Cryptanalysis of Boyen's Attribute-Based Encryption Scheme in TCC 2013

Shweta Agrawal[1], Rajarshi Biswas[1], Ryo Nishimaki[2], Keita Xagawa[2], Xiang Xie[3], and Shota Yamada[4]

[1] IIT Madras, Chennai, India
shweta.a@cse.iitm.ac.in, rajarshi.369@outlook.in
[2] NTT Secure Platform Laboratories, Tokyo, Japan
[ryo.nishimaki.zk,keita.xagawa.zv]@hco.ntt.co.jp
[3] Shanghai Key Laboratory of Privacy-Preserving Computation, China
xiexiang@matrixelements.com
[4] National Institute of Advanced Industrial Science and Technology (AIST), Tokyo
yamada-shota@aist.go.jp

Abstract. In TCC 2013, Boyen suggested the first lattice based construction of attribute based encryption (ABE) for the circuit class NC_1. Unfortunately, soon after, a flaw was found in the security proof of the scheme. However, it remained unclear whether the scheme is actually insecure, and if so, whether it can be repaired. Meanwhile, the construction has been heavily cited and continues to be extensively studied due to its technical novelty. In particular, this is the first lattice based ABE which uses linear secret sharing schemes (LSSS) as a crucial tool to enforce access control.

In this work, we show that the scheme is in fact insecure. To do so, we provide a polynomial-time attack that completely breaks the security of the scheme. We suggest a route to fix the security of the scheme, via the notion of *admissible* linear secret sharing schemes (LSSS) and instantiate these for the class of DNFs. Subsequent to our work, Datta, Komargodski and Waters (Eurocrypt 2021) provided a construction of admissible LSSS for NC_1 and resurrected Boyen's claimed result.

1 Introduction

The long-standing problem of Attribute-Based Encryption (ABE) from Learning with Errors (LWE) was finally resolved in 2013 by two independent works: Gorbunov, Vaikuntanathan and Wee [GVW13] provided an ABE for P, and Boyen provided an ABE for NC_1 [Boy13a]. Subsequently, Boneh et al. [BGG+14] also provided an ABE for arithmetic (rather than Boolean) circuits. These works marked important progress in the area of lattice based cryptography and have had many follow-ups and much impact.

Moreover, the techniques used by these works are very different and have led to generalizations in diverse directions. Here, the construction of Boyen [Boy13a]

was particularly unlike the others since it used linear secret sharing schemes (LSSS) as a crucial tool. While using LSSS in pairing-based ABE constructions is common [GPSW06, LOS+10, LW11b], this was the only *lattice* based ABE to leverage this tool until the very recent work of Datta, Komargodski and Waters [DKW21]. To this day, there are several outstanding open problems in lattice-based ABE that have solutions in the pairings world – notable examples are *ciphertext policy* ABE [BSW07, Wat11], adaptive security [LOS+10, OT10, LW12]. Since pairing-based solutions to these problems make crucial use of the tool of LSSS, there have been multiple attempts to generalize the construction by Boyen to resolve these problems.

In this work, we show that Boyen's ABE construction is insecure, if the scheme is instantiated by the linear secret sharing scheme specified in the paper. We provide a polynomial-time attack that completely breaks the security of the scheme. We examine possible directions to repair the scheme and discuss the technical challenges in instantiating these for circuit class NC_1. Subsequent to our work, the very recent work of Datta, Komargodski and Waters [DKW21] instantiated this approach for NC_1 and resurrected Boyen's claimed result.

Since our attack is quite simple, we provide the formal description directly. We refer the reader to Section 2 for some preliminary definitions, to Section 3 for a recap of Boyen's scheme, to Section 4 for a formal description of our attack, and Section 5 for a discussion of possible approaches to fix the construction.

2 Preliminaries

Notation: Let $\lambda \in \mathbb{N}^+$ be the security parameter. For a positive integer n, $[n]$ denotes $\{1, 2, \ldots, n\}$. A non negative function $\mathsf{negl}(\lambda)$ is negligible if for every polynomial $p(\lambda)$, it holds that $\mathsf{negl}(\lambda) \leq 1/p(\lambda)$ for sufficiently large $\lambda \in \mathbb{N}$. We use the notation $x \xleftarrow{\$} \mathcal{X}$ to denote the process of choosing a value x uniformly at random from distribution $\mathcal{X}$. As usual, PPT stands for probabilistic polynomial-time.

Statistical distance: The statistical distance between two random variables X and Y over a finite domain D is defined as

$$\mathsf{SD}(X, Y) := \frac{1}{2} \sum_{\alpha \in D} \Big| \Pr[X = \alpha] - \Pr[Y = \alpha] \Big|.$$

Two random variables are δ-close if $\mathsf{SD}(X, Y) \leq \delta$. Two distribution ensembles $\{X_\lambda\}_{\lambda \in \mathbb{N}}$ and $\{Y_\lambda\}_{\lambda \in \mathbb{N}}$ are statistically indistinguishable if $\mathsf{SD}(X_\lambda, Y_\lambda)$ is negligible in λ. We say that these random variables are computationally indistinguishable if for every PPT algorithm $\mathcal{A}$ it holds that

$$\Big| \Pr_{x \leftarrow X_\lambda} [\mathcal{A}(1^\lambda, x) = 1] - \Pr_{y \leftarrow Y_\lambda} [\mathcal{A}(1^\lambda, y) = 1] \Big| \leq \mathsf{negl}(\lambda).$$

In the following, let $\mathsf{SampZ}(\gamma)$ be a sampling algorithm for the truncated discrete Gaussian distribution over $\mathbb{Z}$ with parameter $\gamma > 0$ whose support is restricted to $z \in \mathbb{Z}$ such that $|z| \leq \sqrt{n}\gamma$.

2.1 Lattice Preliminaries

For a vector $\mathbf{v} \in \mathbb{R}^n$, let $\|\mathbf{v}\|$, $\|\mathbf{v}\|_\infty$ denote the Euclidean and sup norm, respectively. For two matrices $\mathbf{A}$ and $\mathbf{B}$, $[\mathbf{A} \mid \mathbf{B}]$ denotes their vertical concatenation and $[\mathbf{A}; \mathbf{B}]$ denotes their horizontal concatenation .

For any (ordered) set $\mathbf{S} = \{\mathbf{s}_1, \cdots, \mathbf{s}_k\} \subset \mathbb{R}^m$ of linearly independent vectors, let $\tilde{\mathbf{S}} = \{\tilde{\mathbf{s}}_1, \cdots, \tilde{\mathbf{s}}_k\}$ denote its Gram-Schmidt orthogonalization, defined iteratively as follows: $\tilde{\mathbf{s}}_1 = \mathbf{s}_1$, and for each $i = 2, \cdots, k$, the vector $\tilde{\mathbf{s}}_i$ is the component of $\mathbf{s}_i$ orthogonal to $\mathrm{Span}(\mathbf{s}_1, \ldots, \mathbf{s}_{i-1})$.

A lattice $\mathcal{L}$ of $\mathbb{R}^n$ is a discrete subgroup of $\mathbb{R}^n$. We will only consider full-rank integer lattices, i.e., $\mathcal{L}$ spans $\mathbb{R}^n$ with real coefficients and $\mathcal{L} \subseteq \mathbb{Z}^n$. A basis of $\mathcal{L}$ is an ordered set $\mathbf{B} = (\mathbf{B}_1, ..., \mathbf{B}_n)$ such that

$$\mathcal{L} = \mathcal{L}(\mathbf{B}) = \mathbf{B} \cdot \mathbb{Z}^n = \left\{ \textstyle\sum_{i=1}^n c_i \cdot \mathbf{B}_i \ : \ c_i \in \mathbb{Z} \right\}.$$

By convention, $\mathbf{B}_i$ are column vectors and $\mathbf{B} \cdot \mathbf{k} = k_1\mathbf{B}_1 + \cdots + k_n\mathbf{B}_n$, where $\mathbf{k}$ is a column vector.

2.2 Lattice Trapdoors

We list all known results about lattice trapdoors in the following lemma.

