

Attribute-based Encryption for Access Control Using Elementary Operations

Jason Crampton
*Information Security Group
Royal Holloway University of London*

Alexandre Pinto
*Information Security Group
Royal Holloway University of London and
Instituto Universitário da Maia*

Abstract—Attribute-based encryption (ABE) has attracted considerable attention in the research community in recent years. It has a number of applications such as broadcast encryption and the cryptographic enforcement of access control policies. Existing instantiations of ABE make use of access structures encoded either as trees of Shamir threshold secret-sharing schemes or monotone span programs. In both cases, the appropriate computations for these schemes are interleaved with the standard operations for cryptographic pairings. Moreover, the resulting schemes are not particularly appropriate for access control policies. In this paper, therefore, we start by examining the representation of access control policies and investigate alternative secret-sharing schemes that could be used to enforce them. We develop new ABE schemes based on the Benaloh-Leichter scheme, which employs only elementary arithmetic operations, and then extend this to arbitrary linear secret-sharing schemes. We then compare the complexity of existing schemes with our scheme based on Benaloh-Leichter.

Keywords—access control, attribute-based encryption, secret sharing

I. INTRODUCTION

A fundamental requirement in multi-user computer systems is the ability to provide different levels of access to different users. Access control is a service designed to satisfy this requirement. Typically, an access control service is policy-based. The data owner defines an authorization policy which specifies the resources for which each user has authorized access. An attempt by a user to interact with a resource is interpreted as a “request” which is evaluated by a trusted software component, often known as a “policy decision point” (PDP). The PDP takes the request and the policy as inputs and returns a decision indicating whether the request is authorized or not. The policy enforcement point is responsible for ensuring that all requests are evaluated by the PDP and for enacting the decision returned by the PDP.

It is increasingly common for data owners to outsource the storage of their data. However, the data owner may not trust the storage provider to enforce the desired policy; moreover, the data owner may be concerned that the storage provider is “honest-but-curious” and may read data that it stores. These factors have led to a resurgence of interest in the cryptographic enforcement of access control [20]–[22], although its origins go back 30 years to the work of Akl and

Taylor [1]. In this model, the storage provider will release (encrypted) data on request and the requesting entity will only be able to decrypt the data if it possesses the relevant decryption key. Thus, the enforcement of the access control policy is based on the distribution of keys to the relevant parties.

Most research on cryptographic enforcement of access control focuses on schemes that facilitate encryption and decryption of data (rather than the key management aspects of enforcement). Attribute-based encryption (ABE) is a prominent example of this type of research [8], [15]. Ciphertexts and decryption keys are associated with attribute sets and a user associated with a particular set of attributes is provided with the appropriate key. Ciphertexts are computed using a public key and the decrypter determines under what conditions the ciphertext may be decrypted.

In this setting, an access control policy is specified indirectly by identifying authorized sets of attributes. Thus a policy may be considered to be a collection of subsets of the attribute universe, which defines the authorized sets. Hence, there is a natural correspondence between attribute-based access control policies and secret-sharing schemes. The set of authorized sets may be represented as an “access tree” and the first ABE schemes encoded these parse trees using threshold gates based on the Shamir secret-sharing scheme [19]. Extensions to those schemes use monotone span programs, which are alternative (but equivalent) representations of linear secret-sharing schemes [4].

The resulting ABE schemes use cryptographic pairings, because of their nice mathematical properties, and the operations of the associated secret-sharing schemes, which in turn determine the types of access control policies that may be enforced. It is widely assumed that these schemes can be used to enforce access control policies on behalf of data owners who wish to outsource the storage of their data. We believe it is worth examining this assumption by considering how access control policies are usually specified by data owners. This leads us to the conclusion that the secret-sharing mechanisms underpinning existing schemes may not be the most appropriate, and that there may be better ways of instantiating ABE for a large class of access control policies.

The main contributions of this paper are to (i) develop a

canonical representation for access control policies specified in terms of user attributes (ii) demonstrate how such policies can be enforced cryptographically using simpler primitives than are used in existing ABE schemes (iii) provide a comparison of the costs of using our schemes and existing ABE schemes. Our basic scheme, and one that will suffice for any canonical representation, is based on the Benaloh-Leichter secret-sharing scheme [7], which uses simple arithmetic operations. We extend our construction to arbitrary linear secret-sharing schemes. We do not claim that our methods will always produce a more efficient enforcement mechanism; indeed, certain kinds of policies are ideally suited to ABE schemes based on the Shamir secret-sharing scheme. However, we do claim many access control policies do not have this form and that it is important to understand the trade-offs that exist when considering ABE as a cryptographic enforcement mechanism for a given policy.

In the next section, we describe relevant background material and related work. In Section III, we explain how the Benaloh-Leichter secret-sharing scheme can be used as the basis for an ABE scheme and extend our approach to include arbitrary linear secret-sharing schemes. In the following sections, we analyze the security of our scheme and compare the overheads associated with our scheme and those in the literature. We conclude the paper with a summary of our results and discuss opportunities for further research.

II. BACKGROUND

A. Access Control

Given a set of users U , a set of objects O and a set of actions A , an *authorization policy* specifies which triples of the form (u, o, a) , where $u \in U$, $o \in O$ and $a \in A$, are authorized. An access control mechanism will evaluate a “request” of the form (u, o, a) —representing an attempt by user u to perform action a on object o —by determining whether it is authorized by the policy. If we restrict the set of actions to read, as is customary when considering cryptographic access control, then a request may be modelled as a pair (u, o) and a policy specifies which pairs are authorized.

There are many different ways to represent an access control policy. *Access control lists*, for example, define, on a per-object basis, which users are authorized. In our setting (with a single action), an access control list for a given object is, therefore, just a list of users. Informally we might express this symbolically in the form $o \leftarrow u_1 \vee \dots \vee u_m$, to be interpreted as: any one of the users $u_1, \dots, u_m$ is authorized to read object o . Given a request (u, o) , the access control mechanism would locate the rule $o \leftarrow u_1 \vee \dots \vee u_m$ and check whether $u = u_i$ for some i . (Of course, we could regard this as m separate rules of the form $o \leftarrow u_i$ and the access control mechanism would return the same decision.)

Alternatively, we may take a user-centric view of access control, when we enumerate the objects for which a user is authorized; such a list is sometimes known as a *capability*

list. Then we have rules of the form $u \leftarrow o_1, \dots, u \leftarrow o_m$, and request (u, o) is granted if there exists a rule $u \leftarrow o_i$ with $o_i = o$.

It is increasingly common, given the more distributed and open nature of computing systems, to specify policies in terms of users’ attributes (as in XACML 3.0,¹ for example). A user may be authorized for object o if, for example, she has an email address associated with a particular domain name and is over the age of 18. In other words, the set of authorized users is defined intensionally (on the basis of user attributes), rather than extensionally (as used in access control lists), reflecting the fact that the user population for many computer systems can no longer be explicitly identified.

Information flow policies provide the earliest examples of policies defined in this way [6]. Each user and each object is associated with an attribute, a security label taken from an ordered set of labels. The set of authorized users for object o is thus implicitly defined by a rule of the form $o \leftarrow \ell$, where $\ell \in L$ is a security label: every user with security label ℓ is authorized to read o . (In fact, we also have $o \leftarrow \ell'$ for all $\ell' > \ell$.)

To conclude this section, we define an *attribute-based, object-centric* access control policy to be a set of rules of the form $o \leftarrow B_1 \vee \dots \vee B_m$, where B_i is some subset of the set of attributes Att . Every user u is associated with a set of attributes $C \subseteq Att$. Then the request (u, o) is authorized if there exists i such that $C \supseteq B_i$. We may, dually, define an *attribute-based, subject-centric* access control policy.²

B. Secret-sharing Schemes

A *secret-sharing scheme* is used to distribute sensitive information between a set of participants $\mathcal{U}$ in such a way that only particular *authorized* subsets of $\mathcal{U}$ can reconstruct the secret. Such schemes are useful if a coalition of users is required to perform some particular action. A secret-sharing scheme defines methods for computing the shares to be allocated to individual users and for reconstructing the secret from a collection of shares. The threshold secret-sharing scheme due to Shamir, for example, uses polynomial interpolation to compute the shares and reconstruct the secret [19]. In this work, we only need to consider schemes in which the secret value and the shares that are used to reconstitute the secret are taken from the same set, which we denote by $\mathcal{S}$. A collection of authorized sets $\mathcal{A} \subseteq 2^{\mathcal{U}}$ is often called an *access structure*. $\mathcal{A}$ may be defined by enumerating all authorized sets or it may be defined intensionally. The Shamir scheme is an example of the latter approach and

¹<http://docs.oasis-open.org/xacml/3.0/>

²It should be noted that our policies are more coarse-grained than those that may be specified in a language such as XACML. In particular, we are making an assumption that the attributes of users and objects are taken from the same universe, an assumption that is unlikely to hold in general, but one that is accepted in the context of cryptographic enforcement of attribute-based policies.

defines all subsets of size t , for some threshold t , to be the authorized sets.

We define a *distribution scheme* for $\mathcal{A}$ to be a triple $(D_{\mathcal{A}}, R, \mu)$, where $D_{\mathcal{A}}$ is a function that takes a secret $s \in \mathcal{S}$ and some vector of random values $\vec{r}$ chosen at random from R according to the probability distribution μ , and returns a vector of shares $\langle (v_i, s_i) : v_i \in \mathcal{U}, s_i \in \mathcal{S}, 1 \leq i \leq N \rangle$, where N depends on $\mathcal{A}$. Note that we may have $v_i = v_j$ for some $i \neq j$. In other words, one participant may receive multiple shares. Abusing notation somewhat, we will write: $D_{\mathcal{A}}(s, \vec{r}, u)$ to denote $\langle (v_i, s_i) : v_i = u \rangle$, the shares given to user u ; and $D_{\mathcal{A}}(s, \vec{r}, V)$ to denote $\langle (v_i, s_i) : v_i \in V \rangle$, the shares given to users in $V \subseteq \mathcal{U}$. Given an access structure $\mathcal{A}$ and a set of secrets $\mathcal{S}$, a distribution scheme $(D_{\mathcal{A}}, \mu)$ is a *secret-sharing scheme realizing* $\mathcal{A}$ if there exists a reconstruction function $R_{\mathcal{A}}$ such that $R_{\mathcal{A}}(V, D_{\mathcal{A}}(s, \vec{r}, V)) = s$, if $V \in \mathcal{A}$ and returns $\perp$ otherwise.³ We omit the subscript $\mathcal{A}$ when it is clear from the context.

