Hiding the Policy in Cryptographic Access Control

Sascha Müller and Stefan Katzenbeisser

Technische Universität Darmstadt &
Center for Advanced Security Research Darmstadt (CASED)
Security Engineering Group
{mueller,katzenbeisser}@seceng.informatik.tu-darmstadt.de

Abstract. Recently, cryptographic access control has received a lot of attention, mainly due to the availability of efficient *Attribute-Based Encryption (ABE)* schemes. ABE allows to get rid of a trusted reference monitor by enforcing access rules in a cryptographic way. However, ABE has a privacy problem: The access policies are sent in clear along with the ciphertexts. Further generalizing the idea of policy-hiding in cryptographic access control, we introduce *policy anonymity* where – similar to the well-understood concept of *k*-anonymity – the attacker can only see a large set of possible policies that might have been used to encrypt, but is not able to identify the one that was actually used. We show that using a concept from graph theory we can extend a known ABE construction to achieve the desired privacy property.

Keywords: access control, privacy, tree majors, abe, anonymity, hidden policies

1 Introduction

In the last years, new primitives like Attribute-Based Encryption (ABE) and Predicate Encryption (PE) that enable cryptographic access control have been developed in the cryptographic community. Using these ideas, access controls systems can now be constructed that do not rely on a trusted reference monitor to enforce access rules. Instead, the information is encrypted in a way that allows decryption only by parties that are eligible to decrypt them. Specifically, in Ciphertext-Policy Attribute-Based Encryption (CP-ABE), every user receives a private key that corresponds to an individual set of attributes, each attribute attesting a certain property that the user has. Each ciphertext is encrypted with a policy over these attributes in the form of a Boolean formula, and everyone whose attributes satisfy that policy can decrypt the ciphertext. The encrypted data cannot be decrypted and thus is invisible to all other users.

This approach allows to enforce access rules in many practical scenarios. For example, in the popular *Role-based Access Control (RBAC)* approach, users are assigned to *roles* and each user's roles determine which rights he has. CP-ABE can be used to efficiently enforce access rights in an RBAC scenario: For each role there is an attribute, and for each role a user possesses, he receives the corresponding attribute. Access rights are described as logical formulas over the universe of attributes. For example, if data is encrypted with a policy `RoleA AND (RoleB OR RoleC)`, every user who is active in the role `RoleA` and either `RoleB` or `RoleC` can decrypt the data.

As another example, consider a company that hosts DRM protected media files. Users can purchase licenses from various content providers that issue usage licenses containing keys required to decrypt the protected files. Let us assume that two such content providers are `contprov1` and `contprov2`. A usage license could be expressed as a Boolean formula over attributes. For example, the policy could state that the protected file should only be decrypted by someone who has purchased licenses from at least one of the given content providers and is authenticated as an adult. Policy P_2 in Figure 2 is an example of such a DRM policy. It is also possible to automatically extract such policies from policies written in the Open Digital Rights Language (ODRL) [12].

Note that in both examples, rules are enforced automatically by the cryptographic construction. Also, the access rules may be very complex allowing for elaborate, fine-grained access control if so desired by the scenario. Numerous CP-ABE schemes have been proposed with varying features that support different types of policy languages.

While CP-ABE today is well-developed and can be considered practical, there are still some desirable features missing, one of which we are concerned with in this work: In most CP-ABE constructions, the policy is sent along with the ciphertext. This appears sensible as the decryptor needs to know which of his attributes are needed to access the data. However, the policy itself might be considered worth to protect as it might reveal clues to the content of the encrypted data. For example, consider a patient report in a hospital setting that is encrypted with a policy that allows encryption only by parties with the role *neurologist* or *gerontologist*. This policy alone reveals some information about the content, i.e., the patient seems to be advanced in years and might have a neurological condition. Thus, policy privacy can be an essential feature.

1.1 Towards Policy Privacy

Currently, there are two approaches to realize policy privacy. The first and most well-understood approach is predicate encryption (PE), which can be seen as a generalization of ABE in which policies are hidden. Unfortunately, while some PE constructions today are very expressive, they are still quite limited: No particular PE instance is able to support every possible Boolean formula and PE policies are often formulated in unintuitive or inefficient ways. (We will elaborate on this important aspect later on.) This is contrary to our goal of high expressiveness and intuitive policies.

The second approach, which we are concerned with here, is to modify common CP-ABE constructions to somehow hide the policy while still allow an eligible user to decrypt. We first examine that a policy can never been *completely* hidden in a ciphertext, as it has to be stored in a finite space and a known format, so there is always a limited, finite set of possible policies that can be encoded in a particular ciphertext. This motivates to introduce the notion of *policy anonymity*, which is similar to the established notion of anonymity sets [19] and *k*-anonymity [7]: Given a number of candidates for a policy, the anonymity set, an attacker cannot determine which actual policy was used for the encryption.

Extending a CP-ABE construction to have a hiding feature has been attempted by Nishide et al. [14] and Yu et al. [24], both of which extend the CP-ABE scheme of [6], where the policy consists of a single AND-gate. Simply speaking, in these extensions the policy is still an AND-gate, but the decryptor does not now the particular configuration and has to apply all his attribute keys to decrypt. In both cases the anonymity set consists of all policies that consists of a single AND-gate over a subset of all attributes of the system.

In this paper we show that one of Nishide's CP-ABE constructions [14] can be modified in order to support the encryption with every Boolean formula by combining several AND-gates in a specific way and using a novel idea from graph theory. This in turn allows the encryptor to choose a particular anonymity set which contains – among with the original policy – many others.

The idea of the construction is as follows: Given a policy, represented by a syntax tree with $\wedge$ and $\vee$-gates, we construct a *major* of this tree, i.e., a supertree that is built by expanding nodes of the original tree into new subtrees. Such a major can be used to express many different policies by assigning different expressions to its leaves. The set of all such policies makes up the anonymity set. The decryptor knows only that the used policy is among all policies that can be encoded by the supertree. The leaves of this major are encryptions of blinded partial secrets that represent $\wedge$-gates. As these $\wedge$-gates are hidden, an adversary does not know which of the possible policies of the anonymity set is used in the encryption, but by our construction he is still able to decrypt the message if he fulfills the hidden policy. He will determine which of the leaves he is able to satisfy, obtain some of the encoded partial secrets, combine them according to the tree structure using his private key, and unblind the resulting combination to retrieve the secret. Our application of Nishide's construction takes collusion attacks into account, so no group of users (who fulfill different parts of a policy) can decrypt the policy unless one member of the group fulfills the complete policy.