Lemma 2.1. *Let $n, m, q > 0$ be integers with q prime. There exists polynomial time algorithms with the properties below:*

- $\mathsf{TrapGen}(1^n, 1^m, q) \to (\mathbf{A}, \mathbf{T}_\mathbf{A})$ *[Ajt99, AP11, MP12]: A randomized algorithm that outputs $(\mathbf{A}, \mathbf{T}_\mathbf{A})$ where $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ for some $m = \mathcal{O}(n \log q)$, the distribution of $\mathbf{A}$ is 2^{-n} close to uniform, and $\|\tilde{\mathbf{T}}_\mathbf{A}\| \le \tau_0 = O(\sqrt{n \log q})$.*
- $\mathsf{ExtBasis}(\mathbf{A}, \mathbf{T}_\mathbf{A}, \mathbf{B}) \to \mathbf{T}_{[\mathbf{A}|\mathbf{B}]}$ *[CHKP12]: A deterministic algorithm that given full-rank matrices $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{B} \in \mathbb{Z}_q^{n \times \bar{m}}$ and a basis $\mathbf{T}_\mathbf{A}$ of $\Lambda_q^\perp(\mathbf{A})$, outputs a basis $\mathbf{T}_{[\mathbf{A}|\mathbf{B}]}$ of $\Lambda_q^\perp([\mathbf{A} \mid \mathbf{B}])$ such that $\|\tilde{\mathbf{T}}_\mathbf{A}\| = \|\tilde{\mathbf{T}}_{[\mathbf{A}|\mathbf{B}]}\|$.*
- $\mathsf{RndBasis}(\mathbf{T}, \tau) \to \mathbf{S}$ *[CHKP12]: A randomized algorithm that given a basis $\mathbf{T} \in \mathbb{Z}^{m \times m}$ and parameter $\tau \ge \|\tilde{\mathbf{T}}\| \cdot \omega(\sqrt{\log n})$, outputs a new basis $\mathbf{S}$ such that $L(\mathbf{T}) = L(\mathbf{S})$ and $\|\tilde{\mathbf{S}}\| \le \tau \cdot \sqrt{m}$.*
- $\mathsf{SamplePre}(\mathbf{A}, \mathbf{T}_\mathbf{A}, \mathbf{U}, \sigma) \to \mathbf{R}$ *[GPV08]: A randomized algorithm that given $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, and a basis $\mathbf{T}_\mathbf{A} \in \mathbb{Z}^{m \times m}$ of $\Lambda_q^\perp(\mathbf{A})$, $\mathbf{U} \in \mathbb{Z}_q^{n \times k}$, and $\sigma = \|\tilde{\mathbf{T}}_\mathbf{A}\| \cdot \omega(\sqrt{\log n})$, outputs a random sample $\mathbf{R}$ whose row's distributions are statistically close to $D_\sigma(\Lambda_q^\perp(\mathbf{A}))$ conditioned on $\mathbf{A} \cdot \mathbf{R} \equiv \mathbf{U} \pmod q$.*

2.3 Key-Policy Attribute-Based Encryption

Definition 2.1. *A key-policy attribute-based encryption scheme* kpABE *for a class of policies $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ where $\mathcal{F}_\lambda = \{f : \mathcal{X}_\lambda \to \{0,1\}\}$ is a tuple* (Setup, Extract, Encrypt, Decrypt) *of PPT algorithms with the following properties:*

- $\mathsf{Setup}(1^\lambda, U) \to (\mathsf{Pub}, \mathsf{Msk})$**:** *The setup algorithm takes the security parameter λ and the description of attribute universe U as input and outputs the public parameters* Pub *and a master secret key* Msk*.*

$\mathsf{Extract}(\mathsf{Pub}, \mathsf{Msk}, \mathsf{Policy}) \to \mathsf{Key}_{\mathsf{Policy}}$**:** *The key-generation algorithm takes the public parameters* Pub*, the master secret key* Msk*, and an access policy* $\mathsf{Policy} \in \mathcal{F}_\lambda$ *as input. It outputs a private key* $\mathsf{Key}_{\mathsf{Policy}}$*.*

$\mathsf{Encrypt}(\mathsf{Pub}, \mathsf{Attrib}, \mathsf{Msg}) \to \mathsf{Ctx}_{\mathsf{Attrib}}$**:** *The encryption algorithm takes as input the public parameters* Pub*, a set of attributes* $\mathsf{Attrib} \in \mathcal{X}_\lambda$*, and a message* Msg*. The algorithm outputs a ciphertext* Ctx_x *which is an encryption of message* Msg *labelled with a set of attributes* x*.*

$\mathsf{Decrypt}(\mathsf{Pub}, \mathsf{Key}_{\mathsf{Policy}}, \mathsf{Ctx}_{\mathsf{Attrib}}) \to \mathsf{Msg}/\perp$**:** *The decryption algorithm takes as input the public parameters* Pub*, a key* $\mathsf{Key}_{\mathsf{Policy}}$*, a ciphertext* $\mathsf{Ctx}_{\mathsf{Attrib}}$*, and outputs a message* Msg *or a rejection symbol* $\perp$*.*

Correctness: For correctness, we require that there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all sufficiently large $\lambda \in \mathbb{N}$, for every policy $\mathsf{Policy} \in \mathcal{F}_\lambda$, a set of attributes $\mathsf{Attrib} \in \mathcal{X}_\lambda$ where Attrib satisfies Policy, and message Msg, it holds that

$$\Pr\left[\mathsf{Msg} = \mathsf{Msg}' \;\middle|\; \begin{array}{c} (\mathsf{Pub}, \mathsf{Msk}) \leftarrow \mathsf{Setup}(1^\lambda, U); \\ \mathsf{Key}_{\mathsf{Policy}} \leftarrow \mathsf{Extract}(\mathsf{Pub}, \mathsf{Msk}, \mathsf{Policy}); \\ \mathsf{Ctx}_{\mathsf{Attrib}} \leftarrow \mathsf{Encrypt}(\mathsf{Pub}, \mathsf{Attrib}, \mathsf{Msg}); \\ \mathsf{Msg}' \leftarrow \mathsf{Decrypt}(\mathsf{Pub}, \mathsf{Key}_{\mathsf{Policy}}, \mathsf{Ctx}_{\mathsf{Attrib}}) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

over the choice of the random coins used in the algorithms.

Security model for key-policy ABE: We review a security model for key-policy ABE schemes [SW05, GPSW06]. The game-based security model allows the adversary to query policies for any private keys that cannot be used to decrypt the challenge ciphertext. The security definition guarantees that the adversary will choose to be challenged on an encryption to a set of attributes $\mathsf{Attrib}^\dagger$ and can ask for any private key for access policy Policy such that $\mathsf{Attrib}^\dagger$ does not satisfy Policy. Below is the formal security game.

Definition 2.2 (Selective security). *We say that a KP-ABE scheme* $\mathsf{kpABE} = (\mathsf{Setup}, \mathsf{Extract}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *over a policy space* $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ *and a message space* $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ *is* selectively secure *if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}()$ *such that*

$$\mathsf{Adv}^{\mathsf{sel}}_{\mathsf{kpABE},\mathcal{A}}(\lambda) := \left|\Pr\left[\mathsf{Exp}^{\mathsf{sel}}_{\mathsf{kpABE},\mathcal{A}}(\lambda, 0) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{sel}}_{\mathsf{kpABE},\mathcal{A}}(\lambda, 1) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

for all sufficiently large $\lambda \in \mathbb{N}$*, where for each* $b \in \{0, 1\}$ *and* $\lambda \in \mathbb{N}$*, the experiment* $\mathsf{Exp}^{\mathsf{sel}}_{\mathsf{kpABE},\mathcal{A}}$ *between the adversary* $\mathcal{A}$ *and a challenger is defined as follows:*

1. ***Target****: The adversary submits the challenge set of attributes* $\mathsf{Attrib}^\dagger$*.*
2. ***Setup****: The challenger samples* $(\mathsf{Pub}, \mathsf{Msk}) \leftarrow \mathsf{Setup}(1^\lambda, U)$ *and gives* Pub *to the adversary.*
3. ***Key Query Phase 1****: The adversary can adaptively query the challenger with any function* Policy*. For each query, the challenger replies with* $\mathsf{Key}_{\mathsf{Policy}} \leftarrow \mathsf{Extract}(\mathsf{Pub}, \mathsf{Msk}, \mathsf{Policy})$ *as long as* $\mathsf{Attrib}^\dagger$ *does not satisfy* Policy*.*

4. ***Challenge***: *The adversary submits a pair of messages* $(\mathsf{Msg}_0, \mathsf{Msg}_1)$ *and the challenger replies with* $\mathsf{Ctx}_{\mathsf{Attrib}^\dagger} \leftarrow \mathsf{Encrypt}(\mathsf{Pub}, \mathsf{Attrib}^\dagger, \mathsf{Msg}_b)$.
5. ***Key Query Phase 2***: *Again, the adversary can adaptively query the challenger with any function* Policy. *For each query, the challenger replies with* $\mathsf{Key}_{\mathsf{Policy}} \leftarrow \mathsf{Extract}(\mathsf{Pub}, \mathsf{Msk}, \mathsf{Policy})$ *as long as* $\mathsf{Attrib}^\dagger$ *does not satisfy* Policy.
6. ***Output*** : *The adversary outputs a bit* b' *which is defined as the output of the experiment.*

Remark 2.1 (Static Security.). We can consider weaker security notion than the above selective security that we call static security, where the adversary chooses the key policies for which it makes the key queries at the beginning of the game in addition to the target attribute.

