the width of BP).*

Proof. We define a series of hybrid games, where the first and the last games correspond to the real experiments encrypting messages μ_0, μ_1 respectively. We show that these games are indistinguishable except with negligible probability. Recall that in a selective security game, the challenge attribute vector $\mathbf{x}^*$ is declared before the Setup algorithm and all the secret key queries that adversary makes must satisfy $\mathsf{BP}(\mathbf{x}^*) = 0$. First, we define auxiliary simulated $\mathcal{ABE}^*$ algorithms.

– $\mathsf{Setup}^*(1^\lambda, 1^\ell, 1^L, \mathbf{x}^*)$: The simulated setup algorithm takes as input the security parameter λ, the challenge attribute vector $\mathbf{x}^*$, its length ℓ and the maximum length of the branching program L.

1. Choose a random matrix $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$ and a vector $\mathbf{u}$ at random.
2. Let $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ be the primitive matrix with the public trapdoor basis $\mathbf{T_G}$.
3. Choose $\ell + 6$ random matrices $\{\mathbf{R}_i\}_{i \in [\ell]}, \{\mathbf{R}_{0,i}\}_{i \in [5]}, \mathbf{R}^{\mathsf{c}}$ from $\{-1, 1\}^{m \times m}$ and set
   (a) $\mathbf{A}_i = \mathbf{A} \cdot \mathbf{R}_i - x^* \mathbf{G}$ for $i \in [\ell]$,
   (b) $\mathbf{V}_{0,i} = \mathbf{A} \cdot \mathbf{R}_{0,i} - \mathbf{v}_0[i] \cdot \mathbf{G}$ for $i \in [5]$ where $\mathbf{v}_0 = [1, 0, 0, 0, 0]$,
   (c) $\mathbf{A}^{\mathsf{c}} = \mathbf{A} \cdot \mathbf{R}^{\mathsf{c}} - \mathbf{G}$.
4. Output the master public key

$$\mathsf{mpk} := \left(\mathbf{A}, \mathbf{A}^{\mathsf{c}}, \{\mathbf{A}_i\}_{i \in [\ell]}, \{\mathbf{V}_{0,i}\}_{i \in [5]}, \mathbf{G}, \mathbf{u}\right)$$

and the secret key

$$\mathsf{msk} := \left(\mathbf{x}^*, \mathbf{A}, \mathbf{R}^{\mathsf{c}}, \{\mathbf{R}_i\}_{i \in [\ell]}, \{\mathbf{R}_{0,i}\}_{i \in [5]}\right)$$

– $\mathsf{Enc}^*(\mathsf{mpk}, \mathbf{x}^*, \mu)$: The simulated encryption algorithm takes as input $\mathsf{mpk}, \mathbf{x}^*$ and the message μ. It computes the ciphertext using the knowledge of short matrices $\{\mathbf{R}_i\}, \{\mathbf{R}_{0,i}\}, \mathbf{R}^{\mathsf{c}}$ as follows.
1. Choose a vector $\mathbf{s} \in \mathbb{Z}_q^n$ at random.
2. Choose noise vector $\mathbf{e} \xleftarrow{\$} \chi^m$ and compute $\psi_0 = \mathbf{A}^\intercal \mathbf{s} + \mathbf{e}$.
3. Compute an encoding of an identity as $\psi^{\mathsf{c}} = (\mathbf{A}^{\mathsf{c}})^\intercal \mathbf{s} + (\mathbf{R}^{\mathsf{c}})^\intercal \mathbf{e}$.
4. For all bits of the attribute vector $i \in [\ell]$ compute

$$\psi_i = (\mathbf{A}_i + x_i \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{R}_i^\intercal \mathbf{e}$$

5. For all $i \in [5]$, encode the bits of the initial state configuration vector $\mathbf{v}_0 = [1, 0, 0, 0, 0]$

$$\psi_{0,i} = (\mathbf{V}_{0,i} + \mathbf{v}_0[i] \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{R}_{0,i}^\intercal \mathbf{e}$$