Example To give an intuition of the hiding property of our system, examine Figure 1, which represents the structure of a policy anonymity set which is sent along with a ciphertext. The form of this tree is known to everyone, but the leaves are hidden using the ideas of Nishide's construction. Each leaf hides an $\wedge$-gate with an unknown con-

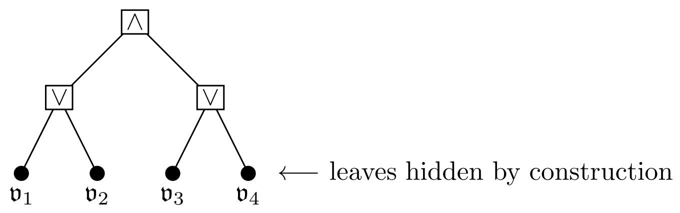


Fig. 1: Sample obfuscated policy

figuration. Each $\wedge$-gate could also represent the constant values $\perp$ (false) or $\top$ (true). Figure 2 shows some policies that might be encoded with this tree. Consider, for example, policy P_4. In our construction, each of $x_1, \dots, x_{10}$ may represent an expression of the form $A = x$ for an attribute A and an attribute value x. Here, the leaf $\mathfrak{v}_1$ could encode the expression $\mathfrak{v}_1 \equiv x_1 \wedge x_2 \wedge x_3$, $\mathfrak{v}_2$ could encode $\mathfrak{v}_2 \equiv x_4$, $\mathfrak{v}_3 \equiv x_5 \wedge x_6$, and $\mathfrak{v}_4 \equiv x_7 \wedge x_8 \wedge x_9 \wedge x_{10}$. There are various ways to encode simpler policies like $x_1 \wedge x_2$ or $x_1 \wedge (x_2 \vee x_3)$. For example, the former policy can be encoded by mapping, $\mathfrak{v}_1 \equiv \perp$,

$\mathfrak{v}_2 \equiv x_1 \wedge x_2$ $\mathfrak{v}_3 \equiv \top$, and $\mathfrak{v}_4$ to a random $\wedge$-gate, or by mapping $\mathfrak{v}_1 \equiv x_1$, $\mathfrak{v}_2 \equiv \bot$, $\mathfrak{v}_3 \equiv x_2$, and $\mathfrak{v}_4 \equiv \bot$. Several other mappings are possible. This shows that the policies encoded in a simple tree can be very complex and diverse.

P_1 : `RoleA ∧ (RoleB ∨ RoleC)`

P_2 : `(adult ∨ cc = verified) ∧`
`((contprov1.article1 = purchased ∧ account1 = balanced) ∨`
`(contprov2.article1 = purchased ∧ account2 = balanced))`

P_3 : `userrole = surgeon ∧ employer = hospitalx`

P_4 : $((x_1 \wedge x_2 \wedge x_3) \vee x_4) \wedge ((x_5 \wedge x_6) \vee (x_7 \wedge x_8 \wedge x_9 \wedge x_{10}))$

Fig. 2: Example policies for Fig. 1

An attacker cannot know the concrete semantics of the leaves, but he can determine if an attribute set satisfies the partial policy of a leaf. We will use this ability in the decryption algorithm.

1.2 Related Work

In predicate encryption schemes [10, 9, 20, 5], decryption is possible if a predicate over the user attributes and the ciphertext attributes is fulfilled. Current PE constructions are very powerful and support rather expressive predicates. Currently the most versatile solutions seem to be those that use inner product queries [10, 9]. It has been shown [9] that such a scheme can be used to construct a scheme that supports, for example, DNFs or CNFs of some bounded degree, or a predicate that can be expressed by a polynomial over the attributes. However, this predicate (for example a predicate for DNFs of some degree d) is encoded in the user keys, so it is fixed after the key generation algorithm. The complexity of the system is dependent on the size of that predicate. This means that no single PE scheme is able to express every possible policy in polynomial size and due to the bounded size of the predicate can only support a limited set of policies. Speaking in terms of anonymity, there is a fixed anonymity set that applies to all ciphertexts of an instantiation of a PE system.

In our approach, there is no fixed anonymity set. Instead, each encrypting party decides on the anonymity set when encrypting. All policies are expressed as syntax trees, so every Boolean formula can be expressed in polynomial size. As we will show in Section 4.1, the anonymity set is exponential in the size of the tree major that was used to encode the policy.

Furthermore it should be noted that predicate encryption schemes require very large groups and are only efficient for small attribute sets thereby making them unfeasible for many applications.

Aside from PE schemes, policy privacy has also been examined in the context of trust negotiation [8]. Here, large scrambled circuits are used to obfuscate the underlying policy, which is similar to our idea of using large tree majors. Trust negotiation is an interactive process whereas in this paper we are concerned with an off-line access control mechanism. Recently, Seyalioglu and Sahai [18] proposed an encryption scheme which also uses garbled circuits and hides the policy. However, in their scheme, the public key of a recipient must be used for the encryption, making it infeasible in the CP-ABE setting where the identities of the recipients are not known.

Literature on smallest common supertrees and related topics is extensive [17, 22, 16], however the constraints we are dealing with in our scenario have to our knowledge not yet been discussed. For a good, though somewhat dated, survey see [3]. There are several CP-ABE schemes [10, 13, 23, 2], but only [14], which we modify in this paper, and [24] support policy hiding.

Outline In the following section we discuss how to obfuscate policies by creating syntax tree majors. The syntax tree majors are then used in our CP-ABE system described in Section 3. Section 4 discusses various security aspects of this system. Section 5 concludes.