2.4 Linear Secret Sharing Scheme

We follow the notion and notation in Beimel's survey [Bei11].

Definition 2.3 (Access structure [Bei11]). *Let* $\mathcal{P} = \{P_1, P_2, \cdots, P_n\}$ *be a set of parties. A collection* $\mathbb{A} \subseteq 2^{\{P_1, P_2, \cdots, P_n\}}$ *is monotone if for any* B *and* C *: if* $B \in \mathbb{A}$ *and* $B \subset C$ *then* $C \in \mathbb{A}$. *An access structure (respectively, monotone access structure) is a collection (respectively, monotone collection)* $\mathbb{A}$ *of nonempty subsets of* $\{P_1, P_2, \cdots, P_n\}$, *i.e.,* $\mathbb{A} \subseteq 2^{\{P_1, P_2, \cdots, P_n\}} \setminus \{\emptyset\}$. *The sets in* $\mathbb{A}$ *are called the authorized sets, and the sets not in* $\mathbb{A}$ *are called the unauthorized sets.*

In an ABE system, the role of the parties is defined by attributes. An access structure $\mathbb{A}$ in ABE contains the authorized sets of attributes. Unless otherwise stated, by an access structure we mean a monotone access structure for the rest of this paper.

Definition 2.4 (Linear secret sharing scheme). *A linear secret sharing scheme* Π *over a set of parties* $\mathcal{P}$ *consists of an index map* ρ *and a share generating matrix* $\mathsf{L} \in \mathbb{Z}_q^{\ell \times (1+\theta)}$ *with* ℓ *rows and* θ *columns.* Π *specifies the number of shares* ℓ *and* θ *depends on the structure of* Π. *For* $i \in [\ell]$, *the function* ρ *maps the* i*-th row of* L *to its corresponding party in* $\mathcal{P}$ *denoted by* $\rho(i)$. *Let* $s \in \mathbb{Z}_q$ *be the secret to be shared among the parties. We construct a vector* $\mathbf{v} = (s, r_1, r_2, \ldots, r_\theta)^\top$ *where* $r_1, r_2, \ldots, r_\theta \leftarrow \mathbb{Z}_q$. *The matrix* L *maps* $\mathbf{v}$ *to a vector* $\mathsf{L} \cdot \mathbf{v} = (s_1, s_2, \ldots, s_\ell)^\top$ *which is a set of the shares of secret* s *according to* Π. *Each party* $\rho(i)$ *is assigned the share* $s_i = (\mathsf{L} \cdot \mathbf{v})_i$.

Every LSSS in the above definition enjoys the linear reconstruction property: Let Π be an LSSS for an access structure $\mathbb{A}$. Let $S \in \mathbb{A}$ be an arbitrary authorized set, and let $I \subset [\ell]$ be the corresponding index set defined as $I = \{i \in [\ell] : \rho(I) \in S\}$. We can find a reconstruction vector $\mathbf{g} = (g_1, \ldots, g_\ell)^\top \in \mathbb{Z}_q^\ell$ in polynomial time in the size of L (see, e.g., [Bei11]), such that (a) the support of $\mathbf{g}$ is a subset of I, that is, $g_i = 0$ for all $i \notin [\ell]$ and (b) if $\{s_i = (\mathsf{L} \cdot \mathbf{v})_i\}_{i \in I}$ are valid shares of a secret s according to Π, we have $\sum_{i \in I} g_i \cdot s_i = s$. This means that $\mathbf{g}^\top \mathsf{L} = (1, 0, \ldots, 0)$.

Converting monotone boolean formulas to LSSS matrices and index functions: We review the algorithm described in [LW11a, Appendix G] for converting any monotone boolean formula into its equivalent LSSS matrix. This section is a paraphrase of [LW11a, Appendix G]. We will use $(1, 0, 0, \ldots, 0)$ as the sharing vector for the LSSS matrix.

1. Represent the monotone boolean formula as an access tree, that is, a tree, where internal nodes denote $\wedge$ and $\vee$ and leaves denote the variables.
2. Initialize a global counter c.
3. Label the root node of the access tree with the vector (1) (a vector of length 1).
4. Visit each level of the tree following a top-down approach, and in the process of doing that, mark each node with a vector which is determined by the vector assigned to its parent node as follows:
   - If the parent node is $\vee$ labelled by the vector $\mathbf{v}$, label its children by $\mathbf{v}$. Keep the value of c.
   - If the parent node is $\wedge$ labelled by the vector $\mathbf{v}$, pad $\mathbf{v}$ with 0's at the end (if necessary) to make it of length c. Label one of its children with the vector $(\mathbf{v} \mid 1)$ and the other one with the vector $(0^c \mid -1)$, where "$\mid$" denotes the concatenation and 0^c denotes the zero vector of length c. Next, increment c by one.
5. If the entire tree are labelled, the labels of the leaves form the rows of the LSSS matrix. If any of these vectors are of different lengths, pad them with 0's.

Example: We consider the boolean formula $P(A, B, C, D) = A \vee (B \wedge (C \vee D))$. The access tree along with the labels (vectors) for each node is shown in Figure 2.1. From the labels of each leaf, we obtain the following LSSS matrix and index function:

$$\mathsf{L} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \text{ and } \rho(i) := \begin{cases} A & \text{if } i = 1 \\ B & \text{if } i = 2 \\ C & \text{if } i = 3 \\ D & \text{if } i = 4 \end{cases},$$

where first row corresponds to attribute A, second row corresponds to attribute B and so on. Each subset of the rows of this matrix includes the vector $(1, 0)$ in its span if and only if the corresponding attributes satisfy the formula $P(A, B, C, D) = A \vee (B \wedge (C \vee D))$. We denote the LSSS by $\mathsf{M} = (\mathsf{L}, \rho)$.

Requirement on LSSS: First, Boyen's KP-ABE requires an LSSS matrix and share to be defined in $\mathbb{Z}$ instead of $\mathbb{Z}_q$ for some prime q. Second, the LSSS matrix and reconstruction vector to be *low norm*. Fortunately, the matrix in the above construction is in $\{-1, 0, +1\}^{\ell \times (1+\theta)}$ and the reconstruction vector in the above construction is in $\{0, 1\}^{\ell}$.

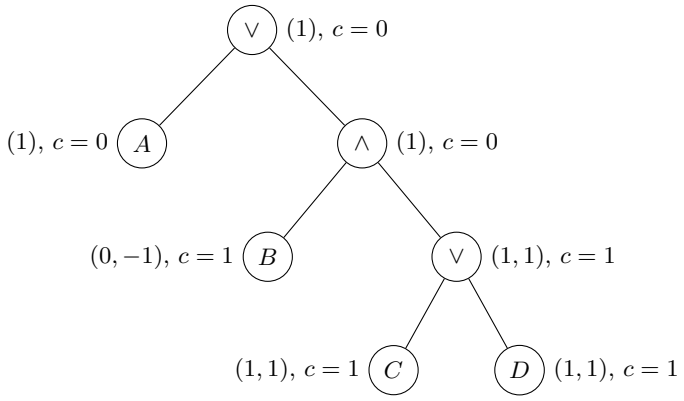

Fig. 1: Access tree for $P(A, B, C, D) = A \vee (B \wedge (C \vee D))$ with labels

3 Boyen's KP-ABE scheme

We review Boyen's KP-ABE scheme for low-norm LSSS [Boy13a]. The following scheme is a simple version of the scheme in the paper, where it is assumed that each attribute i appears exactly once on the i-th row of the LSSS matrix. Thus, ρ is always the identity function. Let ℓ be the number of attributes and we consider $U = \{1, 2, \ldots, \ell\}$.