6. Encrypt the message μ as $\tau = \mathbf{u}^\intercal \mathbf{s} + e + \lfloor q/2 \rceil \mu$, where $e \leftarrow \chi$.
7. Output the ciphertext

$$\mathsf{ct} = \left(\mathbf{x}, \psi_0, \{\psi_i\}_{i \in [\ell]}, \psi^{\mathsf{c}}, \{\psi_{0,i}\}_{i \in [5]}, \tau\right)$$

– $\mathsf{KeyGen}^*(\mathsf{msk}, \mathsf{BP})$: The simulated key-generation algorithm takes as input the master secret key msk and the description of the branching program BP. It computes the secret key $\mathsf{sk}_{\mathsf{BP}}$ as follows.
1. Obtain a short homomorphically derived matrix associated with the output public key of the branching program:

$$\mathbf{R}_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{SIM}}\left(\mathsf{BP}, \mathbf{x}^*, \{\mathbf{R}_i\}_{i \in [\ell]}, \{\mathbf{R}_{0,i}\}_{i \in [5]}, \mathbf{R}^{\mathsf{c}}, \mathbf{A}\right)$$

2. By the correctness of $\mathsf{Eval}_{\mathsf{SIM}}$, we have $\mathbf{V}_{\mathsf{BP}} = \mathbf{A}\mathbf{R}_{\mathsf{BP}} - \mathsf{BP}(\mathbf{x}^*) \cdot \mathbf{G}$. Let $\mathbf{F} = [\mathbf{A} || (\mathbf{V}_{\mathsf{BP}} + \mathbf{G})] \in \mathbb{Z}_q^{n \times 2m}$. Compute $\mathbf{r}_{\text{out}} \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{R}_{\mathsf{BP}}, \mathbf{T_G}, \mathbf{u}, \alpha)$ such that $\mathbf{F} \cdot \mathbf{r}_{\text{out}} = \mathbf{u}$ (this step relies on the fact that $\mathsf{BP}(\mathbf{x}^*) = 0$).
3. Output the secret key for the branching program

$$\mathsf{sk}_{\mathsf{BP}} := (\mathsf{BP}, \mathbf{r}_{\text{out}})$$

Game Sequence. We now define a series of games and then prove that all games **Game i** and **Game i+1** are either statistically or computationally indistinguishable.

– **Game 0**: The challenger runs the real ABE algorithms and encrypts message μ_0 for the challenge index $\mathbf{x}^*$.
– **Game 1**: The challenger runs the simulated ABE algorithms $\mathsf{Setup}^*, \mathsf{KeyGen}^*, \mathsf{Enc}^*$ and encrypts message μ_0 for the challenge index $\mathbf{x}^*$.
– **Game 2**: The challenger runs the simulated ABE algorithms $\mathsf{Setup}^*, \mathsf{KeyGen}^*$, but chooses a uniformly random element of the ciphertext space for the challenge index $\mathbf{x}^*$.
– **Game 3**: The challenger runs the simulated ABE algorithms $\mathsf{Setup}^*, \mathsf{KeyGen}^*, \mathsf{Enc}^*$ and encrypts message μ_1 for the challenge index $\mathbf{x}^*$.

– **Game 4**: The challenger runs the real ABE algorithms and encrypts message μ_1 for the challenge index $\mathbf{x}^*$.

Lemma 9. *The view of an adversary in* **Game 0** *is statistically indistinguishable from* **Game 1**. *Similarly, the view of an adversary in* **Game 4** *is statistically indistinguishable from* **Game 3**.