Henceforth, we may, in the interests of brevity, write $D(s)$ and $D(s, u)$, rather than $D(s, \vec{r})$ and $D(s, u, \vec{r})$. (In other words, it is tacitly assumed that D is a randomized algorithm.)

C. Access Structures

The earliest secret-sharing schemes [9], [19] only worked for particular classes of authorized sets. A (t, n) -threshold scheme, for example, allows any set of t (or more) users, of a total of n , to reconstruct the secret. More generally, we may wish to explicitly specify the sets of users authorized to reconstruct the secret, rather than have those sets predetermined by the scheme (as is the case in the Shamir threshold scheme).

A collection of authorized sets is often called an *access structure* and is said to be *monotonic* if V' is an authorized set whenever $V \subseteq \mathcal{U}$ is an authorized set and $V' \supseteq V$. In other words, a monotonic access structure is an order filter in $(2^{\mathcal{U}}, \subseteq)$.⁴ Since $\mathcal{U}$ is a finite set, an access structure is uniquely defined by its minimal elements. Conversely, any antichain $\{V_1, \dots, V_k\}$ in $(2^{\mathcal{U}}, \subseteq)$ uniquely defines a monotonic policy (in which the set of authorized sets contains every set that is a superset of at least one V_i).⁵ We call this the *canonical representation* of an access structure. Thus, the policy enforced by the Shamir secret-sharing scheme is monotonic, and is defined by the canonical access structure comprising all subsets of $\mathcal{U}$ of size t , which we will denote by $\binom{\mathcal{U}}{t}$.

³Strictly speaking, a secret-sharing scheme must satisfy the *correctness* and *perfect privacy* criteria [4], which are properly defined in information-theoretic terms. However, for our purposes, it is sufficient to capture the intuition behind correctness and perfect privacy, which requires the existence of a reconstruction function with the properties described.

⁴Given a partially ordered set $(X, \leq)$, an *order filter* is any subset Y of X with the property that if $x \in X$, $y \in Y$ and $x > y$ then $x \in Y$.

⁵Given a partially ordered set $(X, \leq)$, $Y \subseteq X$ is an *antichain* if for all $y, y' \in Y$, $y \leq y'$ if and only if $y = y'$.

Determining whether a collection of users forms an authorized set is analogous to determining whether a logical formula holds. In particular, the propositional variables correspond to users and a valuation for a user set $V \subseteq \mathcal{U}$ on these variables is defined by the characteristic function of V . In this interpretation, each set B_i is represented as a conjunction of the attributes it contains, and the access is the disjunction of the clauses representing $B_1, \dots, B_m$. In other words, we may regard an access structure as a formula in DNF (with no negated propositional variables). We will also make use of the parse tree of such a formula, where convenient.

It will also be convenient to consider (monotone) formulas that are not in DNF and formulas that use threshold operators. Accordingly, we introduce the n -ary operator $\mathbb{X}_t$ (a superposition of the $\wedge$ and $\vee$ symbols) to denote the logical operator that returns true if any t of its n inputs are true. Thus $\mathbb{X}_1$ is equivalent to $\vee$ (the n -ary logical OR operator) and $\mathbb{X}_n$ is equivalent to the n -ary $\wedge$ operator (logical AND). We will omit t from $\mathbb{X}_t$ when it may be deduced from the context or is irrelevant.

D. Secret-sharing and Attribute-based Access Control

Suppose that we have an attribute-based object-centric policy. Then, for a given object o , there may be several rules of the form $o \leftarrow B_1, \dots, o \leftarrow B_m$. A user u is associated with a set of attributes C and is authorized to access o if there exists $B_i \subseteq C$. In other words, the authorization policy is defined using a monotonic access structure defined by the collection of all sets that are supersets of some B_i , which corresponds to the canonical access structure $\{B_1, \dots, B_m\}$ (or $\bigvee_{i=1}^m \bigwedge_{b \in B_i} b$). In this setting, the access structure is defined over user attributes, rather than users (as would be the case in a conventional secret-sharing scheme).⁶ We can deal with user-centric policies in a similar way; in this case, it is the user that is associated with an access structure, which implicitly defines those objects (defined by their respective attributes) that are accessible to the user.

Alternatively, we may represent an access control policy for o as an arbitrary logical formula. Thus, the policy $a_1 \wedge (a_2 \vee a_3)$, for example, means that any user with attribute a_1 and (at least) one of the attributes a_2 and a_3 is authorized for o . However, bearing in mind the way in which access control policies are usually represented as a list of alternative “routes to authorization”, we will generally consider policies in canonical form. Hence, the policy $a_1 \wedge (a_2 \vee a_3)$ will be written as $(a_1 \wedge a_2) \vee (a_1 \wedge a_3)$.

E. Cryptographic Access Control

Cryptographic access control can be implemented using symmetric cryptographic primitives, and is particularly ef-

⁶In the simplest case, these attributes might be identifiers: each user is assigned to exactly one attribute and each attribute is assigned to at most one user. Then a policy is just an access control list containing the identifiers of authorized users and, in this case, each authorized set is a singleton.

fective for information-flow policies [1], [2], [11]. Information flow policies are “symmetric” in the following sense: both users and objects are associated with a security label. Informally, therefore, the “attribute” ℓ authorizes both users and objects and we have $o \leftarrow \ell \rightarrow s$. Thus it makes sense to have a key $\kappa(\ell)$ associated with the security label ℓ : object o is encrypted using $\kappa(\ell)$ and decrypted using the same key. This has one important consequence that distinguishes enforcement mechanisms that use symmetric primitives: it enforces some degree of control over the encryption of messages, unlike the asymmetric case where any party may encrypt. A second interesting aspect of information flow policies is that the set of security labels is ordered and u is authorized to read o if $\lambda(u)$, the security label of u , is greater than or equal to $\lambda(o)$, the security label of o . Thus, schemes that enforce information flow policies either require each user to have multiple decryption keys (one for every label less than or equal to $\lambda(u)$) or a single key that can be used to derive the other keys [11].

We can also use asymmetric cryptographic algorithms and a secret-sharing scheme to enforce attribute-based access control policies. In this setting, we assume the existence of four algorithms: **Setup**, **Encrypt**, **KeyGen**, and **Decrypt**. **Setup** and **KeyGen** are run by a trusted party: **Setup** establishes a master public key and corresponding master private key. The latter is used by **KeyGen** to generate decryption keys for the appropriate users. **Encrypt** uses the master public key and may be performed by any party. **Decrypt** is only successful if performed using an appropriate decryption key.

When asymmetric primitives are used we may consider two basic possibilities, which have come to be known as *ciphertext-policy* and *key-policy*. The first takes an object-centric view and enforces policies specified in the form $o \leftarrow \bigvee_i B_i$ by “embedding” the policy (defined by an access structure) in the ciphertext. In this setting, an object is encrypted with respect to an access structure $\bigvee_i B_i$, while a decryption key is associated with an attribute set. Key-policy enforcement takes a user-centric view, enforcing policies of the form $u \leftarrow \bigvee_i B_i$ and embedding the policy in the key. In this setting, objects are encrypted with respect to an attribute set and decryption keys are associated with an access structure.

F. Attribute-based Encryption

Goyal *et al.* developed a method of enforcing user-centric attribute-based access control policies known as *key-policy attribute-based encryption* (KP-ABE) [15]. The basic idea is to associate each decryption key to an access structure that is represented as a tree of threshold gates.⁷ A protected object (the plaintext) is encrypted together with a set of

⁷Note that a simple threshold secret-sharing scheme is rarely likely to be appropriate. We are not aware of any attribute-based access control policies that authorize any user that has t out of n attributes.

attributes using a pairing-based (asymmetric) cryptographic algorithm, which blinds the plaintext with a random secret value r , and this value is shared according to the policy of each key. The attributes are further encoded with part of the random secret. Each leaf node corresponds to an attribute and is associated with a share of the secret; an attribute may appear in more than one leaf node. The blinding factor r can only be removed by reconstruction of the secret, and this can only be achieved if the ciphertext’s attributes satisfy the policy of the decryption key. Bethencourt *et al.* developed an analogous method of enforcing object-centric attribute-based access control policies, known as *ciphertext-policy ABE* (CP-ABE) [8].

The original instantiations of CP-ABE and KP-ABE, which use threshold gates to represent access structures, require the technique of polynomial interpolation associated with Shamir’s secret-sharing scheme. Subsequent instantiations have used monotone span programs [16, Appendix A], which can represent any linear secret-sharing scheme, of which Shamir’s scheme is a special case [4].

The first construction for an ABE scheme [18] addressed a problem in biometric authentication, where identities are represented by collections of attributes (biometric features). Due to the “noise” that is always present in biometric readings, it is customary to assume that a reading matches a biometric template if some threshold number of attributes match. Clearly, in this setting, it makes good sense to use threshold schemes and the literature mainly uses threshold gates as the basic building blocks of ABE schemes ever since [8], [15].

In Section III, we argue that techniques based on the Benaloh-Leichter secret-sharing scheme, or arbitrary linear secret-sharing schemes, may be used to enforce attribute-based access control policies. This leads to simpler representations of the access structures and correspondingly simpler cryptographic operations. The basic idea is to encode the policy in canonical form, which corresponds to a tree comprising three levels: the root node is a $\vee$ -node; each leaf node represents an attribute; and each of the remaining nodes is a $\wedge$ -node.

Attribute-based encryption makes use of pairings, which are functions defined over elliptic curves that possess certain “nice” properties. Paterson *et al.* introduced the notions of Type 1, Type 2 and Type 3 pairings [14]. The original ABE schemes used Type 1 pairings, but recent work on the discrete log problem has rendered Type 1 pairings effectively obsolete [3]. It is generally accepted that Type 3 pairings should be used, which we now define.