$\mathsf{Setup}(1^\lambda, 1^\ell)$**:** Given as input the security parameter 1^λ and the length of attributes 1^ℓ, do the following:

1. Generate $(\mathbf{A}_i, \mathbf{B}_i) \leftarrow \mathsf{TrapGen}(1^\lambda)$ for $i \in [\ell]$. (Let τ_0 be the quality of $\mathbf{B}_i$.)
2. Generate $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$ and $\mathbf{u} \leftarrow \mathbb{Z}_q^n$.
3. Output

$$\mathsf{Pub} = \left(\{\mathbf{A}_i\}_{i \in [\ell]}, \mathbf{A}_0, \mathbf{u}\right) \text{ and } \mathsf{Msk} = \{\mathbf{B}_i\}_{i \in [\ell]}.$$

$\mathsf{Extract}(\mathsf{Pub}, \mathsf{Msk}, \mathsf{Policy})$**:** Upon input the master public key Pub, the master secret key Msk and the policy Policy, do the following:

1. Convert policy Policy into a (low-norm, and preferably deterministic) LSSS $\mathsf{L} \in \mathbb{Z}^{\ell \times (1+\theta)}$ assigning the i-th row of L to the attribute of index $i \in [\ell]$.[5]
2. Select θ ephemeral matrices $\mathbf{Z}_i \leftarrow \mathbb{Z}_q^{n \times m}$ for $i \in [\theta]$.
3. Create an ephemeral matrix $\mathbf{M} := [\mathbf{M}_{\mathrm{td}} \mid \mathbf{M}_{\mathrm{ext}}] \in \mathbb{Z}_q^{\ell n \times (\ell + 1 + \theta)m}$, where $\mathbf{M}_{\mathrm{td}} \in \mathbb{Z}_q^{\ell n \times \ell m}$ is a block-diagonal assembly of $\mathbf{A}_1, \ldots, \mathbf{A}_\ell$ and $\mathbf{M}_{\mathrm{ext}} \in \mathbb{Z}_q^{\ell n \times (1+\theta)m}$ is a "tensor" product of the LSSS matrix L and a randomization matrix consisting of $\mathbf{Z}_1, \ldots, \mathbf{Z}_\theta$ with $\mathbf{A}_0$. The structure of $\mathbf{M}$ is

[5] Boyen does not specify the conversion algorithm.

shown below:

$$\mathbf{M} = \left[\begin{array}{c|c} \begin{matrix} \boxed{\mathbf{A}_1} & & \\ & \boxed{\mathbf{A}_2} & \\ & & \ddots \\ & & \boxed{\mathbf{A}_\ell} \end{matrix} & \begin{matrix} l_{1,0}\ \boxed{\mathbf{A}_0} & l_{1,1}\ \boxed{\mathbf{Z}_1} & \dots\ l_{1,\theta}\ \boxed{\mathbf{Z}_\theta} \\ l_{2,0}\ \boxed{\mathbf{A}_0} & l_{2,1}\ \boxed{\mathbf{Z}_1} & \dots\ l_{2,\theta}\ \boxed{\mathbf{Z}_\theta} \\ \vdots & \vdots & \vdots \\ l_{\ell,0}\ \boxed{\mathbf{A}_0} & l_{\ell,1}\ \boxed{\mathbf{Z}_1} & \dots\ l_{\ell,\theta}\ \boxed{\mathbf{Z}_\theta} \end{matrix} \\ \underbrace{\qquad\qquad}_{\text{Public, constant, from Pub}} & \underbrace{\qquad}_{\text{From Pub}}\ \underbrace{\qquad\qquad\qquad}_{\text{Secret, random, ephemeral}} \end{array}\right] \bmod q \quad \in \mathbb{Z}_q^{\ell n \times (\ell+1+\theta)m}$$

4. Create a trapdoor $\mathbf{W}$ for the lattice $\Lambda_q^{\perp}(\mathbf{M})$ using the master secret key $\mathbf{B}_i$'s as follows: We have $\mathbf{M} = [\mathbf{M}_{\text{td}} \mid \mathbf{M}_{\text{ext}}]$, where $\mathbf{M}_{\text{td}} := \text{Diag}(\mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_\ell)$, and its trivial trapdoor $\mathbf{B} := \text{Diag}(\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_\ell)$. The algorithm extends this trivial trapdoor into $\mathbf{W}$ by using $\mathsf{ExtBasis}$, that is, $\mathbf{W} := \mathsf{ExtBasis}(\mathbf{M}_{\text{td}}, \mathbf{B}, \mathbf{M}_{\text{ext}})$.
5. Re-randomize $\mathbf{W}$ into a structure-less short trapdoor $\mathbf{K}$; $\mathbf{K} \leftarrow \mathsf{RndBasis}(\mathbf{W}, \tau)$, where $\tau = \tau_0 \cdot \omega(\sqrt{\log n})$.
6. Output the private key $\mathsf{Key}_{\mathsf{Policy}} = (\mathbf{K}, \mathsf{L})$.[6]

$\mathsf{Encrypt}(\mathsf{Pub}, \mathsf{Attrib}, \mathsf{Msg} \in \{0,1\})$**:** Upon input the master public key Pub, the attribute vector Attrib and the message bit Msg, do the following:

1. Construct an encryption matrix $\mathbf{F}$ as follows: Define $\mathbf{F}_0 := \mathbf{A}_0$. For $i \in [\ell]$, let $\mathbf{F}_i := \mathbf{A}_i$ if $i \in \mathsf{Attrib}$ and $\mathbf{O}$ otherwise. Let
$$\mathbf{F} := [\mathbf{F}_1 \mid \mathbf{F}_2 \mid \cdots \mid \mathbf{F}_\ell \mid \mathbf{F}_0] \in \mathbb{Z}_q^{n \times (1+\ell)m}.$$
2. Select a random vector $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ and low-norm Gaussian noises $\mathbf{e}_0 \in \mathbb{Z}$ and $\mathbf{e}_1 \in \mathbb{Z}^{(1+\ell)m}$.
3. Compute
$$c_0 := \mathbf{u}^\top \cdot \mathbf{s} + \mathbf{e}_0 + \lfloor q/2 \rceil \cdot \mathsf{Msg}$$
$$\mathbf{c}_1 := \mathbf{F}^\top \mathbf{s} + \mathbf{e}_1.$$
4. Output the ciphertext $\mathsf{Ctx}_{\mathsf{Attrib}} = (c_0, \mathbf{c}_1)$.

$\mathsf{Decrypt}(\mathsf{Pub}, \mathsf{Key}_{\mathsf{Policy}}, \mathsf{Ctx}_{\mathsf{Attrib}})$**:** Upon input the master public key Pub, the secret key $\mathsf{Key}_{\mathsf{Policy}}$ and the ciphertext $\mathsf{Ctx}_{\mathsf{Attrib}}$, do the following:

1. Find a short vector $\mathbf{g} \in \mathbb{Z}^\ell$ satisfying
$$\mathbf{g}^\top \cdot \mathsf{L} = \big[d, 0, \dots, 0\big] \text{ for small non-zero } d \in \mathbb{Z} \text{ and } \forall i \in [\ell] : (g_i = 0) \vee (i \in \mathsf{Attrib}).$$
2. Apply the linear combination $\mathbf{g}$ to the block-rows of $\mathbf{M}$ to transform $\mathbf{M}$ into a real encryption matrix $\mathbf{M}'$ that matches the encryption matrix $\mathbf{F}$ of the given ciphertext; that is, compute
$$\begin{aligned} \mathbf{M}' &:= (\mathbf{g}^\top \otimes \mathbf{I}_n) \cdot \mathbf{M} \\ &= [g_1\mathbf{A}_1 \mid g_2\mathbf{A}_2 \mid \cdots \mid g_\ell\mathbf{A}_\ell \mid d\mathbf{A}_0 \mid \mathbf{O} \dots \mathbf{O}] \in \mathbb{Z}_q^{n \times (\ell+1+\theta)m}. \end{aligned}$$

[6] We omit the two optimizations mentioned in the original scheme. One of the optimizations has a problem, we discuss it in Appendix A.