Proof. We prove for the case of **Game 0** and **Game 1**, as the other case is identical. First, note the differences between the games:

– In **Game 0**, matrix $\mathbf{A}$ is sampled using $\mathsf{TrapSamp}$ algorithm and matrices $\mathbf{A}_i, \mathbf{A}^{\mathsf{c}}, \mathbf{V}_{0,j} \in \mathbb{Z}_q^{n\times m}$ are randomly chosen for $i \in [\ell], j \in [5]$. In **Game 1**, matrix $\mathbf{A} \in \mathbb{Z}_p^{n\times m}$ is chosen uniformly at random and matrices $\mathbf{A}_i = \mathbf{A}\mathbf{R}_i - x_i^* \cdot \mathbf{G}$, $\mathbf{A}^{\mathsf{c}} = \mathbf{A}\mathbf{R}^{\mathsf{c}} - \mathbf{G}$, $\mathbf{V}_{0,j} = \mathbf{A}\mathbf{R}_{0,j} - \mathbf{v}_0[j] \cdot \mathbf{G}$ for randomly chosen $\mathbf{R}_i, \mathbf{R}^{\mathsf{c}}, \mathbf{R}_{0,j} \in \{-1,1\}^{m\times m}$.
– In **Game 0**, each ciphertext component is computed as:

$$\begin{aligned}
\psi_i &= (\mathbf{A}_i + x_i^* \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{e}_i = (\mathbf{A}_i + x_i^* \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{R}_i^\intercal \mathbf{e} \\
\psi^{\mathsf{c}} &= (\mathbf{A}^{\mathsf{c}} + \mathbf{G})^\intercal \mathbf{s} + \mathbf{e}^1 = (\mathbf{A}^{\mathsf{c}} + \mathbf{G})^\intercal \mathbf{s} + (\mathbf{R}^{\mathsf{c}})^\intercal \mathbf{e} \\
\psi_{0,j} &= (\mathbf{V}_{0,j} + \mathbf{v}_0[j] \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{e}_i = (\mathbf{V}_{0,j} + \mathbf{v}_0[j] \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{R}_{0,j}^\intercal \mathbf{e}
\end{aligned}$$

On the other hand, in **Game 1** each ciphertext component is computed as:

$$\psi_i = (\mathbf{A}_i + x_i^* \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{R}_i^\intercal \mathbf{e} = (\mathbf{A}\mathbf{R}_i)^\intercal \mathbf{s} + \mathbf{R}_i^\intercal \mathbf{e} = \mathbf{R}_i^\intercal \left(\mathbf{A}^\intercal \mathbf{s} + \mathbf{e}\right)$$

Similarly, $\psi^{\mathsf{c}} = (\mathbf{R}^{\mathsf{c}})^\intercal (\mathbf{A}^\intercal \mathbf{s} + \mathbf{e})$ and $\psi_{0,j} = \mathbf{R}_{0,j}^\intercal (\mathbf{A}\mathbf{s} + \mathbf{e})$.
– Finally, in **Game 0** the vector $\mathbf{r}_{\text{out}}$ is sampled using $\mathsf{SampleLeft}$, whereas in **Game 1** it is sampled using $\mathsf{SampleRight}$ algorithm.

For sufficiently large α (See Section-5), the distributions produced in two games are statistically indistinguishable. This follows readily from [AFV11, Lemma 4.3], Theorem-3 and Theorem-4. We will provide the proof here for completeness.

We would like to prove that the tuple $\mathbf{A}, \{\mathbf{A}_i, \psi_i\}_{i\in[\ell]}, \mathbf{A}^{\mathsf{c}}, \psi^{\mathsf{c}}, \{\mathbf{V}_{0,j}, \psi_{0,j}\}_{j\in[5]}$ in **Game 0** is statistically indistinguishable from the set from **Game 1**. The generalisation of left-over hash lemma [DORS08,ABB10] states that, for two matrices $\mathbf{R}_i \xleftarrow{\$} \{-1,1\}^{m\times m}, \mathbf{A}, \mathbf{A}_i \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ and any vector $\mathbf{e} \in \mathbb{Z}_q^m$, the following is true, when q is square-free (q does not have a square of a prime number as its factor): $(\mathbf{A}, \mathbf{A}\mathbf{R}_i, \mathbf{R}_i^\intercal \mathbf{e},) \approx_s (\mathbf{A}, \mathbf{A}_i, \mathbf{R}_i^\intercal \mathbf{e})$. With the matrices $\mathbf{R}_i, \mathbf{R}_{0,j}$ independently chosen from $\{-1,1\}^{m\times m}$ we can extend this to have:

$$\begin{aligned}
&\left(\mathbf{A}, (\mathbf{A}\mathbf{R}_i, \mathbf{R}_i^\intercal \mathbf{e}), (\mathbf{A}\mathbf{R}^{\mathsf{c}}, (\mathbf{R}^{\mathsf{c}})^\intercal \mathbf{e}), \left(\mathbf{A}\mathbf{R}_{0,j}, \mathbf{R}_{0,j}^\intercal \mathbf{e}\right)\right) \\
&\approx_s \left(\mathbf{A}, (\mathbf{A}_i, \mathbf{R}_i^\intercal \mathbf{e}), (\mathbf{A}^{\mathsf{c}}, (\mathbf{R}^{\mathsf{c}})^\intercal \mathbf{e}), \left(\mathbf{V}_{0,j}, \mathbf{R}_{0,j}^\intercal \mathbf{e}\right)\right)
\end{aligned}$$

Hence, for every fixed matrix $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$ and every bit $x_i^*, \mathbf{v}_0[j] \in \{0,1\}$,

$$\begin{aligned}
&\left(\mathbf{A}, (\mathbf{A}\mathbf{R}_i - x_i^* \cdot \mathbf{G}, \mathbf{R}_i^\intercal \mathbf{e}), (\mathbf{A}\mathbf{R}^{\mathsf{c}} - \mathbf{G}, (\mathbf{R}^{\mathsf{c}})^\intercal \mathbf{e}), \left(\mathbf{A}\mathbf{R}_{0,j} - \mathbf{v}_0[j], \mathbf{R}_{0,j}^\intercal \mathbf{e}\right)\right) \\
&\approx_s \left(\mathbf{A}, (\mathbf{A}_i, \mathbf{R}_i^\intercal \mathbf{e}), (\mathbf{A}^{\mathsf{c}}, (\mathbf{R}^{\mathsf{c}})^\intercal \mathbf{e}), \left(\mathbf{V}_{0,j}, \mathbf{R}_{0,j}^\intercal \mathbf{e}\right)\right)
\end{aligned}$$

Now, we can extend this statistical indistinguishability to the joint distribution of these tuples for all $i \in [\ell], j \in [5]$, since the matrices $\mathbf{R}_i, \mathbf{R}_{0,j}$ are independently chosen from $\{-1,1\}^{m\times m}, \forall i \in [\ell], j \in [5]$. Thus,

$$\begin{aligned}
&\left(\mathbf{A}, \left(\{\mathbf{A}\mathbf{R}_i - x_i^* \cdot \mathbf{G}, \mathbf{R}_i^\intercal \mathbf{e}\}_{i\in[\ell]}\right), (\mathbf{A}\mathbf{R}^{\mathsf{c}} - \mathbf{G}, (\mathbf{R}^{\mathsf{c}})^\intercal \mathbf{e}), \left(\{\mathbf{A}\mathbf{R}_{0,j} - \mathbf{v}_0[j], \mathbf{R}_{0,j}^\intercal \mathbf{e}\}_{j\in[5]}\right)\right) \\
&\approx_s \left(\mathbf{A}, \left(\{\mathbf{A}_i, \mathbf{R}_i^\intercal \mathbf{e}\}_{l\in[\ell]}\right), (\mathbf{A}^{\mathsf{c}}, (\mathbf{R}^{\mathsf{c}})^\intercal \mathbf{e}), \left(\{\mathbf{V}_{0,j}, \mathbf{R}_{0,j}^\intercal \mathbf{e}\}_{j\in[5]}\right)\right)
\end{aligned}$$