Definition 1. *Given three multiplicative cyclic groups of prime order p , $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, where g_1, g_2 are respectively generators for the first two groups, a function $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a Type 3 bilinear map if it has the following properties [17]:*

- for all $u \in \mathbb{G}_1, v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$ (bilinearity);
- $e(g_1, g_2) = 1_{\mathbb{G}_T}$ for all $g_1 \in \mathbb{G}_1$ if and only if $g_2 = 1_{\mathbb{G}_2}$;
- $e(g_1, g_2) = 1_{\mathbb{G}_T}$ for all $g_2 \in \mathbb{G}_2$ if and only if $g_1 = 1_{\mathbb{G}_1}$ (non-degeneracy); and
- there is no efficiently computable homomorphism between G_1 and G_2 .

We discuss relevant security notions for pairings in Section IV-B.

Throughout the paper, we write $[n]$ to denote $\{1, \dots, n\}$ and $[0, n]$ to denote $\{0, \dots, n\}$ (where n is a positive integer).

III. KP-ABE USING ELEMENTARY OPERATIONS

Existing schemes for ABE make use of secret-sharing schemes. In particular, they use access structures defined using $\mathbb{X}$ -gates or represented by monotone span programs. We believe that (a) neither representation is particularly useful for practical access control policies; (b) these approaches to secret-sharing require the use of algorithms that are more complex than required. In this section we demonstrate how KP-ABE schemes can be built using simpler algorithms and access structures that are of direct relevance to access control policies.

A. The Benaloh-Leichter Secret-sharing Scheme

Benaloh and Leichter developed a secret-sharing scheme that can be used to assign shares for any monotonic access structure [7]. In this setting, it is convenient to view an access structure as a monotone formula. Informally, given a secret s , the policy $u_1 \vee u_2$ can be enforced by giving s to each user, while the policy $u_1 \wedge u_2$ can be enforced by selecting a value s' at random and giving the values s' and $s - s'$ to u_1 and u_2 , respectively. Thus we can compute the shares for any access structure, represented as a monotone logical formula, by defining the parse tree of the formula and recursively splitting a secret, working from the root node to the leaf nodes.

Benaloh and Leichter observed that $\vee$ and $\wedge$ can also be encoded using Shamir's threshold scheme [7, §1], because $\mathbb{X}_1$ and $\mathbb{X}_2$ are logically equivalent to $\vee$ and $\wedge$, respectively. Benaloh and Leichter also noted that it is often simpler to apply the direct methods described above and that these methods are computationally simpler for $\wedge$ operators (and equivalent for $\vee$ operators). It is this observation, in the context of cryptographically enforced access control, that we will explore in the remainder of this paper.

In Section III-C, we demonstrate how the Benaloh-Leichter secret-sharing scheme may be used to implement KP-ABE. Henceforth, we will write KP-ABE_{BL} to denote this scheme and KP-ABE_{Sh} to denote a KP-ABE scheme based on Shamir's threshold scheme, adapted for Type 3 pairings instead of Type 1. We then show how our construction can be extended to arbitrary linear secret-sharing

schemes (of which Benaloh-Leichter and Shamir are special cases), which we will denote by KP-ABE_{LSSS}, without the need to make the conceptual jump to Monotone Span Programs.

B. Linear Secret-sharing Schemes

Let $\mathcal{S}$ be a finite field. Then a secret-sharing scheme is *linear* if for each $(v_i, s_i) \in D(s, \vec{r})$, $s_i = \alpha_{i,0}s + \sum_{j=1}^{\ell} \alpha_{i,j}r_j$, where the random vector $\vec{r}$ belongs to $\mathcal{S}^{\ell}$ for some ℓ determined by $\mathcal{A}$ and $\vec{\alpha}_i \in \mathcal{S}^{\ell+1}$ is fixed (and thus independent of $\vec{r}$ and s). In other words, each share is some fixed linear combination of the secret and the elements of the random string. For brevity, we will write $\vec{\alpha}_i \cdot \langle s, \vec{r} \rangle$ to denote $\alpha_{i,0}s + \sum_{j=1}^{\ell} \alpha_{i,j}r_j$. Equivalently, a secret-sharing scheme is linear if the secret can be reconstructed as a linear combination of the user shares [4]. Given $\mathcal{A}$, $D(s, \vec{r}) = \langle (v_1, s_1), \dots, (v_N, s_N) \rangle$ and $V \in \mathcal{A}$, let I_V denote the set $\{i : (v_i, s_i) \in D(s, \vec{r}), v_i \in V\}$. Then a secret-sharing scheme is linear if for every $V \in \mathcal{A}$, there exists a vector $\vec{\beta}_V \in \mathcal{S}^N$, independent of s and $\vec{r}$, such that $\vec{\beta}_{V,i} = 0$ if $i \notin I_V$ and

$$R(V, D(s, \vec{r})) = \vec{\beta}_V \cdot D(s, \vec{r}) = \sum_{i=1}^N \beta_{V,i} s_i.$$

All secret-sharing schemes considered in this paper are linear. If a secret-sharing scheme realizing $\mathcal{A}$ is *linear* then we may construct an $N \times (\ell + 1)$ matrix $\mathcal{M}_{\mathcal{A}}$ in which the i th row is $\vec{\alpha}_i$ and $D(s, \vec{r}) = \mathcal{M}_{\mathcal{A}} \cdot \langle s, \vec{r} \rangle$. In Section III-D, we show how to construct a KP-ABE scheme from an arbitrary Linear Secret Sharing Scheme (LSSS).

Example 1. Suppose, for example, that $\mathcal{A}$ is defined by the formula $(a_1 \wedge a_2) \vee (a_1 \wedge a_3)$. This defines a linear secret-sharing scheme. Using Benaloh-Leichter to share a secret s , we may select t_1 and t_2 at random and allocate t_1 and t_2 to a_1 , $s - t_1$ to a_2 and $s - t_2$ to a_3 . Alternatively, if we consider the (logically equivalent) access structure $\mathcal{A}'$ given by $a_1 \wedge (a_2 \vee a_3)$, we select t at random and allocate it to a_1 , and then assign $s - t$ to both a_2 and a_3 . Whence we have

$$\mathcal{M}_{\mathcal{A}} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{pmatrix} \quad \text{and} \quad \mathcal{M}_{\mathcal{A}'} = \begin{pmatrix} 0 & 1 \\ 1 & -1 \\ 1 & -1 \end{pmatrix}$$

For $\mathcal{A}$, a_1 receives two shares (determined by the first two rows of $\mathcal{M}_{\mathcal{A}}$), whereas for $\mathcal{A}'$, a_1 receives a single share. The set $\{a_1, a_2\}$ is an authorized set:

$$\beta_{\{a_1, a_2\}} = (1, 0, 1, 0) \quad \text{and} \quad \beta'_{\{a_1, a_2\}} = (1, 1, 0).$$

Note that for any Benaloh-Leichter scheme realizing access structure $\mathcal{A}$, $\mathcal{M}_{\mathcal{A}}$ will only contain elements from $\{0, 1, -1\}$ and $\beta_{V,i} = 1$ if and only if $i \in I_V$. Informally,

then, the linear reconstruction of a secret in the Benaloh-Leichter scheme for authorized set V simply requires the addition of all shares associated with V . This is something we exploit in the next section.

C. KP-ABE_{BL}

We assume our access trees are in canonical form. The equivalent logical formula is, therefore, in DNF and its parse tree comprises a single root $\vee$ -node, m additional non-leaf $\wedge$ -nodes, and the i th $\wedge$ -node has $|B_i|$ children, which are leaf nodes corresponding to the attributes in B_i . As in standard KP-ABE, we share the master secret t between the authorized sets. Applying the Benaloh-Leichter scheme, this means each of the $\wedge$ -nodes receives t and the leaf nodes of the i th node each receive a share of t such that the sum of the shares is t .

Our scheme defines four algorithms: **Setup**, **Encrypt**, **KeyGen** and **Decrypt** and assumes the existence of a (public) Type 3 pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ and public parameters $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, which are generators of the respective groups. **Setup** and **KeyGen** are run by the scheme administrator, while **Encrypt** and **Decrypt** may be run by anyone. Encryption is performed using the master public key and B , a subset of Att ; decryption is performed using a key associated with an access structure $\mathcal{A}$. Decryption is successful only if $B \in \mathcal{A}$.

- The **Setup** algorithm outputs public and private key material. The private key material comprises secret values t and $\{t_a : a \in Att\}$. The public key material comprises $T = e(g_1, g_2)^t$ and $\{T_a : a \in Att\}$, where $T_a = g_2^{t_a}$.
- Given the public key material, a message M and a set of attributes $B \subseteq Att$, the **Encrypt** algorithm outputs the ciphertext

$$(B, MT^s, \{T_b^s : b \in B\})$$

where s is a secret chosen uniformly at random.

- A private (decryption) key $\kappa(\mathcal{A})$ associated with an access structure $\mathcal{A}$ is generated using the **KeyGen** algorithm, which takes the private master key and $\mathcal{A}$ as input. We share t according to $\mathcal{A}$ using the Benaloh-Leichter secret-sharing scheme. That is, for each $B \in \mathcal{A}$, we select $|B|$ values $\langle \sigma_{b,B} \rangle_{b \in B}$, such that $\sum_{b \in B} \sigma_{b,B} = t$. Then

$$\kappa(\mathcal{A}) \stackrel{\text{def}}{=} \left\{ K_{b,B} = g_1^{\sigma_{b,B}/t_b} : b \in B, B \in \mathcal{A} \right\}$$

- The **Decrypt** algorithm takes a ciphertext (B, C, D) and decryption key $\kappa(\mathcal{A})$, where $\mathcal{A} = \{B_1, \dots, B_m\}$, as inputs.

If $B_i \subseteq B$ for some i , then $T_b^s \in D$ for all $b \in B_i$. Thus, we can use $g_1^{\sigma_{b,B_i}/t_b} \in \kappa(\mathcal{A})$ and $g_2^{st_b} \in D$ to

compute

$$\begin{aligned} \prod_{b \in B_i} e(K_{b,B_i}, T_b^s) &= \prod_{b \in B_i} e(g_1^{\sigma_{b,B_i}/t_b}, g_2^{st_b}) \\ &= \prod_{b \in B_i} e(g_1, g_2)^{s\sigma_{b,B_i}} \\ &= e(g_1, g_2)^{s \sum_{b \in B_i} \sigma_{b,B_i}} \\ &= e(g_1, g_2)^{st} \end{aligned}$$

Now $C = Me(g_1, g_2)^{st}$, assuming the ciphertext is well-formed, and hence we can remove the blinding factor $e(g_1, g_2)^{st}$ to recover M .