2 Syntax Tree Majors

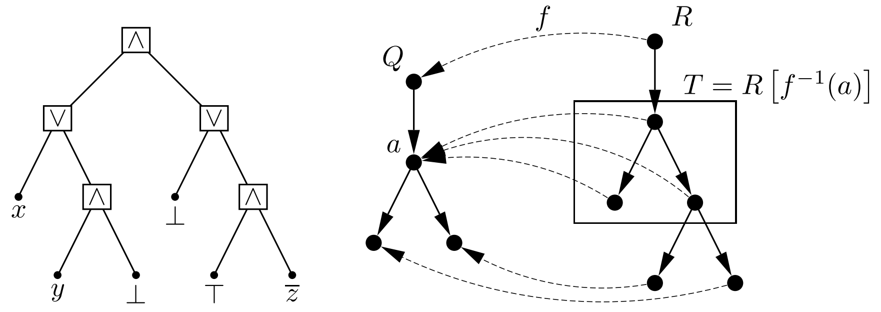
The basic idea of our system is to take a policy, encoded as a monotonic syntax tree, and find another policy that semantically contains many different policies, including the original one. An attacker is not able to decide which policy was actually used for the encryption.

Definition 1 (monotonic syntax tree). *A monotonic syntax tree T is a tree where all inner nodes are labeled with either $\wedge$ or $\vee$ and the leaves represent either Boolean variables or the constant values $\perp$ or $\top$. If the root of T is labeled $\wedge$, then every inner node of odd depth is labeled $\vee$, and every inner node of even depth is labeled $\wedge$. We call such a tree* $\wedge$-rooted*. Analogously, a* $\vee$-rooted *tree is a tree whose root is labeled $\vee$ and where every inner node of odd depth is labeled $\wedge$, and every inner node of even depth is labeled $\vee$.*

It is easy to see that any syntax tree over the operands $\wedge$ and $\vee$ can be transformed into a monotonic syntax tree by contracting adjacent $\wedge$- and $\vee$-nodes. As the labeling of all inner nodes follows from the labeling of the root node, we usually omit the labels of the inner nodes, calling the resulting tree *implicitly labeled*.

As explained in the introduction, we will use a CP-ABE scheme that encrypts the leaves, which correspond to attributes, but the construction will hide the concrete correspondence between leaves and attributes. Also note that our construction supports only monotonic syntax trees, but as there might be negative attributes (i.e., attributes that attest that the possessor does *not* have a certain property), even non-monotonic policies can be represented by monotonic syntax trees by applying DeMorgan's laws until all negations are atomic.

In order to further obfuscate the policy, we compute a larger policy such that by mapping some of its leaves to the values $\top$ and $\perp$ we are able to encode the original policy. For example, the monotonic syntax tree in Figure 3a represents the formula $x \wedge \bar{z}$. As an adversary does not know which leaves (if any) are mapped to $\top$ and $\perp$, there are many possible forms the encoded policy might have, and as the configuration of the



(a) Monotonic syntax tree (b) A mapping f of a major to a minor (leaves omitted)

Fig. 3: Examples

leaves is hidden, he is not able to access the concrete policy. We say that the larger policy *semantically contains* many smaller policies. More formally:

Definition 2 (semantic containment). *Let F and G be Boolean formulas over vectors of Boolean variables $x = (x_1, \ldots, x_n)$, resp. $y = (y_1, \ldots, y_m)$ where $m \leq n$. We call F semantically contained in G if there exists a function ϕ that maps the variables of x to either variables of y or to constant values $\top$ or $\perp$, such that $G(\psi(\phi(x))) = F(\psi(x))$ for all configuration mappings $\psi : x \mapsto \{\perp, \top\}^n$.*

We can apply this definition to syntax trees as follows: Let Q be a monotonic syntax tree with leaves $\mathbb{L}(Q) = \left\{\mathfrak{u}_1, \ldots, \mathfrak{u}_{|\mathbb{L}(Q)|}\right\}$ and R a monotonic syntax tree with leaves $\mathbb{L}(R) = \left\{\mathfrak{v}_1, \ldots, \mathfrak{v}_{|\mathbb{L}(R)|}\right\}$. We say that R *semantically contains* Q, if there is a function $\phi : \mathbb{L}(R) \to \mathbb{L}(Q) \cup \{\perp, \top\}$ such that for all configurations $\psi : \mathbb{L}(Q) \to \{\top, \perp\}$, it holds that $\psi(\phi(R)) \equiv \psi(Q)$, i.e., after applying ϕ to R, it computes the same value as Q for every possible configuration of the variables.

The type of supertree we examine is closely related to the notion of *tree majors*. Informally, a tree R is a major of a tree Q, if Q can be obtained from R by contracting a number of edges. Equally, a major of Q can be constructed by expanding some nodes into subtrees. A major can be characterized by a mapping $f : V(R) \to V(Q)$ of vertexes of a tree R to Q. We call R a *syntax tree major* of Q if we can find a mapping f with the following properties: Given a node $a \in Q$, the nodes of $f^{-1}(a)$ form a connected subtree T of R, which we denote $T = R\left[f^{-1}(a)\right]$. This is illustrated in Figure 3b. Different subtrees must not overlap and all edges of Q must be preserved in R. This is similar to the definition of a tree major.

However, in our scenario we additionally require the expanded tree to preserve the labeling of all nodes, as it needs to have the same semantics as the original tree. To understand the implications of this, let the label of a in Figure 3b be $\vee$. All other labels of Q follow from this by Definition 1. Now consider the subtree T of R. As our definition does not allow adjacent $\vee$-nodes, some nodes of T must be labeled $\wedge$. However, as both the direct predecessor of T in R and all direct successors of T in R are labeled with $\wedge$, no node of T can have the label $\wedge$. From this consideration, it follows that all subtrees introduced in a tree major of a syntax tree must have even height.

Both R_1 and R_2 of Figure 4 are examples of such majors. Note the placements of the leaves a and b. In both cases, the root of the smallest subgraph that contains both nodes has a root labeled with $\vee$. This node will take on the role that the parent of $f(a)$ and $f(b)$ in Q has (which is also an $\vee$-node).