3. Let $\mathbf{M}''$ be the matrix containing only the $|\mathsf{Attrib}|+1$ non-zero block-columns of $\mathbf{M}'$. Let $\mathbf{K}''$ be the matrix obtained by removing from secret key $\mathbf{K}$ the matching rows and columns, i.e., rows and columns with the same indices as the columns removed from $\mathbf{M}'$. We have $\mathbf{M}' \cdot \mathbf{K} \equiv \mathbf{O} \pmod q$, which gives $\mathbf{M}'' \cdot \mathbf{K}'' \equiv \mathbf{O} \pmod q$, and $\mathbf{K}''$ is a basis of $\Lambda_q^\perp(\mathbf{M}'')$. [7]
4. Similarly, let $\mathbf{F}''$ be the matrix retaining the $|\mathsf{Attrib}|+1$ non-zero block-columns of $\mathbf{F}$, $\mathbf{c}_1''$ be the ciphertext vector from which only the matching components of $\mathbf{c}_1$ remain.
5. Let us define $\mathbf{G}$ as

$$\begin{aligned}\mathbf{G} &:= \mathrm{Diag}(g_1, g_2, \ldots, g_\ell, d) \otimes \mathbf{I}_m \\ &= \begin{bmatrix} g_1\mathbf{I}_m & & & \\ & \ddots & & \\ & & g_\ell \cdot \mathbf{I}_m & \\ & & & d \cdot \mathbf{I}_m \end{bmatrix} \in \mathbb{Z}^{(\ell+1)m \times (\ell+1)m}.\end{aligned}$$

   We also define $\mathbf{G}'' \in \mathbb{Z}^{(|\mathsf{Attrib}|+1)m \times (|\mathsf{Attrib}|+1)m}$ from $\mathbf{G}$ by retaining all non-zero diagonal blocks of $\mathbf{G}$. Since $\mathbf{F}'' \cdot \mathbf{G}'' \equiv \mathbf{M}'' \pmod q$ and $\mathbf{M}'' \cdot \mathbf{K}'' \equiv \mathbf{O} \pmod q$, we have $\mathbf{F}'' \cdot \left(\mathbf{G}'' \cdot \mathbf{K}''\right) \equiv \mathbf{O} \pmod q$. Thus $\mathbf{T} := \mathbf{G}'' \cdot \mathbf{K}''$ is a trapdoor for sampling short vectors in $\Lambda_q^\perp(\mathbf{F}'')$, whose norm is bounded as $\|\mathbf{T}\| \le \|\mathbf{G}''\| \cdot \|\mathbf{K}''\| \le \max\{g_1, \ldots, g_\ell, d\}\|\mathbf{K}''\|$.
6. Using $\mathsf{SamplePre}$ with trapdoor $\mathbf{T}$, find a short non-zero vector $\mathbf{f}''$ such that $\mathbf{F}'' \cdot \mathbf{f}'' \equiv \mathbf{u} \pmod q$.
7. Compute $v := c_0 - (\mathbf{f}'')^\top \cdot \mathbf{c}_1'' \bmod q$.
8. Output

$$\mathsf{Msg} := \lfloor (2/q) \cdot v \rceil \bmod 2.$$

Correctness: Recall that we assume that the LSSS matrix and reconstruction vector are low-norm and can be reconstructed in $\mathbb{Z}$. We analyze two components of the system

$\mathsf{Extract}$ in which the randomized invocation of $\mathsf{RndBasis}$ to obtain $\mathbf{K}$. We have $\|\tilde{\mathbf{K}}\| \le \tau \cdot \sqrt{(\ell+1+\theta)m}$.

$\mathsf{Decrypt}$ in which the calculation of the trapdoor $\mathbf{T}$ from $\mathbf{K}''$, multiplies the norm of $\mathbf{K}''$ by a factor $\le \max\{g_1, \ldots, g_\ell, d\}$ that only depends on the linear-sharing reconstruction vector $\mathbf{g}$, which in turn is a function of policy Policy and its input attribute vector Attrib.

[7] This part in the original description of Boyen's scheme has a problem. We discuss this in Appendix A.

In Decrypt, we find a short solution $\mathbf{f}''$ such that $\mathbf{F}'' \cdot \mathbf{f}'' \equiv \mathbf{u} \pmod q$. We compute

$$\begin{aligned} v &:= c_0 - (\mathbf{f}'')^\top \cdot \mathbf{c}_1'' \\ &\equiv \mathbf{u}^\top \cdot \mathbf{s} + e_0 + \lfloor q/2 \rceil \mathsf{Msg} - (\mathbf{f}'')^\top \cdot \left((\mathbf{F}'')^\top \mathbf{s} + \mathbf{e}_1''\right) \\ &\equiv \lfloor q/2 \rceil \mathsf{Msg} + e_0 - (\mathbf{f}'')^\top \cdot \mathbf{e}_1'' \pmod q. \end{aligned}$$

Thus, if we bound the error terms, e.g., $e_0 - (\mathbf{f}'')^\top \cdot \mathbf{e}_1'' < q/5$, we can recover Msg by computing $\lfloor (2/q)v \rceil \bmod 2$.

4 Cryptanalysis of Boyen's KP-ABE Scheme

In this section, we show Boyen's KP-ABE scheme in the previous section is insecure if *the LSSS conversion scheme is specified arbitrarily*, say, if we use the conversion scheme by Lewko and Waters [LW11a]. We demonstrate an attack based on *the basis extension technique* by Cash et al. [CHKP12], which allows an adversary to break the static security of KP-ABE. The attack allows an adversary to decrypt the ciphertext encrypted with an unauthorized set of attributes which does not satisfy the policy. We use an algorithm ExtendRight to obtain the basis of an extended matrix which will serve as the *trapdoor* for the encryption matrix $\mathbf{F}$ and allows us to decrypt the ciphertext.

Idea: Let us consider the static security model: The adversary first declares the challenge set of attributes $\mathsf{Attrib}^\dagger$. It also obtains a secret key $\mathbf{K}$ for a policy Policy of its choice, where $\mathsf{Attrib}^\dagger$ does not satisfy Policy. Let L be the corresponding LSSS matrix obtained by the conversion algorithm in Extract. The adversary carefully crafts $\mathsf{Attrib}^\dagger$ and Policy so that the spanned space by the submatrix of L obtained by taking the rows corresponding to $\mathsf{Attrib}^\dagger$ contains the zero vector. Let $\mathbf{g}$ be the transformation vector such that $\mathbf{g}^\top \mathsf{L} = \mathbf{0}$ and $(g_i = 0) \vee (i \in \mathsf{Attrib}^\dagger)$ for all $i \in [\ell]$. The adversary uses this $\mathbf{g}$ to transform the ephemeral matrix $\mathbf{M}$ obtained from LSSS of Policy to the encryption matrix $\mathbf{F}$ corresponding to $\mathsf{Attrib}^\dagger$ which can be obtained from simple linear combinations of the rows of $\mathbf{M}$. Along the way, the secret key $\mathbf{K}$ that it obtained in the key query phase also gets transformed into $\mathbf{K}'$. We use ExtendRight to obtain a new trapdoor $\mathbf{T}$ from $\mathbf{K}'$ which is the trapdoor for $\mathbf{F}$. The adversary thus decrypts the challenge ciphertext in the process.

4.1 Attack Procedure

Let the attributes in the system be $U = \{1, 2, 3\}$. To each attribute, we have a random matrix $\mathbf{A}_i \in \mathbb{Z}_q^{n \times m}$ associated with it as per the Setup algorithm.

Step 1: The adversary announces $\mathsf{Attrib}^\dagger = \{2,3\}$ as a challenge set of attributes and receives the public parameters $\mathsf{Pub} = (\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3, \mathbf{u})$. The encryption matrix will look like

$$\mathbf{F} = [\ \mathbf{O}\ \ \mathbf{A}_2\ \ \mathbf{A}_3\ \mid\ \mathbf{A}_0\ ] \in \mathbb{Z}_q^{n \times 4m},$$

where $\mathbf{A}_0 \in \mathbb{Z}_q^{n \times m}$ is a randomly chosen matrix sampled during Setup.