In summary, the simplicity of the algorithm for reconstructing shares in the Benaloh-Leichter scheme means that the procedure for reconstructing the term used to the blind the plaintext (as part of the **Decrypt** algorithm) is significantly simplified. (In schemes based on the Shamir threshold scheme or monotone span programs, more complex computations, such as linear interpolation, are required.) In the next section we show how the insights acquired from KP-ABE_{BL} can be applied to arbitrary linear secret-sharing schemes.

D. KP-ABE_{LSSS}

KP-ABE_{BL} uses the fact that the Benaloh-Leichter scheme has a particularly simple algorithm for reconstructing shares when the access structure is in DNF. We stress this is unlikely to be a significant drawback in practical situations: object-centric policies are generally expressed as a set of rules, which leads naturally to a representation in canonical form, as we have seen. However, existing schemes are able to enforce policies based on arbitrary linear secret-sharing schemes [16, Appendix A], using the equivalence of *monotone span programs* and linear secret-sharing schemes [4]. We believe the use of a monotone span program (MSP) to represent a linear secret-sharing scheme (LSSS) obscures the role of secret-sharing in the construction of KP-ABE. Moreover, it is hard to conceive of an access control policy that would be expressed directly as an MSP, so it will be necessary to convert the policy (represented as an LSSS) into an MSP in order to use existing ABE schemes of this form. In this section, we demonstrate how any LSSS can be used *directly* as the basis for an ABE scheme, without the need to represent it as a MSP.

As we have seen, a linear secret-sharing scheme for $\mathcal{A} = \{B_1, \dots, B_m\}$ essentially uses a matrix $\mathcal{M}_{\mathcal{A},D}$ for the distribution algorithm and a matrix $\mathcal{M}_{\mathcal{A},R}$ for reconstruction, where the i th row of $\mathcal{M}_{\mathcal{A},R}$ is the vector β_i used to reconstruct the secret for authorized set B_i . Some linear secret-sharing schemes, like Shamir's and Benaloh-Leichter, do not require the explicit construction of these matrices. In fact, for policies with an exponential number of authorized sets (as can easily be the case with threshold policies) this

would be infeasible. However, in our generalized construction, the decrypter must be able to compute the entries of $\mathcal{M}_{\mathcal{A},R}$ for the row corresponding to the ciphertext's attribute set.⁸ Informally, key fragments will have shares encoded in their respective exponents: raising each key fragment to some appropriate exponent (determined by β_i) and then multiplying the key fragments results in a linear combination of shares in the exponent, which is then used to reconstruct the share (and subsequently unblind the message in the ciphertext).

We now describe an algorithm $R^L(\mathcal{A}, V)$ that, given access to the D, R algorithms of a secret sharing scheme for $\mathcal{A}$, constructs the entries of $\mathcal{M}_{\mathcal{A},R}$ in the row corresponding to set V : $\vec{\beta} = \langle \beta_{V,1}, \dots, \beta_{V,N} \rangle$. We also state a result asserting that this algorithm is efficient if R is, the proof of which may be found in the appendix.

Briefly, R^L picks a random secret s and distributes it among V . It then applies the reconstruction algorithm to the shares received, after decreasing the value of the first share by 1. The result is used to deduce the value of the first coefficient, and then the process is repeated until all coefficients have been found.

```

 $R^L(\mathcal{A}, V)$ 
 $s \xleftarrow{\$} \mathcal{S}$ 
 $D(s, V) \xrightarrow{\$} \langle (v_1, \sigma_1), \dots, (v_N, \sigma_N) \rangle$ 
For  $1 \leq i \leq N$ :
     $\sigma_i \leftarrow \sigma_i - 1$ 
     $s' \leftarrow R(V, \langle (v_1, \sigma_1), \dots, (v_N, \sigma_N) \rangle)$ 
     $\beta_{V,i} \leftarrow (s - s')$ 
     $\sigma_i \leftarrow \sigma_i + 1$ 
End for

```

Figure 1. The algorithm $R^L(\mathcal{A}, V)$

Claim 1. *For every linear secret-sharing scheme $(\mathcal{S}, D, R, \text{Auth}, \mathcal{U})$ for some $\mathcal{A} \in \text{Auth}$ with polynomial-time (in N) reconstruction and distribution algorithms, the time complexity of algorithms $R^L(\mathcal{A}, V)$ and $D^L(\mathcal{A})$ is also polynomial in N .*

A similar idea can be used to construct a corresponding algorithm for the distribution phase, $D^L(\mathcal{A})$, that can construct the full matrix $\mathcal{M}_{\mathcal{A},D}$. This algorithm is described in the appendix and is used in the security analysis of Section IV-C.

We now show how to modify the KP-ABE_{BL} scheme presented in the previous section to work with a given linear secret-sharing scheme $(\mathcal{S}, D, R, \text{Auth}, \mathcal{U})$ with $\mathcal{S} = \mathbb{Z}_p$ and $\mathcal{U} = \text{Att}$.

Table I provides an overview of the four algorithms used in KP-ABE_{LSSS}.

⁸In our proof of security, it will also be necessary to be able to compute the matrix $\mathcal{M}_{\mathcal{A},D}$.

- The **SetUp** and **Encrypt** algorithms are unchanged.
- In **KeyGen**, we run the distribution algorithm for the LSSS on input t , which results in a vector of shares of the form $\langle (b_i, \sigma_i) : 1 \leq i \leq N \rangle$, where $b_i \in \text{Att}$. Then

$$\kappa(\mathcal{A}) = \left\{ K_i = g_1^{\sigma_i/t_{b_i}} : 1 \leq i \leq N \right\}$$

- The **Decrypt** algorithm takes the ciphertext in the form (B, C, D) and decryption key $\kappa(\mathcal{A})$ as inputs. If $B \notin \mathcal{A}$ then output $\perp$. Otherwise, construct $\vec{\beta}_B$ using R^L . Then compute

$$\begin{aligned} \prod_{i=1}^N e(K_i, T_{b_i}^s)^{\beta_{B,i}} &= \prod_{i=1}^N e(g_1^{\sigma_i/t_{b_i}}, g_2^{st_{b_i}})^{\beta_{B,i}} \\ &= \prod_{i=1}^N e(g_1, g_2)^{s\sigma_i\beta_{B,i}} \\ &= e(g_1, g_2)^{s \sum_{i=1}^N \sigma_i\beta_{B,i}} \\ &= e(g_1, g_2)^{st} \end{aligned}$$

Finally compute $M = C/e(g_1, g_2)^{st}$.

Briefly, the idea of this construction is that a ciphertext is multiplied by a blinding factor $e(g_1, g_2)^{st}$. While $e(g_1, g_2)$ is public, the two random exponents, s and t , are not. The exponent s is different with every encryption, so that different ciphertexts are independent and can not be used to decrypt other messages. This independence makes it possible to use the same t for every ciphertext. Each time a decryption key is generated, t is shared between the attributes in the form of key fragments. Each key fragment is encoded as an element of $\mathbb{G}_1$ and is blinded by the (inverse of the) secret value of the corresponding attribute. Conversely, the ciphertext fragments encode the ciphertext exponent s also blinded by the secret value of the corresponding attribute. Ciphertext and key fragments for the same attribute are then paired, so the attribute's secret value cancels on both sides, leaving the element $e(g_1, g_2)$ raised to s multiplied by some share of t . Notice that this is possible because, although the decrypter usually can not execute the pairings for all N fragments in the key, the only fragments that are required are those with $\beta_{B,i} > 0$, which are precisely those for $i \notin B$, all of them known to the decrypter.

The products of key fragments (raised to powers determined by $\vec{\beta}_B$) allow us to form the scalar product $\vec{\beta}_B \cdot D(t)$ in the exponent of $e(g_1, g_2)$. Thus, we are able to recover the ciphertext. We use the term “lift” (and related terms) to refer to the computation of a linear combination in the exponents of a fixed base, which is usually clear from the context and so is omitted. For example, if a secret is s , the lifted secret is g_1^s in base g_1 . Notice how it's impossible to use fragments from different ciphertexts because these would have different values for s and would be incompatible.

Table I
ALGORITHMS USED IN KP-ABE_{LSSS}

Algorithm	Inputs	Outputs
Setup	Attribute set Att Public parameters $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, p, e)$	Private master key $(\{t_a : a \in Att\}, t)$ Public master key $(\{T_a : a \in Att\}, T)$, where $T_a = g_2^{t_a}$, $T = e(g_1, g_2)^t$
Encrypt	Public master key Message M Attribute set $B \subseteq Att$	Ciphertext $(B, MT^s, \{T_b^s : b \in B\})$
KeyGen	Access structure $\mathcal{A}$ Private master key	Decryption key $\left\{ K_i = g_1^{\sigma_i/t_{b_i}} : \langle (b_i, \sigma_i), \dots, (b_N, \sigma_N) \rangle \xleftarrow{\$} D_{\mathcal{A}}(t) \right\}$
Decrypt	Ciphertext (B, C, D) Decryption key $\kappa(\mathcal{A})$	Plaintext M if $B \in \mathcal{A}$ (and $\perp$ otherwise)

IV. SECURITY ANALYSIS

In this section we prove that our constructions are secure. The proof is a generalization of the proof of security for KP-ABE_{Sh} by Goyal *et al.* [16], but does not require the use of monotone span programs.

A. Secret-sharing Schemes

An information-theoretically secure secret-sharing scheme $(\mathcal{S}, D, R, \text{Auth}, \mathcal{U})$ satisfies two properties [4], [5]:

- The scheme is said to be *correct* if for all $\mathcal{A} \in \text{Auth}$, any authorized set of users $V \in \mathcal{A}$ can recover the secret by executing the reconstruction algorithm on their subset of shares.

$$\Pr[R_{\mathcal{A}}(V, D_{\mathcal{A}}(s, V)) = s] = 1$$

- The scheme is said to have *perfect privacy* if for every two secrets $s_1, s_2 \in \mathcal{S}$, for every unauthorized set T and for every possible vector of shares $\vec{w} = \langle (v_i, \sigma_i) : v_i \in T \rangle$,

$$\Pr_{\vec{r}}[D_{\mathcal{A}}(s_1, \vec{r}, T) = \vec{w}] = \Pr_{\vec{r}}[D_{\mathcal{A}}(s_2, \vec{r}, T) = \vec{w}].$$

Informally, every unauthorized set of users $T \notin \mathcal{A}$ is unable to compute anything about the secret from their shares.