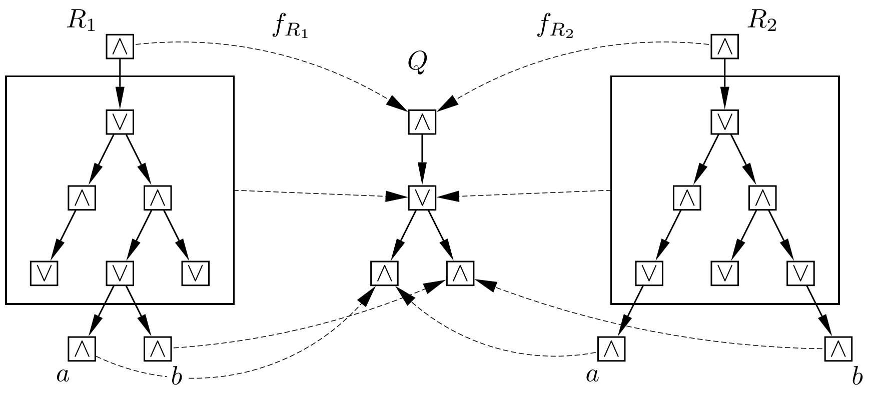


Fig. 4: Two valid syntax tree majors

Generally, all syntax tree majors must follow the rule that if two nodes a and b have a common parent in Q, then their unique common ancestor in R must have the same label as that parent. As a counter example consider the tree major R_3 (Figure 5), where the root of the smallest subtree containing a and b is an $\wedge$-node, thus not qualifying as a syntax tree major. The original graph contained a formula $f(a) \vee f(b)$, but this cannot be encoded in the given major, as a and b are only connected by an $\wedge$-node. It is now easy to see that in these cases the smallest subtrees containing a and b must have odd height.

More formally, we adapt the definition of tree minors from [15] to implicitly labeled monotonic syntax trees and define *syntax tree majors* as follows:

Definition 3 (syntax tree major). *A tree R is a syntax tree major of a tree Q if there exists a surjection $f : V(R) \to V(Q)$ such that*

1. *for each $a \in V(Q), T = R\left[f^{-1}(a)\right]$ is a connected subtree of R, and every path from the root of T to a leaf of T consists of an even number of edges;*
2. *for each pair $a, b \in V(Q)$, $f^{-1}(a) \cap f^{-1}(b) = \emptyset$;*
3. *for $S = \{(u,v) \in E(R) \mid f(u) \neq f(v)\}$, there exists a bijection $\xi : S \to E(Q)$ such that for each $e(s,t) \in S$, $\xi(e) = (f(s), f(t))$.*
4. *For each pair of edges $(x,a) \in E(Q)$ and $(x,b) \in E(Q)$, let U be the smallest subtree of R that has both a and b as leaves. Then the paths from the root of U to the roots of the subtrees $f^{-1}(a)$ and $f^{-1}(b)$ have odd length.*

We call f the *characteristic function* of the major.

If R is a syntax tree major of Q according to this definition, it semantically contains Q, and it is straightforward to configure R such that it computes the same function as Q. A proof for this statement along with complete algorithms for finding a suitable configuration of R is given in Appendix A.

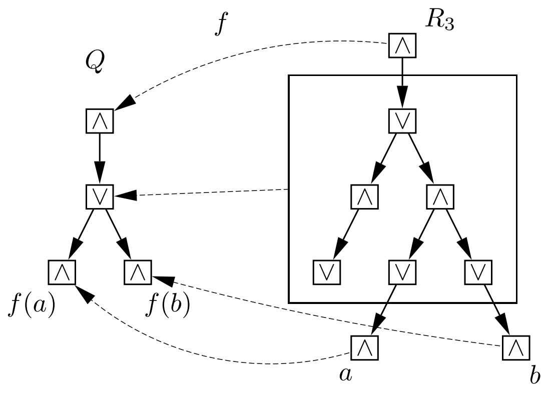


Fig. 5: An invalid syntax tree major

3 Building the System

We now describe a CP-ABE system with hidden policies, where policies are represented as syntax trees. It is based on [14], but extended to support any Boolean formula by utilizing syntax tree majors. The leaves of the syntax tree are expressions of the form $A = x$, where A identifies an attribute and x the value that this attribute must have. See Figure 2 for some example policies. Our construction supports n attributes, denoted $L_1, \dots, L_n$. Each attribute can take on one of a number of symbolic values. We denote the number of possible values of an attribute L_i by n_i and the symbolic values of the attribute by $v_{i,1}, \dots, v_{i,n_i}$. Thus, each leaf of the tree encodes an expression $L_i = v_{i,t}$. Using this approach, we are able to support every policy that can be expressed as a Boolean formula.

Note that we can also emulate numeric attributes using a bag of bits representation [2], where each number is represented by a bit string and there are two attributes for each bit. To use this, the policy would first be formulated in a more abstract form, using comparisons with numbers in the leaves like $A = x$, $A \leq x$, or $A \geq x$. These leaves would then be expanded into subtrees that evaluate the expressions using the bit representations, as outlined in [2].

3.1 Setup and KeyGen

Setup An asymmetric bilinear group $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ of order p with generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ is chosen. The trusted authority randomly selects random values for $\omega, \overline{\omega}, \beta, \overline{\beta} \in \mathbb{Z}_p^*$ and for each value $v_{i,t}$ of each attribute he also selects a random $\left\{a_{i,t} \in \mathbb{Z}_p^*\right\}_{1 \leq t \leq n_i, 1 \leq i \leq n}$. The public key PK consists of the bilinear group with generators g_1, g_2 and the values $\left\{A_{i,t} = g_1^{a_{i,t}}\right\}_{1 \leq t \leq n_i, 1 \leq i \leq n}$, as well as $Y = e(g_1, g_2)^{\omega}$, $\overline{Y} = e(g_1, g_2)^{\overline{\omega}}$, $B = g_1^{\beta}$, and $\overline{B} = g_1^{\overline{\beta}}$. The master key MK is

$$MK = \left\langle \omega, \beta, \overline{\omega}, \overline{\beta}, \{\{a_{i,1}\}_{1 \leq t \leq n_i}\}_{1 \leq i \leq n} \right\rangle .$$

Intuitively we hereby construct two parallel cryptosystems that use the same group structure and the same secret attribute keys but differ in the values of the secret key components ω and β. We will the denote the cryptosystem that uses ω and β the *primary* cryptosystem and the one that uses $\overline{\omega}$ and $\overline{\beta}$ the *secondary* cryptosystem. The primary cryptosystem will be used to encrypt the actual secret message, while the secondary one will help the decryptor to decide which nodes he can access with his attribute set. To this end, we encrypt the fixed value 1 using the secondary cryptosystem. The decryptor will try to decrypt this value from the ciphertext to see if he can satisfy the policy of the gate.