Also, due to the fact that applying any function to two statistically indistinguishable entities results in entities which are atleast as statistically indistinguishable as the original pair, we eventually get:

$$\begin{aligned}
&\left(\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e}, \left(\{\mathbf{A}\mathbf{R}_i - x_i^* \cdot \mathbf{G}, (\mathbf{A}\mathbf{R}_i)^\intercal \mathbf{s} + \mathbf{R}_i^\intercal \mathbf{e}\}_{i\in[\ell]}\right), \left(\mathbf{A}\mathbf{R}^{\mathsf{c}} - \mathbf{G}, (\mathbf{A}\mathbf{R}^{\mathsf{c}})^\intercal + (\mathbf{R}^{\mathsf{c}})^\intercal \mathbf{e}\right), \right. \\
&\qquad \left. \left(\{\mathbf{A}\mathbf{R}_{0,j} - \mathbf{v}_0[j], (\mathbf{A}\mathbf{R}_{0,j})^\intercal + \mathbf{R}_{0,j}^\intercal \mathbf{e}\}_{j\in[5]}\right)\right) \\
&\approx_s \left(\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e}, (\mathbf{A}_i, (\mathbf{A}_i + x_i \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{R}_i^\intercal \mathbf{e}), \left(\{\mathbf{A}^{\mathsf{c}}, (\mathbf{A}^{\mathsf{c}} + \mathbf{G})^\intercal \mathbf{s} + (\mathbf{R}^{\mathsf{c}})^\intercal \mathbf{e}\}_{i\in[\ell]}\right), \right. \\
&\qquad \left. \left(\{\mathbf{V}_{0,j}, (\mathbf{V}_{0,j} + \mathbf{v}_0[j] \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{R}_{0,j}^\intercal \mathbf{e}\}_{j\in[5]}\right)\right)
\end{aligned}$$

Thus, we can conclude that the public parameters in **Game 0** are statistically indistinguishable from those in **Game 1**, and that the output of Enc is statistically indistinguishable from that of Enc^*. When the "large" Gaussian parameter α is chosen appropriately (as discussed in 5), the output of the KeyGen and KeyGen^* algorithms are also statistically indistinguishable. Thus, the view of an adversary in **Game 0** is statistically indistinguishable from the view in **Game 1**.

Lemma 10. *If the decisional* LWE *assumption holds, then the view of an adversary in* **Game 1** *is computationally indistinguishable from* **Game 2**. *Similarly, if the decisional* LWE *assumption holds, then the view of an adversary in* **Game 3** *is computationally indistinguishable from* **Game 2**.

Proof. Assume there exist an adversary Adv that distinguishes between **Game 1** and **Game 2**. We show how to break LWE problem given a challenge $\{(\mathbf{a}_i, y_i)\}_{i\in[m+1]}$ where each y_i is either a random sample in $\mathbb{Z}_q$ or $\mathbf{a}_i^\intercal \cdot \mathbf{s} + e_i$ (for a fixed, random $\mathbf{s} \in \mathbb{Z}_q^n$ and a noise term sampled from the error distribution $e_i \leftarrow \chi$). Let $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m] \in \mathbb{Z}_q^{n\times m}$ and $\mathbf{u} = \mathbf{a}_{m+1}$. Let $\psi_0^* = [y_1, y_2, \ldots, y_m]$ and $\tau = y_{m+1} + \mu \lfloor q/2 \rfloor$.

Now, run the simulated Setup^* algorithm where $\mathbf{A}, \mathbf{u}$ are as defined above. Run the simulated KeyGen^* algorithm. Finally, to simulate the challenge ciphertext set ψ_0^*, τ as defined above and compute

$$\psi_i = \mathbf{R}_i^\intercal \cdot \psi_0^* = \mathbf{R}_i^\intercal \left(\mathbf{A}^\intercal \mathbf{s} + \mathbf{e}\right)$$

for $i \in [\ell]$. Similarly, $\psi^{\mathsf{c}} = (\mathbf{R}^{\mathsf{c}})^\intercal(\mathbf{A}^\intercal\mathbf{s} + \mathbf{e})$ and $\psi_{0,j} = \mathbf{R}_{0,j}^\intercal(\mathbf{A}^\intercal\mathbf{s} + \mathbf{e})$, for $j \in [5]$. Note that if y_i's are LWE samples, then this corresponds exactly to the **Game 1**. Otherwise, the ciphertext corresponds to an independent random sample as in **Game 2** by the left-over hash lemma. Thus, an adversary which distinguishes between **Game 1** and **Game 2** can also be used to break the decisional LWE assumption with almost the same advantage. The computational indistinguishability of **Game 3** and **Game 2** follows from the same argument.