B. Security Definitions and Assumptions

We adopt the “selective-set” security definition used by Goyal *et al.* [15], which is reproduced in the appendix. The original ABE schemes [8], [15] used Type 1 pairings and the proof of security relied on the assumption that the standard decision bilinear Diffie-Hellman (DBDH) is hard. However, Type 1 pairings are now regarded as either insecure or very slow (depending on the size of the security parameter), and Type 3 pairings are preferred [3]. For that reason, our constructions make use of Type 3 pairings.

The standard DBDH assumption only applies to Type 1 pairings. Generalizations of this assumption—distinguished by parameters (i, j, k) specifying the groups from which the elements in the challenge are chosen—are now used [10].

Informally, the assumption states that it is hard to distinguish the element of a target group corresponding to xyw when only given access to x, y, w as elements of two source groups encoding these values. The indices $i, j, k \in \{1, 2\}$ define what group is used to encode each of x, y, w . However, we were not able to prove the security of our scheme on the basis of any of these assumptions.⁹

We believe a natural extension of the DBDH assumption to the asymmetric setting is one that provides the adversary with all the possible information about x, y, w in the spirit of DBDH, that is, encoding each value in *both* source groups. Nevertheless, in some cases we do not need all that information, and so we formalize a generalized hardness assumption $\text{DBDH}_{i,j,k}^*$: for $\text{DBDH}_{i,j,k}^*$, the adversary receives group elements, g_i^x, g_j^y and g_k^w , where $i, j, k \in \{1, 2, *\}$ and the value $*$ indicates the adversary receives a group element from $\mathbb{G}_1$ and $\mathbb{G}_2$ for the corresponding exponent (so the adversary may receive as many as six group elements).

More formally, we fix a generator $\mathcal{G}$ that returns three groups of prime order p , a Type 3 pairing between them and generators for the first two groups, and define the distinguishability experiment shown in Figure 2, where B is an adversary, $\delta \in \{0, 1\}$ and λ is the security parameter.

- $\text{Expt}_{\mathcal{G}, B}^{\text{DBDH}_{i,j,k}^*, \delta}(1^\lambda)$:

$$I = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, p, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$$

$$x, y, w \xleftarrow{\$} \mathbb{Z}_p$$

$$Z_0 \leftarrow e(g_1, g_2)^{xyw}; Z_1 \xleftarrow{\$} \mathbb{G}_T$$

$$\iota \leftarrow \{\}$$

If $i = *$, then $\iota \leftarrow \iota \cup \{g_1^x, g_2^x\}$ else $\iota \leftarrow \iota \cup \{g_i^x\}$

If $j = *$, then $\iota \leftarrow \iota \cup \{g_1^y, g_2^y\}$ else $\iota \leftarrow \iota \cup \{g_j^y\}$

If $k = *$, then $\iota \leftarrow \iota \cup \{g_1^w, g_2^w\}$ else $\iota \leftarrow \iota \cup \{g_k^w\}$

Return $\delta' \xleftarrow{\$} B(I, \iota, Z_\delta)$

Figure 2. The experiment used to define the $\text{DBDH}_{i,j,k}^*$ assumption

⁹Like the scheme of Goyal *et al.*, our scheme can be proved to be “secure” for Type 1 pairings under the standard DBDH assumption.

Definition 2. Define the advantage of B in $\text{Expt}_{\mathcal{G},B}^{\text{DBDH}_{i,j,k}^*,\delta}(1^\lambda)$, denoted $\text{Adv}_{\mathcal{G},B}^{\text{DBDH}_{i,j,k}^*}(\lambda)$, to be

$$\left| \sum_{\delta=0}^1 (-1)^\delta \Pr[\text{Expt}_{\mathcal{G},B}^{\text{DBDH}_{i,j,k}^*,\delta}(1^\lambda) = 0] \right|.$$

Then the $\text{DBDH}_{i,j,k}^*$ assumption holds if for any PPT adversary B , $\text{Adv}_{\mathcal{G},B}^{\text{DBDH}_{i,j,k}^*}(\lambda)$ is negligible in the security parameter λ , where the probabilities are taken over all coins used in the experiment.

C. Security of $\text{KP-ABE}_{\text{LSSS}}$

We prove security under the $\text{DBDH}_{1,*}^*$ assumption. That is, the adversary receives $g_1^x, g_1^y, g_2^y, g_2^w$ in the challenge.

Theorem 1. If there is an adversary $A = (A_0, A_1, A_2)$ that can break $\text{KP-ABE}_{\text{LSSS}}$ in the selective-set model with advantage ϵ , then there is another adversary B that can win the $\text{DBDH}_{1,*}^*$ game with advantage $\epsilon' = 2\epsilon$.

Proof: We construct an adversary B that tries to win the experiment $\text{Expt}_{\mathcal{G},B}^{\text{DBDH}_{i,j,k}^*,\delta}(1^\lambda)$. It does this by simulating the experiment $\text{Expt}_{A,ABE,Att}^{\text{SetSel},\mu}(1^\lambda)$ for an adversary $A = (A_0, A_1, A_2)$. Adversary B is defined in Figure 3.

The values X, Y_1, Y_2, W received by B implicitly define variables x, y, w such that $X = g_1^x$, $Y_1 = g_1^y$, $Y_2 = g_2^y$ and $W = g_2^w$. In B 's simulation, the public key is composed of n independent random elements in $\mathbb{G}_2$, plus one value equal to $e(g_1, g_2)^{xy}$. This is exactly the same distribution of the public key in the original scheme, if we let xy take the role of t in the scheme's definition.

When B crafts the challenge for A , the fragments W^{t_i} can be rewritten as T_i^w . Thus, B uses w in the role of s in the scheme's definition. When δ is 0, $Q = (e(g_1, g_2)^{xy})^w$ plays the role of T^s . In this case, the challenge ciphertext is correctly produced according to the rules of the real game. If Q is a random element, the encryption produced in the challenge phase is just a random value that holds no information about M_μ . We denote by $\hat{t}_i$ the value $\log_{g_2} T_i$. Notice that this is implicitly defined in the Setup phase as either t_i or yt_i .

In order to correctly simulate the experiment for A , B has to answer its oracle queries adequately. By the rules of $\text{Expt}_{A,ABE,Att}^{\text{SetSel},\mu}(1^\lambda)$, A can only submit key generation queries for policies that do not authorize the set B . We describe the way B answers these queries and at the same time justify why this is a good simulation of the original game.

When B receives a key generation query for a policy $\mathcal{A}$, it must produce key fragments that embed the shares of xy for that policy. One way to do this is by computing shares to x first. Using $D^\perp$ to obtain the coefficients $\tilde{\alpha}_i$ for $\mathcal{A}$, and knowing $X = g_1^x$ and $\vec{r} = \langle r_1, \dots, r_\ell \rangle$, it would be possible

$B(I, \iota, Q)$:

Init:

$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow I$

$(X, Y_1, Y_2, W) \leftarrow \iota$

$B \leftarrow A_0$

Setup:

$R \xleftarrow{\$} e(X, Y_2)$

For $i \in \text{Att}$:

– $t_i \xleftarrow{\$} \mathbb{Z}_p$

– If $i \in B$ then $T_i \leftarrow g_2^{t_i}$ else $T_i \leftarrow Y_2^{t_i}$

$\text{PK} = (T_1, \dots, T_n, R)$

Phase 1:

$(M_0, M_1, st) \leftarrow A_1^O(\text{PK})$

Challenge:

$\mu \xleftarrow{\$} \{0, 1\}$

$E \xleftarrow{\$} (B, M_\mu \cdot Q, \{W^{t_i} : i \in B\})$

Phase 2:

$\mu' \leftarrow A_2^O(st, E)$

If $\mu' = \mu$ Return 1 else Return 0

Figure 3. Adversary B

to compute each lifted share by:

$$g_1^{\sigma_i} = X^{\alpha_{i,0}} \cdot \prod_{j=1}^{\ell} g_1^{\alpha_{i,j} \cdot r_j} = g_1^{\alpha_{i,0} \cdot x + \sum_{j=1}^{\ell} \alpha_{i,j} \cdot r_j}, \quad (1)$$

$$\langle (v_i, \sigma_i) : i \in [N] \rangle \leftarrow D(x, \vec{r}) \quad (2)$$

However, to produce a well-formed challenge ciphertext, B must produce the fragments $K_i = Y_1^{\sigma_i / \hat{t}_i}$. But (1) reveals the lifted share, not the share itself, and so B can not use that strategy to compute the answers to oracle queries. Therefore, the shares for $i \in B$ have to be directly picked by B in the simulation and the others have to be implicitly computed in a consistent way such that the shares of the attributes in B can not recover g_1^x .

B runs $D^\perp(\mathcal{A})$ to obtain $\mathcal{M}_{\mathcal{A},D}$ and write the shares as a product $\vec{\sigma} = \mathcal{M}_{\mathcal{A},D} \cdot \langle x, \vec{r} \rangle$. Since $\vec{r}$ is chosen at random and independently of x , this matrix has rank $\ell + 1$. The shares are individually random and any set of linearly independent shares has at most $\ell + 1$ elements.

Since B is an unauthorized set for $\mathcal{A}$, it must have at most $m \leq \ell$ linearly independent shares. Otherwise, the unauthorized users in B could write the linear system $\mathcal{M}_{\mathcal{A},D} \cdot \langle x, \vec{r} \rangle = \vec{\sigma}$, with $\langle x, \vec{r} \rangle$ as unknowns. This would have a unique solution and the shares would determine the secret x . B can thus choose the values of these m shares uniformly at random, knowing that they will be insufficient to recover x . However, B must now produce the values of all the remaining shares. At this point, the system $\mathcal{M}_{\mathcal{A},D} \cdot \langle x, \vec{r} \rangle = \vec{\sigma}$ has one unknown fixed (the secret x , implicitly fixed by X) and m values in the results vector $\vec{\sigma}$. This turns it into a

smaller system with m equations and ℓ unknowns, which therefore has only m independent columns.