KeyGen Let $L = [L_1, L_2, \ldots, L_n] = [v_{1,t_1}, v_{2,t_2}, \ldots, v_{n,t_n}]$ be the attribute list for the user who wishes to obtain the secret key. If the user is not eligible of the requested attributes, the trusted authority returns $\perp$. Otherwise, it picks random values $s, \lambda_i \in \mathbb{Z}_p^*$ for $1 \le i \le n$, and computes $D_0 = g_2^{\beta^{-1}(\omega - s)}$ and $\overline{D_0} = g_2^{\overline{\beta}^{-1}(\overline{\omega} - s)}$. For $1 \le i \le n$, the authority also computes $[D_{i,1}, D_{i,2}] = [g_2^{s + a_{i,t_i}\lambda_i}, g_2^{\lambda_i}]$ where $L_i = v_{i,t_i}$. The secret key SK_L is $\langle D_0, \overline{D_0}, \{D_{i,1}, D_{i,2}\}_{1 \le i \le n} \rangle$.

3.2 Encryption

After the encryptor has decided on the encryption policy and constructed a monotonic syntax tree Q, he creates a syntax tree major R of Q which is used to hide the actual policy Q.

Constructing a tree major. There are three ways to construct a syntax tree major of a syntax tree Q that represents a policy: One way is to randomly expand edges of Q into trees of even height. This will result in a random major R.

Another, more interesting approach is to "mix" Q with other trees, constructing a common major that from an adversary's point of view could encode all of the input trees as well as numerous combinations of them. This is discussed in Appendix B. Note that while the resulting tree could encode all of the input trees, we will configure the leaves such that only the desired tree Q is encoded, so satisfying any of the other trees used as input to the algorithm does not allow decryption unless Q is satisfied, too.

A third approach to construct suitable syntax tree majors could be to initially decide on a large generic tree R_0 that semantically contains all possible policies that are used in a given setting. For example, an encryptor may find that all policies that he normally uses are syntax tree minors of a 3-ary tree of height 4. Then he could always set R to that tree and use it as a syntax tree major for all encryptions. Using such an approach for a policy represented by a syntax tree Q, a mapping $f : V(R) \to V(Q)$ must be found that adheres to Definition 3. If Q is indeed a minor of R, such a mapping can be found in $O(|V(R)|)$ by a brute force algorithm that tries to match vertices of Q to vertices of R starting from the leaves of both trees. In this case, after initially selecting the generic tree R_0, the process of constructing a tree major is omitted for all further encryptions, and instead the encryptor simply sets $R := R_0$.

The result of any of these the possible approaches is a syntax tree major R and a mapping $f : V(R) \to V(Q)$ that characterizes the relationship between Q and R. We will configure R such that it computes exactly the same function as Q, but keep this configuration invisible to an attacker.

Encoding the formula. After constructing a syntax tree major R with root T and mapping f, the encryptor randomly chooses an $r \in \mathbb{Z}_p^*$ and executes `EncodeSecret`(T,r) (see Algorithm 1). This algorithm encodes the secret value $r \in \mathbb{Z}_p^*$ into the tree. Beginning from the root of the tree, the algorithm recursively traverses downwards to the leaves. If a node is an $\wedge$-node, the secret r is split into partial secrets, one for each child of the node (the decryptor must satisfy all children to recover the secret), so that the sum of all partial secrets equals r. If a node is an $\vee$-node, the secret is propagated to all child nodes. The output is a labeling $m(c)$ of all nodes of $c \in R$ to partial secrets in $\mathbb{Z}_p^*$. The idea is that a decryptor needs to be able decrypt a sufficient set of partial secrets to recover the main secret r. In the next step, the encryptor chooses which of the leaves of R should be used to compute the desired formula and which ones should be set to $\perp$ or $\top$ such that R computes the same function as Q. Let $\tilde{M} : \mathbb{L}(R) \to \{\mathscr{F}, \perp, \top, \text{rand}\}$ be the result of this, i.e., if $\tilde{M}(\mathfrak{v}) = \mathscr{F}$, encode a genuine part of Q, if $\tilde{M}(\mathfrak{v}) = \perp$, encode $\perp$, $\tilde{M}(\mathfrak{v}) = \top$, encode $\top$, or else randomly choose what to compute. There are various ways to find such a mapping. We explain one such way in Appendix A (see Algorithms 2 and 3). After this process, each leaf $\mathfrak{v}$ is marked with $\tilde{M}(\mathfrak{v})$, a partial secret $m(\mathfrak{v})$, and there is a mapping $f(\mathfrak{v})$ that maps $\mathfrak{v}$ to a leaf of Q that represents an expression of the form $L_i = v_{i,t}$. (Note that the algorithms mark all nodes of R, but from now on we will only need the marks of the leaves.)

The first part of the ciphertext is $\tilde{C} = MY^r$ and $C_0 = B^r$, which encodes the value r and the secret message M.