To conclude, note that **Game 0** corresponds to an encryption of the challenge message μ_0 in the real experiment and **Game 4** corresponds to an encryption of the challenge message μ_1 (also in the real experiment). Hence, by the standard hybrid argument, no adversary can distinguish between encryptions of μ_0 and μ_1 with non-negligible advantage establishing the selective security of our ABE scheme.

5 Parameter Selection

This section provides a concise description on the selection of parameters for our scheme, so that both correctness (see Lemma 8) and security (see Theorem 5) of our scheme are satisfied.

For a family of width-5 permutation branching programs $\mathcal{BP}$ of bounded length L, with the LWE dimension n, the parameters can be chosen as follows: (we start with an arbitrary q and we will instantiate it later)

- The parameter $m = O(n \log q)$. The error distribution $\chi = D_{\mathbb{Z},\sqrt{n}}$ with parameter $\sigma = \sqrt{n}$. And, the error bound $B = O(\sigma\sqrt{n}) = O(n)$.
- The "large" Gaussian parameter $\alpha = \alpha(n, L)$ is chosen such that the output of the $\mathsf{SampleLeft}$ and the $\mathsf{SampleRight}$ algorithms are statistically indistinguishable from each other, when provided with the same set of inputs $\mathbf{F}$ and $\mathbf{u}$. The $\mathsf{SampleRight}$ algorithm (Algorithm 3) requires

$$\alpha > \|\mathbf{T}_{\mathbf{G}}\|_{\mathsf{GS}} \cdot \|\mathbf{R}_{\mathsf{BP}}\| \cdot \omega(\sqrt{\log m}) \tag{3}$$

From Lemma 7, we have that $\|\mathbf{R}_{\mathsf{BP}}\|_\infty = O(m \cdot L)$. Then, we get:

$$\|\mathbf{R}_{\mathsf{BP}}\| := \sup_{\mathbf{x}\in S^{m-1}} \|\mathbf{R}_{\mathsf{BP}} \cdot \mathbf{x}\| \le m \cdot \|\mathbf{R}_{\mathsf{BP}}\|_\infty \le O(m^2 \cdot L)$$

Finally, from Equation 3, the value of α required for the $\mathsf{SampleRight}$ algorithm is

$$\alpha \ge O(m^2 \cdot L) \cdot \omega(\sqrt{\log m}) \tag{4}$$

The value of the parameter α required for the SampleLeft algorithm (Algorithm 3) is

$$\alpha \geq \|\mathbf{T}_{\mathbf{A}}\|_{\mathsf{GS}} \cdot \omega(\sqrt{\log 2m}) \geq O(\sqrt{n \log q}) \cdot \omega(\sqrt{\log 2m}) \tag{5}$$

Thus, to satisfy both Equation 4 and Equation 5, we set the parameter

$$\alpha \geq O(m^2 \cdot L) \cdot \omega(\sqrt{\log m}) = \tilde{O}(n \log q)^2 \cdot L$$

When our scheme is instantiated with these parameters, the correctness (see Lemma 8) of the scheme is satisfied when $O((n \log q)^6 \cdot L^2 \cdot B) < q/4$. Clearly, this condition is satisfied when $q = \tilde{O}(n^7 L^2)$.

6 Extensions

We note a few possible extensions on our basic construction that lead to further efficiency improvements. First, we can support arbitrary width branching programs by appropriately increasing the dimension of the state vector in the encryption. Second, we can switch to an arithmetic setting, similarly as it was done in [BGG+14].

References

ABB10. S. Agrawal, D. Boneh, and X. Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, pages 553–572, 2010.

AFV11. S. Agrawal, D. M. Freeman, and V. Vaikuntanathan. Functional encryption for inner product predicates from learning with errors. In *ASIACRYPT*, pages 21–40, 2011.

Ajt99. M. Ajtai. Generating hard instances of the short basis problem. In *ICALP*, pages 1–9, 1999.

AKS01. M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *STOC*, pages 601–610, 2001.