Step 2: The adversary submits a key-extract query for a policy $P(\mathsf{Attrib}) = 1 \wedge (2 \vee 3)$, which is unsatisfied by the target set $\mathsf{Attrib}^\dagger = \{2,3\}$ since it lacks 1. The adversary receives a private key $\mathsf{Key}_P = (\mathbf{K}, \mathsf{M})$ for the policy P which is generated as follows.

Using the Lewko-Waters conversion, $\mathsf{Extract}$ converts the policy P into an LSSS matrix L, where

$$\mathsf{L} = \begin{bmatrix} 1 & 1 \\ 0 & -1 \\ 0 & 1 \end{bmatrix}.$$

The ephemeral matrix $\mathbf{M}$ will be

$$\mathbf{M} := \begin{bmatrix} \mathbf{A}_1 & & & \mathbf{A}_0 & \mathbf{Z}_1 \\ & \mathbf{A}_2 & & & -\mathbf{Z}_1 \\ & & \mathbf{A}_3 & & \mathbf{Z}_1 \end{bmatrix} \in \mathbb{Z}_q^{3n \times 5m},$$

where $\mathbf{Z}_1 \in \mathbb{Z}_q^{n \times m}$ is a random matrix sampled in $\mathsf{Extract}$. $\mathsf{Extract}$ generates a short trapdoor for the lattice $\Lambda_q^\perp(\mathbf{M})$, randomizes it into a structure-less short trapdoor $\mathbf{K}$. Thus, $\mathbf{K}$ satisfies $\mathbf{M} \cdot \mathbf{K} \equiv \mathbf{O} \pmod q$ and it is full-rank. For ease of notation, we divide $\mathbf{K}$ into small matrices; Let

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_{1,1} & \dots & \mathbf{K}_{1,5} \\ \vdots & \ddots & \vdots \\ \mathbf{K}_{5,1} & \dots & \mathbf{K}_{5,5} \end{bmatrix},$$

where $\mathbf{K}_{i,j} \in \mathbb{Z}^{m \times m}$.

Step 3: The adversary generates a trapdoor $\mathbf{T}$ for the lattice $\Lambda_q^\perp(\mathbf{F})$ from $\mathbf{K}$. Notice that $\mathbf{g}^\top = (0,1,1)$ helps to transform L into $\mathbf{g}^\top \cdot \mathsf{L} = \mathbf{0}$ and transform $\mathbf{M}$ eventually into the "submatrix" of the encryption matrix $\mathbf{F}$; along the way it will also help the adversary to obtain a trapdoor $\mathbf{T}$ for the lattice $\Lambda_q^\perp(\mathbf{F})$ from the private key $\mathbf{K}$ using basis extension technique $\mathsf{ExtendRight}$. Let us now see how the adversary does that.

Step 3-a: Obtain the trapdoor for $[\mathbf{O}\ \mathbf{A}_2\ \mathbf{A}_3\ \mathbf{O}\ \mathbf{O}]$*:* Define

$$\begin{aligned} \mathbf{M}' := (\mathbf{g}^\top \otimes \mathbf{I}_n) \cdot \mathbf{M} &= \begin{bmatrix} \mathbf{O}\ \mathbf{I}_n\ \mathbf{I}_n \end{bmatrix} \cdot \begin{bmatrix} \mathbf{A}_1 & & & \mathbf{A}_0 & \mathbf{Z}_1 \\ & \mathbf{A}_2 & & & -\mathbf{Z}_1 \\ & & \mathbf{A}_3 & & \mathbf{Z}_1 \end{bmatrix} \\ &= [\ \mathbf{O}\ \ \mathbf{A}_2\ \ \mathbf{A}_3 \mid \mathbf{O} \mid \mathbf{O}\ ]. \end{aligned}$$

Since $\mathbf{M} \cdot \mathbf{K} \equiv \mathbf{O} \pmod q$, we have

$$\mathbf{M}' \cdot \mathbf{K} \equiv (\mathbf{g}^\top \otimes \mathbf{I}_n) \cdot \mathbf{M} \cdot \mathbf{K} \equiv \mathbf{O} \pmod q. \tag{1}$$

Let $\mathbf{M}''$ be the matrix containing only the first three block-columns of $\mathbf{M}'$, that is, $\mathbf{M}'' := \begin{bmatrix} \mathbf{O} & \mathbf{A}_2 & \mathbf{A}_3 \end{bmatrix}$.

Step 3-b: "Shorten" the trapdoor into a trapdoor for $[\mathbf{O}\ \mathbf{A}_2\ \mathbf{A}_3]$*:* We next consider the matrix obtained by removing the last two block-rows of $\mathbf{K}$; i.e., $\mathbf{K}' := \begin{bmatrix} \mathbf{K}_{1,1} & \dots & \mathbf{K}_{1,5} \\ \mathbf{K}_{2,1} & \dots & \mathbf{K}_{2,5} \\ \mathbf{K}_{3,1} & \dots & \mathbf{K}_{3,5} \end{bmatrix}$. By the construction, $\mathbf{K}'$ has rank $3m$. Thus, the adversary can take $3m$ linearly-independent columns from $\mathbf{K}'$; let us denote the columns as $\mathbf{K}'' \in \mathbb{Z}^{3m \times 3m}$. We note that $\mathbf{M}'' \cdot \mathbf{K}' \equiv \mathbf{O} \pmod q$ and, thus, $\mathbf{M}'' \cdot \mathbf{K}'' \equiv \mathbf{O} \pmod q$. Since the rank of $\mathbf{K}''$ (over $\mathbb{Z}$) is $3m$, the adversary can compute a trapdoor $\mathbf{K}'''$ of $\Lambda_q^\perp(\mathbf{M}'')$.

Step 3-c: Extend the trapdoor into a trapdoor for $\mathbf{F} = [\mathbf{O}\ \mathbf{A}_2\ \mathbf{A}_3\ \mathbf{A}_0]$*:* The adversary now use $\mathsf{ExtBasis}$ to obtain a basis $\mathbf{T}$ for the extended matrix $\mathbf{F} = [\mathbf{O}\ \mathbf{A}_2\ \mathbf{A}_3 \mid \mathbf{A}_0] = [\mathbf{M}'' \mid \mathbf{A}_0]$, that is, $\mathbf{T} := \mathsf{ExtBasis}([\mathbf{O}\ \mathbf{A}_2\ \mathbf{A}_3], \mathbf{K}''', \mathbf{A}_0)$.

Step 4: Submit two plaintexts $\mathsf{Msg}_0 = 0$ and $\mathsf{Msg}_1 = 1$ and receives the challenge ciphertext $\mathsf{Ctx}_{\mathsf{Attrib}^\dagger} = (c_0, \mathbf{c}_1)$, where

$$c_0 := \mathbf{u}^\top \cdot \mathbf{s} + \mathbf{e}_0 + \lfloor q/2 \rceil \cdot \mathsf{Msg}_b$$
$$\mathbf{c}_1 := \mathbf{F}^\top \mathbf{s} + \mathbf{e}_1.$$

The adversary can successfully decrypt the challenge ciphertext, since it has the trapdoor of $\mathbf{F}$: generate a short vector $\mathbf{f}$ satisfying $\mathbf{F} \cdot \mathbf{f} \equiv \mathbf{u} \pmod q$; compute $v := c_0 - \mathbf{f}^\top \cdot \mathbf{c}_1 \bmod q$; and output $b' := \lfloor (2/q)v \rceil \bmod 2$ as the guess of b.

4.2 Source of Insecurity

The most important problem is that the conversion from policy to LSSS is unspecified. If we take the suggested conversion scheme in Lewko and Waters [LW11a], the KP-ABE scheme is insecure as explained. The insecurity stems from the fact that, if we apply the conversion directly, then the security proof of [Boy13a, Theorem 5] is incorrect.

The following is a quotation from the extract algorithm simulated by the challenger [Boy13a, The proof of Theorem 5]:

1. As in the real scheme, derive from Policy a (low-norm) linear sharing matrix $\mathsf{L} \in \mathbb{Z}^{\ell \times (1+\theta)}$.
2. Let $\phi = \mathsf{Attrib}^\dagger$. Make L' from L, keeping only the rows of index i such that $i \in \mathsf{Attrib}^\dagger$. Make L'' from L' by dropping the leftmost column of index $j = 0$ (keeping $j = 1, \dots, \theta$).
3. W.l.o.g., suppose that $\mathsf{Attrib}^\dagger = \{i_1, i_2, \dots, i_\phi\} = \{1, 2, \dots, \phi\}$; i.e., the first ϕ attributes, from 1 to ϕ, are arbitrarily assumed to be the attacker's targets.