To solve this system, B needs to determine the full vector $\vec{r}$. It picks a set of m independent columns and randomly chooses the values for the elements in $\vec{r}$ corresponding to the remaining columns. This creates a system with m unknowns and m independent equations. The solution of the system, i.e. m elements of $\vec{r}$, can then be uniquely determined, taking care to treat x symbolically, since it is not a free variable but its value is not known. In general, the solutions for r_j will be of the form $\gamma_j x + \delta_j$ for some constants γ_j, δ_j . Now, all the shares can be produced by computing

$$\begin{aligned}\vec{\sigma} &= \mathcal{M}_{A,D} \cdot \langle x, \vec{r} \rangle \\ &= \mathcal{M}_{A,D} \cdot (\langle x, \gamma_1 x, \dots, \gamma_\ell x \rangle + \langle 0, \delta_1, \dots, \delta_\ell \rangle) \\ &= x\vec{\sigma}^\gamma + \vec{\sigma}^\delta,\end{aligned}$$

where we have rewritten $\vec{\sigma}$ as $x\vec{\sigma}^\gamma + \vec{\sigma}^\delta$.

This construction guarantees that for $i \in B$, $\sigma_i^\gamma = 0$, since this value was either directly chosen or was linearly dependent on a set of m values directly chosen. Therefore, B can compute the fragments $K_i = Y_1^{\sigma_i/\hat{t}_i}$ by:

$$\begin{cases} Y_1^{\sigma_i^\delta/\hat{t}_i}, & i \in B & (\hat{t}_i = t_i) \\ X^{\sigma_i^\gamma/\hat{t}_i} \cdot g_1^{\sigma_i^\delta/\hat{t}_i}, & i \notin B & (\hat{t}_i = yt_i) \end{cases}$$

B executes in polynomial time: D^L is a polynomial algorithm and the matrix manipulations all can be done in polynomial time, which means B is a legitimate adversary.

Finally, we compute the success probability of B. When B receives a correct $e(g_1, g_2)^{xyw}$ tuple, it perfectly simulates the experiment for A. Let G_μ represent A's output in this game, where μ stands for the random challenge bit chosen by B. Then, A's advantage is $\epsilon = |\Pr[G_1 = 1] - \Pr[G_0 = 1]| = |1/2 \Pr[G_\mu = \mu] - 1/2|$. On the other hand, let H_μ represent A's output when B receives a random tuple. In this case, the challenge it gives A has no information about either of the plaintexts chosen by A and so $\Pr[H_\mu = \mu] = 1/2$.

B's advantage is linked to whether A wins its game or not, and can be written as $\epsilon' = |\Pr[G_\mu = \mu] - \Pr[H_\mu = \mu]|$. Rewriting $\Pr[G_\mu = \mu] = 1/2 \cdot (\Pr[G_1 = 1|\mu = 1] - \Pr[G_1 = 1|\mu = 0]) + 1/2$ we have $\epsilon' = 1/2 \cdot |\Pr[G_1 = 1|\mu = 1] - \Pr[G_1 = 1|\mu = 0]| = 1/2 \cdot \epsilon$.

Since $\text{DBDH}_{1,*,2}^*$ is assumed to be hard, B can only have negligible advantage, and so any adversary A trying to break ABE will also have negligible advantage. This shows that ABE is secure in the set-selective model. ■

V. CANONICAL ACCESS STRUCTURES

We now have four methods for constructing shares to be used in an ABE scheme: (i) Shamir threshold scheme, (ii) Benaloh-Leichter scheme, (iii) linear secret-sharing scheme, and (iv) monotone span programs.

These schemes can enforce exactly the same policies, although with differing costs. Beimel established the equivalence between (iii) and (iv) [4]. Any monotone access policy over a set of attributes Att , which includes those specified by linear secret-sharing schemes, can be specified by a DNF and thus be realized by the Benaloh-Leichter scheme and a tree comprising only $\vee$ - and $\wedge$ -gates. But the Benaloh-Leichter scheme itself is a linear secret-sharing scheme, which proves the equivalence between (ii) and (iii). Any structure represented by a DNF can also be represented by a threshold scheme, since threshold gates can simulate $\vee$ and $\wedge$ gates. Thus, (ii) and (i) are equivalent.

We have argued above that it is more natural to specify an access control policy as a canonical access structure rather than an arbitrary monotonic access structure or MSP. But to implement a real scheme, we have to choose a specific kind of secret-sharing scheme to instantiate the general construction. In the rest of the paper, we compare the most commonly used scheme (Shamir) with our proposal to use Benaloh-Leichter schemes, and explore which of these schemes is most suitable, in terms of key storage requirements and encryption/decryption time, for different classes of access structures. One class of access structures, for example, is the set of all threshold policies, where the authorized sets are all the t -subsets of n users, $1 \leq t \leq n$. A second class of access structures is the set of all DNF formulas.

The original ABE schemes can encode arbitrary $\mathbb{X}_t$ -gates. This means that some access structures are very easy to represent as a tree, not least those in which the set of (minimal) authorized sets is all subsets of size t (which would require n leaf nodes and a single root node). Moreover, the corresponding canonical representation using only $\wedge$ - and $\vee$ -gates would require $t \binom{n}{t}$ leaf nodes and $\binom{n}{t} + 1$ non-leaf nodes.

Why, then, would we use an ABE scheme based on the Benaloh-Leichter scheme that can only encode $\wedge$ - and $\vee$ -gates? The answer is twofold: first, it is simpler to compute the shares and reconstruct the secrets, because no polynomial interpolation is required; second, policies that are easy to encode using arbitrary threshold gates are not common in the context of access control. Attribute-based access control is usually performed in the context of a set of attribute types, each of which having an associated domain of values. A rule or policy component is usually specified in terms of one attribute value for each of the attribute types (as in XACML targets, for example). Thus, we might specify a rule that says a user with an attribute of type *role* equal to *doctor* and an attribute of type *location* equal to *internal* is authorized for a particular object. It is less likely that we will need a rule that says a user having any three role attribute values out of five is authorized for some object. And it is even more unlikely that we would define an authorization policy using a MSP. In short, reasons of computational efficiency and relevance to

practical authorization policies may make it more attractive to use an ABE scheme derived from the Benaloh-Leichter secret-sharing scheme.

To simplify the analysis and because of their practical relevance, we first consider canonical access structures. (We consider arbitrary monotonic access structures in Section VI.) The canonical access structure $\{B_1, \dots, B_m\}$ may be represented as a tree of depth 3 in which the degree of the root is m and the degree of the i th node in the next level is the cardinality of B_i .

We may assess the suitability of an ABE scheme by considering the size of ciphertexts, the size of decryption keys, the time taken to encrypt a plaintext, and the time taken to decrypt a ciphertext. The constructions for KP-ABE [15] and CP-ABE [8] generate ciphertexts and keys whose lengths are linear in the size of number of attributes and the size of the access structure. If $N = \sum_{i=1}^m |B_i|$ represents the number of leaf nodes in the access tree (equivalently, the total number of shares distributed by the secret-sharing scheme) and n the total number of attributes in the system, then the size of the ciphertext in a KP-ABE scheme is $O(n)$ while the size of the decryption key is $O(N)$. For CP-ABE, these relations are reversed. Encryption time is at most linear in the size of the ciphertext.

We now perform a more fine-grained analysis of the complexity of decryption and key generation for KP-ABE_{BL} and a Type 3 version of the KP-ABE_{Sh} scheme of Goyal et al. ([15]).

A. Decryption Key Size

The size of a decryption key is determined by the number of leaf nodes in the access tree and the pairing, and is independent of the nature of the secret-sharing scheme. The number of leaf nodes in a canonical access tree is $N = \sum_{i=1}^m |B_i|$ and the total number of nodes is at most $1 + m + N$ (with equality if the access structure contains no singleton sets).

- In KP-ABE_{BL}, key generation requires the computation of the key fragments, which first requires the computation of appropriate shares. The computation of the shares for the elements of B_i requires $|B_i| - 1$ additions. Each key fragment requires a single exponentiation. Thus, in total, key generation requires $N - m$ additions in $\mathbb{Z}_p$ and N exponentiations in $\mathbb{G}_1$.
- In KP-ABE_{Sh}, key generation again requires the computation of the appropriate shares before the computation of the key fragments. In this case, we evaluate the polynomial defined for $\wedge$ -nodes to compute the shares for the associated leaf nodes. Specifically, to compute the share for the j th element of B_i we must evaluate $q(i, j)$, where

$$q(i, x) = \sum_{\ell=0}^{|B_i|-1} a_{i,\ell} x^\ell$$

is the polynomial associated with B_i . Since for each polynomial we have to evaluate $x^2, \dots, x^{|B_i|-1}$, where $x = \{2, \dots, |B_i| - 1\}$, we can reuse the computations made for the largest B_i in the computations of the other polynomials. Letting $b = \max\{|B_1|, \dots, |B_m|\} \leq n$, this requires at most $(b - 2)^2$ multiplications in $\mathbb{Z}_p$. For each B_i , we compute the terms $a_{i,\ell} x^\ell$ for $x = 2, \dots, |B_i|$, with at most $\sum_{i=1}^m (|B_i| - 1)^2$ more multiplications in $\mathbb{Z}_p$; and then $\sum_{i=1}^m (|B_i| - 1)^2$ additions to finally compute the several $q(i, x)$. The number of multiplications required is at most $n^2 + \sum_{i=1}^m (|B_i| - 1)^2 \leq n^2 + N^2 - 2N + m$. Key fragments are computed as in KP-ABE_{BL}.

In other words, key generation in KP-ABE_{Sh} requires $O(n^2 + N^2)$ multiplication operations in $\mathbb{Z}_p$ that are not required in KP-ABE_{BL}. Thus, the difference in the cost of key generation in KP-ABE_{Sh} and KP-ABE_{BL} is determined by the cost of the multiplication operation.

B. Cost of Decryption

We now consider the cost of decryption. In KP-ABE_{BL} (and KP-ABE_{Sh}), decryption first requires the identification of a set $B \in \mathcal{A}$ such that $B \subseteq \hat{B}$, where $\hat{B}$ is the attribute set associated with the ciphertext. The differences arise when we come to reconstruct the secret from the shares.

- In KP-ABE_{BL}, we must first recover the lifted shares by computing the pairings between the key and the ciphertext fragments associated to B . Then, we compute their product, requiring $|B| - 1$ multiplications in $\mathbb{G}_T$ and $|B|$ pairings,
- In KP-ABE_{Sh}, for each share we have to compute the respective Lagrange coefficient, its pairing and combine the two. Then, we compute the product of all resulting terms.