Algorithm 1: `EncodeSecret`(T,r)

```
Input  : Tree R, Subtree $T$ of $R$ (represented by its root), number $r$
Output : $m : V(R) \to \mathbb{Z}_p^*$

$m(T) \longleftarrow r$;
if $T$ is no leaf then
    if $T$ is $\wedge$-rooted then
        Let the number of child nodes be $n$.
        $r_i \xleftarrow{R} \mathbb{Z}_p^*, 1 \leq i \leq n$, such that $\sum_i r_i = r$.
        forall children $c$ do EncodeSecret$(c, r_i)$
    else
        forall children $c$ do EncodeSecret$(c, r)$
    end
end
```

The basic idea of our approach is to encrypt every leaf's partial secret $m(\mathfrak{v})$ with either the constant value represented by $\tilde{M}(\mathfrak{v})$ or — if $\tilde{M}(\mathfrak{v}) = \mathscr{F}$ — with the attribute $f(\mathfrak{v})$. Wlog, assume that the last inner nodes of every path to a leaf are $\wedge$-gates (if such a last inner node is an $\vee$, replace every leaf $\mathfrak{v}$ of that gate with an $\wedge$-gate having the sole child $\mathfrak{v}$). For each of these last inner $\wedge$-gates $\mathfrak{v}$, the encryptor computes ciphertext components $CT^{(\mathfrak{v})}$ for the primary cryptosystem as follows:

Case 1: If all children of $\mathfrak{v}$ are either $\top$ or $\mathscr{F}$, encode a genuine $\wedge$-gate as follows: Pick random values $r_i^{(\mathfrak{v})}$, $\forall i = 1 \dots n$, such that $m(\mathfrak{v}) = \sum_i r_i^{(\mathfrak{v})}$. For each attribute $1 \le i \le n$ set $C_{i,1}^{(\mathfrak{v})} = g_1^{r_i^{(\mathfrak{v})}}$ and compute $\{C_{i,t,2}^{(\mathfrak{v})}\}_{1 \le t \le n_i}$ as follows: if the ith attribute is not found in the children of $f(\mathfrak{v})$ (i.e., the value is *don't care*) or the attribute value $v_{i,t}$ is found in the children, set $C_{i,t,2}^{(\mathfrak{v})} = A_{i,t}^{r_i^{(\mathfrak{v})}}$; otherwise (i.e., the value $v_{i,t}$ is forbidden for this attribute), select $C_{i,t,2}^{(\mathfrak{v})}$ randomly.

Case 2: If one of the children is $\perp$, the decryption must never succeed. In this case, all $C_{i,1}^{(\mathfrak{v})}$ and $C_{i,t,2}^{(\mathfrak{v})}$ are set to random values.

Case 3: If all children are marked rand, flip a coin to decide whether to proceed with Case 1 (encrypting with a random $\wedge$-gate) or with Case 2.

Finally, compute the ciphertext components $H^{(\mathfrak{v})} = \overline{Y}^{m^{(\mathfrak{v})}}$ and $\overline{C_0}^{(\mathfrak{v})} = \overline{B}^{m(\mathfrak{v})}$. This encrypts an additional ciphertext in the secondary cryptosystem which equals to 1.

Combining these components, the ciphertext for leaf $\mathfrak{v}$ is

$$CT^{(\mathfrak{v})} = \left\langle \overline{C_0}^{(\mathfrak{v})}, H^{(\mathfrak{v})}, \left\{ C_{i,1}^{(\mathfrak{v})}, \left\{ C_{i,t,2}^{(\mathfrak{v})}, \right\}_{1 \le t \le n_i} \right\}_{1 \le i \le n} \right\rangle .$$

The final ciphertext is

$$CT = \left\langle \tilde{C}, C_0, \left\{ CT^{(\mathfrak{v})} \forall \text{leaves } \mathfrak{v} \right\} \right\rangle ,$$

along with a topological description of the tree (including the labels but excluding any other marks).

3.3 Decryption

In order to decrypt, the decryptor determines which leaves his attribute set satisfies. This is done by decrypting the second encrypted value using the second cryptosystem with all attributes that he has and comparing the result to the value 1. If the decryptor's attribute set does not fulfill the policy, he gets a value different from 1. For each leaf $\mathfrak{v} \in \mathbb{L}(R)$ set $C'^{(\mathfrak{v})}_{i,2} = C_{i,t_i,2}^{(\mathfrak{v})}$, where $1 \le i \le n$ and $L_i = v_{i,t_i}$ and compute

$$M^{(\mathfrak{v})} = \prod_{i=1}^{n} \frac{e(C_{i,1}^{(\mathfrak{v})}, D_{i,1})}{e(C'^{(\mathfrak{v})}_{i,2}, D_{i,2})} \qquad \text{and} \qquad \tau^{(\mathfrak{v})} = \frac{H^{(\mathfrak{v})} \cdot M^{(\mathfrak{v})}}{e(\overline{C_0}^{(\mathfrak{v})}, \overline{D}_0)} .$$

For each $\mathfrak{v}$, $M^{(\mathfrak{v})} = e(g_1, g_2)^{s \cdot m(\mathfrak{v})}$, where s is specific to the used attribute set and was set in the KeyGen algorithm, and $\tau^{(\mathfrak{v})} = 1$ if the leaf can be satisfied by the decryptor, and otherwise a random value. Note that if the decryptor can not satisfy the leaf, $\tau^{(\mathfrak{v})}$

might also be equal to 1. However, the probability for this occurring is $1/p$ for p the order of the bilinear group, which is negligible.

Note that while the decryptor now knows which parts of the tree he satisfies, he does not know the policies of the respective leaves since their configuration is hidden by the construction. However, with all $\tau^{(\mathfrak{v})}$, he is able to decrypt as follows: First he removes some of the leaves that he does not satisfy (i.e., where $\tau^{(\mathfrak{v})} \neq 1$) as they do not contain any information that he can use. For all $\tau^{(\mathfrak{v})} \neq 1$, replace $\mathfrak{v}$ with the constant value $\perp$ and simplify the tree by substituting subtrees with their obvious results using the formulas $A \wedge \perp = \perp$ and $A \vee \perp = A$. The remaining tree either contains only leaves that can be satisfied or is a single node $\perp$. In the latter case, return $\perp$ (as the attribute set does not satisfy the policy). For each remaining $\vee$-node N, randomly choose a subtree of N and substitute N with it. (This works because Algorithm 1 encoded the same value in all subtrees of an $\vee$ node, so we can use any of them to retrieve it.)