4. W.l.o.g., suppose that the ϕ left-most columns of L'' form a ϕ-dimensional square matrix of full rank. The columns of L from which L'' is derived can always be reordered to achieve this, since the order of its columns (other than that of index $j = 0$) is arbitrary. Notice that this step requires that the challenge $\mathsf{Attrib}^\dagger$ do not satisfy the query Policy. If it did, by definition some non-zero $[d, 0, \ldots, 0]^\top$ would be in the span of L, and thus $[0, \ldots, 0]^\top$ non-trivially in that of L''; therefore the ϕ left-most columns of L'' would not be full-rank.

The security proof relies on a statement that the shortened LSSS matrix L'' is full-rank if the challenge attribute does not satisfy the policy. However, we cannot expect it to be true for *general* LSSS. Even if the space spanned by the rows of L' does not contain a vector $(1, 0, ..., 0)$, L'' would contain $(0, 0, ..., 0)$ if rows of L' are not linearly independent.

5 Discussion on Possible Fixes

In this section, we discuss possible fixes to Boyen's scheme and their limitations.[8] In the presentation slides [Boy13b], Boyen introduced *admissible LSSS* to rescue his KP-ABE scheme:[9]

Definition 5.1 (Admissible LSSS [Boy13b]). *An LSSS* (L, ρ) *for an access structure* $\mathbb{A}$ *is said to be* admissible *if for any set of attributes* $P \notin \mathbb{A}$*, any non-trivial combination of* $\mathsf{L}[I_P]$ *equals to* $(0, \ldots, 0)$*, where* $I_P = \{i \in [\ell] : \rho(i) \in P\}$ *and* $\mathsf{L}[I_P]$ *is the submatrix whose rows are taken from* L *according to* I_P*.*

This patch ensures that the shortened LSSS matrix L'' is *full-rank* in Step 4 of the simulated key-extraction algorithm in the security proof. Therefore, if we use admissible LSSS, the construction is indeed secure. The problem is whether we can construct admissible LSSS that is expressive enough.

Fortunately, any (monotone) boolean function can be converted into admissible LSSS as follows:

1. Convert a boolean function into a *DNF*.
2. Apply the Lewko-Waters conversion.

For example, let us consider the policy $P(A, B, C) = A \wedge (B \vee C)$.

1. Applying the conversion, we obtain a new DNF $P'(A, B, C) = (A \wedge B) \vee (A \wedge C)$.
2. Applying the labeling algorithm, we obtain the labeled tree in Figure 2.

[8] As we noted in the introduction, a fix for the scheme is provided by the recent work [DKW21].

[9] Before the presentation at TCC 2013, subset of authors contacted Boyen about the problem in the proof and an attack.

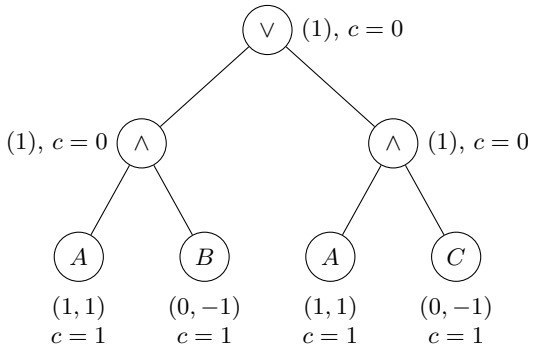

Fig. 2: Access tree for $P'(A, B, C) = (A \wedge B) \vee (A \wedge C)$ with labels

3. We obtain an LSSS $\mathsf{M} = (\mathsf{L}, \rho)$,

$$\mathsf{L} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & -1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix} \text{ and } \rho(i) = \begin{cases} A & \text{if } i = 1, 3 \\ B & \text{if } i = 2 \\ C & \text{if } i = 4 \end{cases}.$$

Unfortunately, this DNF-then-LW conversion sometimes introduces *exponential blowup* even if we restrict boolean functions to be *monotone*, which means that the scheme cannot deal with general NC^1 circuits. Miltersen, Radhakrishnan, and Wegener [MRW05] explicitly constructs a monotone function with a CNF of size $n^{O(1)}$ whose minimum size DNF has size $2^{n-\Theta(n \log\log n/\log n)}$, where the size of CNF denotes the number of clauses in a CNF and the size of DNF denotes the number of terms in a DNF.

Let $\mathsf{cnfsize}(f)$ and $\mathsf{dnfsize}(f)$ denote the minimum number of clauses in a CNF for f and the minimum number of terms in a DNF for f, respectively. Define the majority function Maj by

$$\mathsf{Maj}_k(x) := \begin{cases} 1 & \text{if } \sum_{i=1}^{k} x_i \geq k/2 \\ 0 & \text{o.w.} \end{cases}.$$

We now that $\mathsf{cnfsize}(\mathsf{Maj}_k) \leq \mathsf{dnfsize}(\mathsf{Maj}_k) = \binom{k}{\lceil k/2 \rceil}$. For our case, we only need $k = 2$; $\mathsf{Maj}_2(x_1, x_2) = x_1 \vee x_2$ and $\mathsf{cnfsize}(\mathsf{Maj}_2) = 1$ and $\mathsf{dnfsize}(\mathsf{Maj}_2) = 2$.

Theorem 5.1 ([MRW05]). *Suppose $1 \leq k \leq N$. Let the set of N variables $\{x_1, \ldots, x_N\}$ be partitioned into $\ell = \lceil N/k \rceil$ sets $S_1, \ldots, S_\ell$ with $|S_i| = k$ for $i < \ell$. The function $h_{k,N} \colon \{0,1\}^N \to \{0,1\}$ is defined as follows:*

$$h_{k,N}(x) = \bigwedge_{i=1}^{\ell} \mathsf{Maj}(x_j : j \in S_i).$$

Then, we have

$$\mathsf{cnfsize}(h_{k,N}) \leq \lceil N/k \rceil \cdot \binom{k}{\lceil k/2 \rceil} \text{ and } \mathsf{dnfsize}(h_{k,N}) \geq \binom{k}{\lceil k/2 \rceil}^{\lfloor N/k \rfloor}.$$

Let us consider $h_{2,2\ell}(\mathbf{x}) = \bigwedge_{j=1}^{\ell}(x_{2j-1} \vee x_{2j})$ for some ℓ, which is CNF and monotone. Through the Lewko-Waters conversion, we can obtain the LSSS matrix corresponding to $h_{2,2\ell}$ and we have $\mathbb{L} \in \mathbb{Z}^{2\ell \times \ell}$. However, the DNF size of $h_{2,2\ell}$ is at least $\binom{2}{1}^{2\ell/2} = 2^{\ell}$ according to the above theorem. Thus, if we apply the DNF-then-LW conversion to $h_{2,2\ell}$, then the number of rows of the obtained LSSS matrix is at least 2^{ℓ}.

References

Ajt99. Miklós Ajtai. Generating hard instances of the short basis problem. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *ICALP 99*, volume 1644 of *LNCS*, pages 1–9. Springer, Heidelberg, July 1999. 3

AP11. Joël Alwen and Chris Peikert. Generating shorter bases for hard random lattices. *Theory of Computing Systems*, 48(3):535–553, 2011. 3

Bei11. Amos Beimel. Secret-sharing schemes: A survey. In Yeow Meng Chee, Zhenbo Guo, San Ling, Fengjing Shao, Yuansheng Tang, Huaxiong Wang, and Chaoping Xing, editors, *Coding and Cryptology*, pages 11–46, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. 5

BGG+14. Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Heidelberg, May 2014. 1

Boy13a. Xavier Boyen. Attribute-based functional encryption on lattices. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 122–142. Springer, Heidelberg, March 2013. 1, 7, 12

Boy13b. Xavier Boyen. The presentation slides of 'attribute-based encryption from post-quantum lattice assumptions', 2013. Available at http://ai.stanford.edu/~xb/tcc13/slides/index.html, last visited May 26, 2020. 13

BSW07. John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy*, pages 321–334. IEEE Computer Society Press, May 2007. 2

CHKP12. David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. *Journal of Cryptology*, 25(4):601–639, October 2012. 3, 10