For each share, the Lagrange coefficient is the product of $|B| - 1$ ratios of the form $j/(j - i)$. For each $j \in B$, we require $|B| - 1$ divisions and $|B| - 2$ multiplications. The difference in cost between KP-ABE_{BL} and KP-ABE_{Sh} is then $|B|$ exponentiations in $\mathbb{G}_T$, and $2|B|^2 - 3|B|$ multiplications in $\mathbb{Z}_p$.¹⁰

In other words, for canonical policies, the calculation of Shamir gates causes an increase in cost due to the computation of $O(N^2)$ multiplications in $\mathbb{Z}_p$ and $O(N)$ exponentiations in $\mathbb{G}_T$. The latter is likely to be the dominating cost.

C. Using Symmetric Primitives

Before proceeding further, we note that we can represent any attribute-based policy implemented by KP- or CP-ABE as an information flow policy and then use symmetric primitives to enforce it. A basic assumption of ABE is

¹⁰For simplicity, we ignore the cost of computing the inverses and assume the cost of division is equal to that of multiplication.

that an access control policy may be defined as a secret-sharing scheme, with attributes replacing users in such a scheme. Thus, in KP-ABE, a user is associated with an access structure $\{B_1, \dots, B_m\}$, and an object is associated with a set of attributes C .

Accordingly, we may encrypt the object M with a symmetric key $\kappa(C)$ and encrypt $\kappa(C)$ with $\kappa(D)$ for all $D \subseteq \text{Att}$ such that $D \supseteq C$. The ciphertext is defined to be

$$\left(\text{Enc}(\kappa(C), M), \{ \text{Enc}(\kappa(D), \kappa(C)) : D \supseteq C \} \right),$$

where $\text{Enc}(k, M)$ denotes the symmetric encryption of message M with key k . Finally, the user associated with access structure $\{B_1, \dots, B_m\}$ is given the set of keys $\{\kappa(B_1), \dots, \kappa(B_m)\}$. To decrypt, the user (if authorized) decrypts $\kappa(C)$ and uses this to decrypt M .

Thus, the size of the ciphertext is $O(2^n)$ in the worst case, the size of the decryption “key” is $O(m)$, and decryption is $O(1)$. Note that the size of the decryption “key” is smaller, both in relative terms (since m key components, rather than N , are required) and in absolute terms (because symmetric keys will be an order of magnitude smaller than their asymmetric counterparts). The use of symmetric primitives also means that the ciphertext expansion may not be too problematic in practice. Using KP-ABE, we must include $O(n)$ group elements in the ciphertext to assist in the reconstruction of shares by the decryption algorithm. Given the order of magnitude by which the size of symmetric and asymmetric components are likely to differ, for relatively small values of n the ciphertext expansion may be acceptable. Moreover, in standard approaches to cryptographic enforcement using symmetric primitives, the encrypted keys $\{ \text{Enc}(\kappa(D), \kappa(C)) : D \supseteq C \}$ are made available through some central point, not included with the ciphertext. Hence there is no ciphertext expansion, although storage for the public information will be required. We can also reduce the amount of public information, at the expense of increasing the amount of work required to decrypt [2], [11]. In summary, it is always worth considering symmetric primitives for cryptographic enforcement of access control policies.

VI. MONOTONE ACCESS STRUCTURES

In this section, we consider monotone access structures that are represented by logical formulas using only the $\vee$ - and $\wedge$ -operators. Such an access structure may be represented as a rooted tree whose leaves are labelled with elements of A (which may be repeated). We compare the costs of key generation and decryption in KP-ABE_{Sh} and KP-ABE_{BL}.

A. Key Generation

We have seen that the size of the decryption key is directly related to the number of leaves in a canonical

access tree. We also saw that any differences that exist between KP-ABE_{BL} and KP-ABE_{Sh} arise when considering $\wedge$ -nodes. In particular, generating the shares in KP-ABE_{BL} for a $\wedge$ -node requires $d - 1$ additions, where d is the degree (number of children) of the node; in KP-ABE_{Sh} the same node requires approximately $2d^2$ multiplications as well as $(d - 1)^2$ additions. As in Section V, we can precompute and re-use some of the values required to evaluate the polynomials associated with $\wedge$ -nodes in KP-ABE_{Sh}, requiring no more than $d_{\max}^2$ multiplications, where $d_{\max}$ is the maximum degree of any $\wedge$ -node. Summing over all the $\wedge$ -nodes in the tree, the total cost of generating the shares in KP-ABE_{Sh} requires an additional $O(n^2 + T^2)$ multiplication operations, where T denotes the total number of nodes in the tree.

B. Decryption

A secret needs to be reconstructed at every intermediate $\wedge$ -node (but not at intermediate $\vee$ -nodes). As we saw in Section V, the reconstruction for a single $\wedge$ -node in KP-ABE_{Sh} requires an additional d exponentiations and $2d^2 - 3d$ multiplications, where d is the degree of the node. For an arbitrary access structure, we may have to evaluate more than one such node (unlike the case for canonical access structures). We can reuse the ratios for the Lagrange coefficient, meaning we require no more than $O(n^2)$ multiplications for these computations, and $O(N)$ exponentiations over the entire structure.

C. Optimal Access Structures

Clearly, it is advantageous (whether we wish to use KP-ABE_{Sh} or KP-ABE_{BL}) to reduce the number of leaf nodes because this will reduce the size of the decryption key. However, one significant advantage of the canonical access structure is the ease with which decryption itself can be performed (because we simply need to identify whether the attribute set under which the ciphertext was encrypted is a superset of one of the authorized sets encoded in the decryption key). Moreover, decryption for canonical access structures only requires the evaluation of one $\wedge$ -node, unlike the case for arbitrary monotone access structures. Thus there is a trade-off between the size of the decryption key and the cost of decryption (as one might expect).

We define the *size* of an access structure to be the number of leaf nodes in the tree representation of the formula. Thus the size of a structure in canonical form is simply the sum of the cardinalities of its authorized sets. A monotonic access structure may be equivalent to two or more monotone formulas, with disjunctive normal form and conjunctive normal form being two obvious alternatives (and decryption requires the evaluation of a single $\wedge$ -node in both cases, albeit for different reasons). We define the *complexity* of a monotone access structure to be the minimum size of an equivalent logical formula. Note that the size of an access

structure’s canonical representation is not equal, in general, to its complexity (as the example $(a_1 \wedge a_2) \vee (a_1 \wedge a_3)$, which is the canonical representation of $a_1 \wedge (a_2 \vee a_3)$, demonstrates).

When considering cryptographic enforcement of access control, it is likely, as we have already argued, that we will start with a canonical access structure. Thus an interesting area for future work is to investigate the computation of an optimal access structure from a canonical one. The definition of “optimal” may well depend on the desired trade-off between the number of key fragments (which will be determined by the amount of storage available for decryption keys) and the time required for decryption. In particular, the optimal access structure may not have minimal size. A natural starting point would be to consider the conversion of DNF monotone formulas into equivalent CNF monotone formulas (and vice versa), for which some results are known [12], [13]. However, there will clearly be a role for experimental work because the relative cost of multiplication and exponentiation will have a bearing on the total cost of decryption. These are all issues that we hope to address in the near future.

VII. CONCLUSION

In this paper we have argued that if we wish to use cryptographic methods to implement access control then we should properly understand the access control policies that are likely to require enforcement. We further argued that most attribute-based access control policies will be represented as a collection of authorized sets of attributes. Most importantly, this collection is likely to require explicit enumeration.

On the other hand, existing ABE schemes are based either on trees of threshold gates or monotone span programs (MSPs), neither of which have much relevance to the access control policies that are likely to require enforcement mechanisms. We can, of course, enforce an access control policy in canonical form using KP-ABE_{Sh}, but, as we have seen, this will incur additional costs, both in terms of generating decryption keys and in decryption itself. We may also be able to translate a canonical access structure into an equivalent structure with a more efficient KP-ABE_{Sh} enforcement scheme, although we believe it is unlikely that such a structure with this property exists for every canonical structure.

Another issue with existing approaches is the requirement to perform the computations needed in threshold secret-sharing schemes or MSPs as part of the cryptographic processing, thereby increasing the computational overheads. In particular, we can see no good reason to use ABE schemes that make use of monotone span programs. First, it would be necessary to transform an access control policy into a form that was suitable for enforcement using MSPs (which implies an equivalent representation as a linear secret-sharing

scheme). Second, we have shown there exists an equivalent scheme that only requires us to use the linear secret-sharing scheme that enforces the policy.

We have proposed a more direct encoding of access control policies using the Benaloh-Leichter secret-sharing scheme. The distribution and reconstruction algorithms for this scheme are particularly simple to compute, making the related ABE schemes more efficient. This may be an important consideration for end-users with limited computational resources. We also extend our methods to use arbitrary linear secret-sharing schemes. While this may not have the same practical impact, it is of independent interest that we do not require the “machinery” of MSPs to enforce policies based on linear secret-sharing schemes. A further technical contribution is to modify existing schemes to work with Type 3 pairings, rather than Type 1 pairings, although our proof of security does require a modification to existing DBDH assumptions.

There are a number of opportunities for further research in this area. Our analysis in this paper has concentrated on key-policy ABE. We would like to derive an analogous ciphertext-policy ABE scheme and perform a comparative analysis with existing schemes. We would also like to consider the representation of access control policies in more detail, as we discussed in Section VI. We are currently implementing our schemes and hope to report some experimental results soon.

ACKNOWLEDGEMENTS

We would like to thank Steven Galbraith, Arman Khouzani, Keith Martin, Kenny Paterson, and Jacob Schuldts for insightful discussions.

APPENDIX

A. The D^L Algorithm and Proof of Claim 1

In this section, we describe D^L , the distribution algorithm analogous to the reconstruction algorithm R^L . The difference is that D^L has to produce a matrix with $\ell + 1$ coefficients for each share. The computation of the corresponding coefficients proceeds in exactly the same way as for R^L , but we successively decrease the random values r_j instead of the shares.

Proof of Claim 1: We first consider $R^L(\mathcal{A}, V)$. By assumption, the complexity of R is $O(N^k)$ for some integer k . The complexity of R^L is $O(N)$ times that of R and so its complexity is $O(N^{k+1})$.