Finally, collapse all remaining $\wedge$-nodes to a single one. The message M can now be retrieved as

$$M = \frac{\tilde{C} \prod_{\mathfrak{v}} M^{(\mathfrak{v})}}{e(C_0, D_0)} ,$$

where $\mathfrak{v}$ are all remaining leaves of that single $\wedge$-node. By multiplying a valid combination of $M^{(\mathfrak{v})}$ together, the partial secrets $m(\mathfrak{v})$ add up to the secret value r which then is unblinded by the above formula.

Correctness. Using a secret key SK_L that satisfies the tree, we have

$$\begin{aligned}
\frac{\tilde{C} \prod_{\mathfrak{v}} M^{(\mathfrak{v})}}{e(C_0, D_0)} &= M \cdot \frac{Y^r}{e(C_0, D_0)} \prod_{\mathfrak{v}} \prod_i \frac{e(C_{i,1}^{(\mathfrak{v})}, D_{i,1})}{e(C'^{(\mathfrak{v})}_{i,2}, D_{i,2})} \\
&= M \cdot e(g_1, g_2)^{\omega r - \beta r \beta^{-1}(\omega - s) + \sum_{\mathfrak{v}} \left(\sum_i (r_i^{(\mathfrak{v})} \cdot (s + a_{i,t_i} \lambda_i) - (a_{i,t} r_i^{(\mathfrak{v})} \lambda_i) \right)} \\
&= M \cdot e(g_1, g_2)^{-rs + \sum_{\mathfrak{v}} \left(\sum_i r_i^{(\mathfrak{v})} s \right)} .
\end{aligned}$$

The tree is constructed such that for a leaf $\mathfrak{v}$ that is satisfied, $\sum_i r_i^{(\mathfrak{v})} = m(\mathfrak{v})$ and for a sufficient subset of the leaves, $\sum_{\mathfrak{v}} m(\mathfrak{v}) = r$, so $\sum_{\mathfrak{v}} \left(\sum_i r_i^{(\mathfrak{v})} s \right) = rs$ and the equation yields $M \cdot e(g_1, g_2)^{rs - rs} = M$. Note that if the equation is computed using a key SK_L that does not satisfy the tree, then some $C'^{(\mathfrak{v})}_{i,2}$ will be random values instead of $g_1^{a_{i,t} r_i^{(\mathfrak{v})}}$. In this case, some $m(\mathfrak{v})$ will not be computed correctly, so the exponents do not cancel out and the result will be different from M (with overwhelming probability).

4 Discussion

In this section we discuss the properties of our extended construction. For sake of space, the formal security proof is given in Appendix C.

4.1 Anonymity of the Policy

In our construction the ciphertext is encrypted with a major of the syntax tree. As the leaves of this tree are hidden from an adversary, he cannot decide which of the possible policies was actually used. The anonymity set $\mathbb{A}(E,L)$ is determined by the ciphertext E and the attribute set of the decryptor $L = [L_1,\ldots,L_n]$. We will now briefly discuss the size of $\mathbb{A}(E,L)$. As a lower bound, assume $n_i = 2$ for all $1 \leq i \leq n$, i.e., every attribute has only two possible symbolic values. If the decryptor can access an $\wedge$-gate with his attribute set L, he can conclude that each ith attribute encoded in the policy of the $\wedge$-gate is either set to the value L_i that he owns or is a "don't care". Similarly, if he cannot decrypt an $\wedge$-gate, he can conclude that there is at least one attribute i in the policy of the $\wedge$-gate that is unequal to his attribute value L_i. In both cases the number of possible $\wedge$-gates is $O(2^n)$. For a tree R with leaves $\mathbb{L}(R)$, the number of possible policies is in $O(2^{n\cdot|\mathbb{L}(R)|})$.

In some scenarios, it might suffice if the attacker knows only the general form of the policy, i.e., he wants to know, which nodes of the tree belong to the actual policy and which ones are dummy gates introduced in order to obfuscate the policy. In our construction, the form of the policy is determined by the leaves. Some of these are set to a constant value ($\top$ or $\bot$) to render unused inner nodes inoperative, some are genuine $\wedge$-gates encoding parts of the policy. Thus, to find out which form the original policy has, an attacker must know which $\wedge$-gates are constant values and which ones are not, which for a tree R with leaves $\mathbb{L}(R)$ gives $O(2^{|\mathbb{L}(R)|})$ combinations. However, for reasons of symmetry, some of these forms may be topologically identical, so the number of forms might be smaller than that. The most symmetries are found in a complete n-ary tree. However, in [11] it is shown that even in such a tree, the number of topologically different subtrees is exponential in the number of nodes, so it is at least exponential in the number of leaves. Thus, even taking into account symmetries, the number of possible forms of a policy encoded in a syntax tree major is exponential in the number of leaves.

We show that an attacker cannot distinguish between policies within his anonymity set in Appendix D.

4.2 Comparison with Nishide's Construction

The partial ciphertext $CT^{(\mathfrak{v})}$ of a leaf $\mathfrak{v}$ has roughly the same size as a complete encryption of Nishide's original construction from [14]. In the case where the tree consists only of a single leaf, the ciphertext of our construction is even a bit larger than it would be in the original one, because we store additional values that enable the decryptor to determine whether he is eligible to decrypt. It is natural to ask if this is an improvement.

Concretely, one could transform a policy to DNF form with n conjunctions and encrypt each conjunction separately, creating n ciphertext instances of Nishide's construction. This requires approximately as much memory as using our construction for a tree with n leaves. This is of course only feasible if the policy in question has a small DNF representation, i.e., one with a small number of conjunctions. The leaves in our scheme represent conjunctions, but only of parts of the encoded policy, and they can be combined in various ways following the description of the major which is sent along with the ciphertext. For example, the 2-CNF formula shown in Figure 6 can be encoded with 10 leaves, but its DNF representation consists of $2^5 = 32$ conjunctions, so 32 instances of Nishide's construction would be needed to encode it. This is not a problem in our construction as the original 2-CNF form can be used for the encryption. (Note that when using the 2-CNF syntax tree as a major for encryptions, the policy automatically supports $\wedge$-gates as leaves, so it is actually stored as a conjunction of DNFs. This form has been called CDNF in [21] and is considered very expressive.)