DKW21. Pratish Datta, Ilan Komargodski, and Brent Waters. Decentralized multi-authority abe for dnfs from lwe. In *Eurocrypt*, 2021. 2, 13

GPSW06. Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 89–98. ACM Press, October / November 2006. Available as Cryptology ePrint Archive Report 2006/309. 2, 4

GPV08. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008. 3

GVW13. Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 545–554. ACM Press, June 2013. 1

LOS+10. Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 62–91. Springer, Heidelberg, May / June 2010. 2

LW11a. Allison B. Lewko and Brent Waters. Decentralizing attribute-based encryption. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 568–588. Springer, Heidelberg, May 2011. 6, 10, 12

LW11b. Allison B. Lewko and Brent Waters. Unbounded HIBE and attribute-based encryption. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 547–567. Springer, Heidelberg, May 2011. 2

LW12. Allison B. Lewko and Brent Waters. New proof methods for attribute-based encryption: Achieving full security through selective techniques. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 180–198. Springer, Heidelberg, August 2012. 2

MP12. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012. 3

MRW05. Peter Bro Miltersen, Jaikumar Radhakrishnan, and Ingo Wegener. On converting CNF to DNF. *Theoretical Computer Science*, 347(1):325 – 335, 2005. 14

OT10. Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 191–208. Springer, Heidelberg, August 2010. 2

SW05. Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Heidelberg, May 2005. 4

Wat11. Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 53–70. Springer, Heidelberg, March 2011. 2

A Incorrect Decryption and Optimization

We comment on issues in the original description of Boyen's scheme. The first problem is with optimization for the scheme noted by him in order to make the secret keys short. If we apply the optimization to the scheme described in Sec. 3, the scheme may loose correctness in some cases. The second problem is with the decryption. Similar to the above case, we cannot guarantee correctness of the scheme if the decryption algorithm works as specified. In both cases, the problem stems from the fact that a submatrix of a full-rank matrix needs not be full-rank as well and can be fixed by small change.

On the optimization and decryption: In Step 5 of $\mathsf{Extract}$, Boyen wrote as follows:

> 5. A redundant form of the policy-based private key may be output, as, $\mathsf{Key} = (\mathbf{K}, \mathsf{L})$. However, two optimizations can be made:
> (a) If the sharing matrix L is deterministic in Policy, it may be omitted.
> (b) It is not necessary to transmit all of $\mathbf{K}$ since the decryptor will only ever need the upper-left quadrant of dimension $(\ell+1)m \times (\ell+1)m$, which we denote by $\mathbf{K}' \in \mathbb{Z}^{(\ell+1)m \times (\ell+1)m}$.
> Hence, the private key for Policy may be given in compressed form, as, $\mathsf{Key} = \mathbf{K}'$.

In Step 3 of $\mathsf{Decrypt}$, Boyen wrote as follows:

> 3. Let $\mathbf{M}''$ be the matrix containing only the $|\mathsf{Attrib}| + 1$ non-zero block-columns of $\mathbf{M}'$. Let $\mathbf{K}''$ be the matrix obtained by removing from secret key $\mathbf{K}$ the matching rows and columns, i.e., rows and columns with the same indices as the columns removed from $\mathbf{M}'$. We have $\mathbf{M}' \cdot \mathbf{K} \equiv \mathbf{O} \pmod{q}$, which gives $\mathbf{M}'' \cdot \mathbf{K}'' \equiv \mathbf{O} \pmod{q}$, and $\mathbf{K}''$ is a short trapdoor of $\Lambda_q^{\perp}(\mathbf{M}'')$.

If we apply this optimization (b), then our attack fails. On the other hand, a proper decryptor also fails to decrypt a ciphertext. In addition, even if we do not apply optimization (b), Step 3 of $\mathsf{Decrypt}$ has a problem. We explain the problem by using the example.

Example: Let us use the example $\mathsf{Policy}(1,2,3) = 1 \wedge (2 \vee 3)$ and $\mathsf{Attrib} = \{1,2\}$. In the key-extract algorithm on the query Policy, the ephemeral matrix $\mathbf{M}$ will be of the form

$$\mathbf{M} := \begin{bmatrix} \mathbf{A}_1 & & \mathbf{A}_0 & \mathbf{Z}_1 \\ & \mathbf{A}_2 & & -\mathbf{Z}_1 \\ & & \mathbf{A}_3 & \mathbf{Z}_1 \end{bmatrix} \in \mathbb{Z}_q^{3n \times 5m},$$

where $\mathbf{Z}_1$ is a random matrix chosen by $\mathsf{Extract}$. By the definition of the private key, we have a "short" basis $\mathbf{K}$ of $\Lambda_q^{\perp}(\mathbf{M})$. For ease of notation, we divide $\mathbf{K}$ into small matrices; Let

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_{1,1} & \dots & \mathbf{K}_{1,5} \\ \vdots & \ddots & \vdots \\ \mathbf{K}_{5,1} & \dots & \mathbf{K}_{5,5} \end{bmatrix},$$

where $\mathbf{K}_{i,j} \in \mathbb{Z}^{m \times m}$. If we apply the optimization (b), then the private key is $\mathbf{K}' = \begin{bmatrix} \mathbf{K}_{1,1} & \cdots & \mathbf{K}_{1,4} \\ \vdots & \ddots & \vdots \\ \mathbf{K}_{4,1} & \cdots & \mathbf{K}_{4,4} \end{bmatrix}$. We note that the rank of $\mathbf{K}'$ is *at least* $3m$.

Suppose that we encrypt a message with $\mathsf{Attrib} = \{1, 2\}$. In this case, we use $\mathbf{g} = (1, 1, 0)$ and $\mathbf{M}' = [\mathbf{A}_1 \ \mathbf{A}_2 \ \mathbf{O} \mid \mathbf{A}_0 \mid \mathbf{O}]$. Thus, we use indices $\{1, 2, 4\}$ to make $\mathbf{M}'$ and $\mathbf{K}'$;

$$\mathbf{M}'' = [\mathbf{A}_1 \ \mathbf{A}_2 \ \mathbf{A}_0]$$
$$\mathbf{K}'' = \begin{bmatrix} \mathbf{K}_{1,1} & \mathbf{K}_{1,2} & \mathbf{K}_{1,4} \\ \mathbf{K}_{2,1} & \mathbf{K}_{2,2} & \mathbf{K}_{2,4} \\ \mathbf{K}_{4,1} & \mathbf{K}_{4,2} & \mathbf{K}_{4,4} \end{bmatrix}.$$

It is easy to verify that removing third and fifth block-columns and block-rows from $\mathbf{K}$ (or from $\mathbf{K}'$) *does not* ensure the rank of $\mathbf{K}''$ is $3m$. Thus, the optimization (b) in $\mathsf{Extract}$ and Step 3 in $\mathsf{Decrypt}$ have a problem.

Patch: We can patch the optimization (b) and step 3 as follows:

On the optimization (b), the extract algorithm can take the upper submatrix $\in \mathbb{Z}^{(\ell+1)m \times (\ell+1+\theta)m}$ instead of the upper-left submatrix $\in \mathbb{Z}^{(\ell+1)m \times (\ell+1)m}$. In the above example, this optimization results in $\mathbf{K}' = \begin{bmatrix} \mathbf{K}_{1,1} & \cdots & \mathbf{K}_{1,5} \\ \vdots & \ddots & \vdots \\ \mathbf{K}_{4,1} & \cdots & \mathbf{K}_{4,5} \end{bmatrix}$, which is a submatrix consisting of upper block-rows (instead of the left-top matrix).

On decryption, the decryption algorithm first takes the block-rows corresponding to the indices, takes $(|\mathsf{Attrib}| + 1)m$ linearly-independent vectors, and converts them into the basis. In the above example, this step first computes $\mathbf{K}^* = \begin{bmatrix} \mathbf{K}_{1,1} & \cdots & \mathbf{K}_{1,5} \\ \mathbf{K}_{2,1} & \cdots & \mathbf{K}_{2,5} \\ \mathbf{K}_{4,1} & \cdots & \mathbf{K}_{4,5} \end{bmatrix}$, takes $3m$ linearly-independent vectors from $\mathbf{K}^*$, and converts them into the basis $\mathbf{K}''$ of $\Lambda_q^{\perp}(\mathbf{M}'')$.