ALdP11. N. Attrapadung, B. Libert, and E. de Panafieu. Expressive key-policy attribute-based encryption with constant-size ciphertexts. In *PKC*. 2011.

APG+11. J. Akinyele, M. Pagano, M. Green, C. Lehmann, Z. Peterson, and A. Rubin. Securing electronic medical records using attribute-based encryption on mobile devices. In *SPSM*, pages 75–86, 2011.

Bar86. D A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc1. In *STOC*, pages 1–5, 1986.

BF03. D. Boneh and M. K. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal on Computing*, 32(3):586–615, 2003.

BGG+14. 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*, pages 533–556, 2014.

Boy13. X. Boyen. Attribute-based functional encryption on lattices. In *TCC*, pages 122–142, 2013.

BSW12. D. Boneh, A. Sahai, and B. Waters. Functional encryption: a new vision for public-key cryptography. *Commun. ACM*, 55(11):56–64, 2012.

BV14. Z. Brakerski and V. Vaikuntanathan. Lattice-based fhe as secure as pke. In *ITCS*, pages 1–12, 2014.

CHKP12. D. Cash, D. Hofheinz, E. Kiltz, and C. Peikert. Bonsai trees, or how to delegate a lattice basis. *J. Cryptology*, 25(4):601–639, 2012.

CLT13. J. Coron, T. Lepoint, and M. Tibouchi. Practical multilinear maps over the integers. In *CRYPTO*, pages 476–493, 2013.

Coc01. C. Cocks. An identity based encryption scheme based on quadratic residues. In *Cryptography and Coding, 8th IMA International Conference*, pages 360–363, 2001.

DORS08. Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM Journal on Computing*, 38(1):97–139, 2008.

EMN+09. K. Emura, A. Miyaji, A. Nomura, K. Omote, and M. Soshi. A ciphertext-policy attribute-based encryption scheme with constant ciphertext length. In *Information Security Practice and Experience*, pages 13–23, 2009.

GGH13a. S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, pages 1–17, 2013.

GGH+13b. S. Garg, C. Gentry, S. Halevi, M. Raykova, A Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49, 2013.

GGH+13c. S. Garg, C. Gentry, S. Halevi, A. Sahai, and B. Waters. Attribute-based encryption for circuits from multilinear maps. In *CRYPTO*, pages 479–499, 2013.

GGH15. C. Gentry, S. Gorbunov, and S. Halevi. Graph-induced multilinear maps from lattices. In *TCC*, pages 498–527. 2015.

GKP+13. S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, pages 555–564, 2013.

GPSW06. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, pages 89–98, 2006.

GPV08. C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

GVW12. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, pages 162–179, 2012.

GVW13. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Attribute-based encryption for circuits. In *STOC*, pages 545–554, 2013.

GVW15a. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Predicate encryption for circuits from standard lattices. In *To appear in CRYPTO*, 2015.

GVW15b. S. Gorbunov, V. Vaikuntanathan, and D. Wichs. Leveled fully homomorphic signatures from standard lattices. In *STOC*, 2015.

LOS+10. A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, pages 62–91, 2010.

LW10. A. Lewko and B. Waters. New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In *TCC*, pages 455–479, 2010.

LYZ+13. M. Li, S. Yu, Y. Zheng, K. Ren, and W. Lou. Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. *IEEE Transactions on Parallel and Distributed Systems*, 24(1):131–143, 2013.

MP12. D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, pages 700–718, 2012.

MV10. D. Micciancio and P. Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. In *STOC*, pages 351–358, 2010.

Pei09. C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, pages 333–342, 2009.

PPM+14. J. Papanis, S. Papapanagiotou, A. Mousas, G. Lioudakis, D. Kaklamani, and I. Venieris. On the use of attribute-based encryption for multimedia content protection over information-centric networks. *Transactions on Emerging Telecommunications Technologies*, 25(4):422–435, 2014.

PRV12. B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, pages 422–439, 2012.

Reg09. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.

SW05. A. Sahai and B. Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

Wat09. B. Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In *CRYPTO*, pages 619–636, 2009.