Considering D^L , the algorithm requires the execution of D a total of $1 + N \cdot (\ell + 1)$ times. Without loss of generality, we can assume that $N > \ell$. As above, we take the complexity of D to be $O(N^k)$ for some k , and following the same reasoning we conclude that the complexity of D^L is $O(N^{k+2})$. ■

$D^L(\mathcal{A}) :$

```

 $s \xleftarrow{\$} \mathcal{S}, \vec{r} \xleftarrow{\$} \mathcal{S}^\ell$ 
 $r_0 \leftarrow s$ 
 $D(r_0, \vec{r}) \xrightarrow{\$} \langle (v_i, \sigma_i) : i \in [N] \rangle$ 
For  $i \in [N]$ :
  For  $j \in [0, \ell]$ :
     $r_j \leftarrow r_j - 1$ 
     $D(r_0, \vec{r}) \xrightarrow{\$} \langle (v_i, \sigma'_i) : i \in [N] \rangle$ 
     $\alpha_{i,j} \leftarrow \sigma_i - \sigma'_i$ 
     $r_j \leftarrow r_j + 1$ 
  End for
End for

```

Figure 4. The D^L algorithm

B. Generic Security Definition for KP-ABE

The selective-set security definition for KP-ABE is defined by the experiment in Figure 5. We define an experiment (see Figure 5) $\text{Expt}_{\mathcal{A}, \text{ABE}, \text{Att}}^{\text{SetSel}, \mu}(1^\lambda)$ for a given adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, a KP-ABE scheme $\text{ABE} = (\text{Setup}, \text{Enc}, \text{KeyGen}, \text{Dec})$ and a fixed set of attributes Att .

$\text{Expt}_{\mathcal{A}, \text{ABE}, \text{Att}}^{\text{SetSel}, \mu}(1^\lambda) :$

```

Init
 $(B, \text{st}) \xleftarrow{\$} \mathcal{A}_0$ , where  $B \subseteq \text{Att}$ 
Setup
 $(\text{PK}, \text{MK}) \leftarrow \text{Setup}(1^\lambda)$ 
Phase 1
 $(M_0, M_1, \text{st}) \xleftarrow{\$} \mathcal{A}_1^\mathcal{O}(\text{PK}, \text{st})$ 
Challenge
 $E \xleftarrow{\$} \text{Enc}(\text{PK}, B, M_\mu)$ 
Phase 2
 $\mu' \xleftarrow{\$} \mathcal{A}_2^\mathcal{O}(E, \text{st})$ 
Guess
Return  $\mu'$ 
 $\mathcal{O}(\mathcal{A}) :$ 
If  $B \in \mathcal{A}$  Then Reject
Return  $\text{KeyGen}(\mathcal{A}, \text{PK}, \text{MK})$ 

```

Figure 5. The experiment used to define the security of KP-ABE schemes [15]

The advantage of $\mathcal{A}$ when playing the SetSel game is defined to be

$$\text{Adv}_{\mathcal{A}, \text{ABE}, \text{Att}}^{\text{SetSel}}(\lambda) = \left| \sum_{\mu=0}^1 (-1)^\mu \cdot \Pr[\text{Expt}_{\mathcal{A}, \text{ABE}, \text{Att}}^{\text{SetSel}, \mu}(1^\lambda) = 0] \right|$$

Definition 3. An Attribute-Based Encryption scheme $\text{ABE} = (\text{Setup}, \text{Enc}, \text{KeyGen}, \text{Dec})$ is secure in the selective-set model of security if all polynomial adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ have negligible advantage in the SetSel game.

C. About MSPs and LSSS

We proposed a scheme for KP-ABE based on a generic linear secret-sharing scheme. However, Goyal *et al.* give a construction based on MSPs representing *any* such scheme [15]. Why then our proposal?

There is not a big difference of performance in both constructions: the only part where they differ is in the handling of the shares of the ciphertext's blinding factor. This includes both the creation of the shares, used in Key Generation, and the reconstruction of the secret exponent, used in decryption. All other steps are exactly the same, including the whole setup and encryption process.

It is useful to observe that both versions subsume the tree representation policies seen before in this paper. This is because each node in the tree represents a linear secret-sharing scheme, then the tree as a whole represents a linear scheme itself, and can be summarized only by the coefficients of the corresponding $\mathcal{M}_{\mathcal{A}, \mathcal{D}}$ matrix.

The creation of shares in our construction can obviously be done by invoking once the share distribution algorithm as a black box. In [16], the same is done by starting from a matrix of $N\ell$ entries, where ℓ is the number of random “pieces” (values in $\mathbb{Z}_p$) that are necessary for the creation of the shares, and N is the number of shares, and multiplying the matrix by the randomness vector to obtain the list of shares.

From the point of view of an implementer of an ABE scheme, these two versions may differ in two things: the amount of information that has to be stored and manipulated, and the time it takes to do that manipulation. The MSP matrix will take $N\ell[\mathbb{Z}_p]$ bits, where $[\mathbb{Z}_p]$ stands for the size of the representation of one element in $\mathbb{Z}_p$ and is at least $\log p$. In comparison, a share distribution algorithm will usually *compute* these coefficients as needed, which leads to much smaller storage requirements.

For an example, consider Shamir's LSSS. Its distribution algorithm first picks a random polynomial of degree $\ell - 1$, $P(x) = s + \sum_{i=0}^{\ell-1} \alpha_i x^i$. Then, it computes $P(j)$ for $j \in \{1, \dots, N\}$. The values in a possible MSP's matrix¹¹ are the powers j^i for $i \in \{0, \dots, \ell - 1\}$. In the share distribution algorithm, these values may be implicitly computed. All that the program needs to have is the parameters: ℓ, N, p . These can be represented simply in $O(\log N + \log \ell + \log p)$ bits. For large enough values of these parameters, the whole program will be smaller than the corresponding matrix.

Regarding the computation time, the matrix method requires time at most $O(N(\ell + 1))$. As for the black-box algorithm's complexity, we can only assume that it is efficient in practice which means it must be upper bounded by some low-degree polynomial in the number of shares.

¹¹Note that in this case it is not necessary that $s = \sum_{i=1}^{\ell} r_i$, as is required in [16]. That is an unnecessary condition, since we can express the secret s as any linear combination of the r_i values by a target vector with ℓ coefficients, that replaces the $\vec{1}$ vector of [16].

In decryption, both methods have to start by computing the reconstruction coefficients $\tilde{\beta}$. After finding these, computation proceeds in the same way, by pairing the relevant ciphertext and key fragments, raising them to the corresponding coefficient and multiplying everything to obtain the element of the target group that was used to blind the message. The only possible difference is in the computation of $\tilde{\beta}$. Goyal *et al.* do not address this issue, beyond saying that such coefficients exist. In our approach, we execute the share reconstruction algorithm at most N times to recover the relevant coefficients. Once this step finishes, both versions decrypt in exactly the same way.

In summary, since MSPs and LSSSs are equivalent, the two constructions are also equivalent and similar in computation costs. The only difference is between the evaluation of one matrix multiplication against the execution of a dedicated share distribution algorithm. However, we would argue our scheme is the more natural for application developers (as well as policy authors), who would be able to use the LSSS of their choice directly, without worrying with how to represent it in the language of monotone span programs. This has the advantage of not requiring specific code to perform matrix calculations and eliminates the need to store a (potentially large) matrix of coefficients.

REFERENCES

- [1] S. G. Akl and P. D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems*, 1(3):239–248, 1983.
- [2] M. J. Atallah, M. Blanton, N. Fazio, and K. B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Transactions on Information and System Security*, 12(3):18, 2009.
- [3] R. Barbulescu, P. Gaudry, A. Joux, and E. Thomé. A quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. *IACR Cryptology ePrint Archive*, 2013:400, 2013.
- [4] A. Beimel. *Secure Schemes for Secret Sharing and Key Distribution*. PhD thesis, Technion, 1996.
- [5] A. Beimel. Secret-sharing schemes: A survey. In Y. M. Chee, Z. Guo, S. Ling, F. Shao, Y. Tang, H. Wang, and C. Xing, editors, *IWCC*, volume 6639 of *Lecture Notes in Computer Science*, pages 11–46. Springer, 2011.
- [6] D. E. Bell and L. J. LaPadula. Secure computer systems: A mathematical model, volume II. *Journal of Computer Security*, 4(2/3):229–263, 1996.
- [7] J. C. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In S. Goldwasser, editor, *CRYPTO*, volume 403 of *Lecture Notes in Computer Science*, pages 27–35. Springer, 1988.
- [8] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334. IEEE Computer Society, 2007.
- [9] G. Blakley. Safeguarding cryptographic keys. In *AFIPS National Computer Conference*, volume 49 of *AFIPS Conference Proceedings*, pages 313–317. AFIPS Press, 1979.
- [10] L. Chen, Z. Cheng, and N. P. Smart. Identity-based key agreement protocols from pairings. *International Journal of Information Security*, 6(4):213–241, 2007.
- [11] J. Crampton, K. M. Martin, and P. R. Wild. On key assignment for hierarchical access control. In *CSFW*, pages 98–111. IEEE Computer Society, 2006.
- [12] C. Domingo, N. Mishra, and L. Pitt. Efficient read-restricted monotone CNF/DNF dualization by learning with membership queries. *Machine Learning*, 37(1):89–110, 1999.
- [13] T. Eiter, G. Gottlob, and K. Makino. New results on monotone dualization and generating hypergraph transversals. In J. H. Reif, editor, *STOC*, pages 14–22. ACM, 2002.
- [14] S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.
- [15] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In A. Juels, R. N. Wright, and S. D. C. di Vimercati, editors, *ACM Conference on Computer and Communications Security*, pages 89–98. ACM, 2006.
- [16] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. *IACR Cryptology ePrint Archive*, 2006:309, 2006.
- [17] B. Lynn. *On the Implementation of Pairing-Based Cryptosystems*. PhD thesis, Stanford University, 2007.
- [18] A. Sahai and B. Waters. Fuzzy identity-based encryption. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473. Springer, 2005.
- [19] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
- [20] S. D. C. D. Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Encryption policies for regulating access to outsourced data. *ACM Transactions on Database Systems*, 35(2):12, 2010.
- [21] G. Wang, Q. Liu, and J. Wu. Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 735–737. ACM, 2010.
- [22] S. Yu, C. Wang, K. Ren, and W. Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *INFOCOM*, pages 534–542. IEEE, 2010.