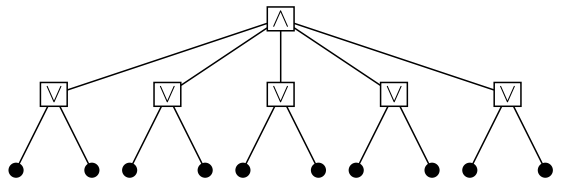


Fig. 6: 2-CNF of size 5

4.3 Reducing the Size of the Ciphertext

For each leave's encryption every attribute of the system is used. This is the only way to maximize the anonymity set, because when some attribute A is not used for the decryption of a leaf $\mathfrak{v}$, then the decryptor can obviously conclude that the partial policy of $\mathfrak{v}$ does not contain A. However, if the universe of attributes of a given system is very large, it might be feasible to use only a comparatively small set of attributes for the encryption of each leaf while still using enough attributes to get a sufficiently large anonymity set. Similarly to [14], each leaf $\mathfrak{v}$ may be encrypted with its own set of attributes $\mathscr{A}_{\mathfrak{v}}$. $\mathscr{A}_{\mathfrak{v}}$ can be a random superset of the set attributes actually used in the leaf. However, in order to hide as much of the semantics of each partial policy, some care must be taken, as it should be understood which information an attacker gains by the knowledge of $\mathscr{A}_{\mathfrak{v}}$. It must also be considered that $\mathscr{A}_{\mathfrak{v}}$ must be sent along with each leaf, which slightly increases the size of the ciphertext.

As a more systematic approach, the universe of attributes could be partitioned into different domains $\mathbb{D}_i, 1 \leq i \leq n_D$ with n_D the number of domains. For example one domain $\mathbb{D}_1$ could contain all user-specific attributes, $\mathbb{D}_2$ could contain all device-specific attributes, $\mathbb{D}_3$ all location-specific attributes, etc. If each domain $\mathbb{D}_i$ consists of $|\mathbb{D}_i|$ attributes, then the anonymity set of a respective leaf $\mathscr{A}_{\mathfrak{v}}$ with $\mathscr{A}_{\mathfrak{v}} = \mathbb{D}_i$ is $O(2^{|\mathbb{D}_i|})$. With this approach, an advisor knows that a leaf $\mathfrak{v}$ with $\mathscr{A}_{\mathfrak{v}} = \mathbb{D}_1$ might encode a partial policy over some user-specific properties (or as always an encoding of $\perp$ or $\top$), but he does not know which one or which ones. This gives the encryptor precise control over what information is disclosed with an encrypted leaf. Moreover, instead of listing each element of $\mathscr{A}_{\mathfrak{v}}$, with this approach only the index i of $\mathbb{D}_i$ needs to be sent along with the partial ciphertext of $\mathfrak{v}$, $CT^{(\mathfrak{v})}$.

5 Conclusion

We introduced the notion of policy anonymity in cryptographic access control. To this end, we proposed the idea to obfuscate the policy used in an encryption by constructing a syntax tree major of the syntax tree that encodes the desired policy. The leaves are then hidden from an adversary using a cryptographic primitive. We discussed how these majors can be characterized and how to configure the leaves to encode a specific, given policy in one of its majors. The majors can be chosen arbitrarily large, and the larger a major is the larger becomes the anonymity set. From the anonymity set, an adversary gains only very general informations about the encoded policy; for example he knows an upper bound of its complexity and that some of his attribute sets satisfy certain parts of the major. We then used these primitives to modify a CP-ABE scheme with partially hidden policies to support every policy that can be expressed as a Boolean formula and enable an encryptor to obfuscate that policy.

Our construction compares favorably to [14] as it is able to efficiently encode any policy that can be expressed as a Boolean formula and is not limited to policies with small DNF represenations and to the various Predicate Encryption schemes. However, it may be possible to construct a scheme that hides even more properties of the encoded policy by using a different encoding of the it, like garbled circuits which presently have been utilized to solve different problems [8, 18]. We leave this as future work. Also, our approach may be applicable to other CP-ABE system that like Nishide's support only $\wedge$-conjunctions.

References

1. *2007 IEEE Symposium on Security and Privacy (S&P 2007), 20-23 May 2007, Oakland, California, USA*. IEEE Computer Society, 2007.
2. J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy* [1], pages 321–334.
3. P. Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1-3):217–239, 2005.
4. D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical identity based encryption with constant size ciphertext. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2005.
5. D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In S. P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 535–554, Amsterdam, The Netherlands, February 2007. Springer.
6. L. Cheung and C. C. Newport. Provably secure ciphertext policy ABE. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 456–465. ACM, 2007.
7. V. Ciriani, S. D. C. di Vimercati, S. Foresti, and P. Samarati. *k*-anonymity. In T. Yu and S. Jajodia, editors, *Secure Data Management in Decentralized Systems*, volume 33 of *Advances in Information Security*, pages 323–353. Springer, 2007.
8. K. B. Frikken, J. Li, and M. J. Atallah. Trust negotiation with hidden credentials, hidden policies, and policy cycles. In *NDSS*. The Internet Society, 2006.
9. J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In N. P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2008.
10. A. B. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In H. Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 62–91. Springer, 2010.

11. D. W. Matula. On the number of subtrees of a symmetric n-ary tree. *SIAM Journal on Applied Mathematics*, 18(3):668–703, May 1970.
12. S. Müller and S. Katzenbeisser. A new DRM architecture with strong enforcement. In *ARES*, pages 397–403. IEEE Computer Society, 2010.
13. S. Müller, S. Katzenbeisser, and C. Eckert. On multi-authority ciphertext-policy attribute-based encryption. *Bulletin of the Korean Mathematical Society (B-KMS)*, 46(4):803–819, July 2009.
14. T. Nishide, K. Yoneyama, and K. Ohta. Attribute-based encryption with partially hidden ciphertext policies. *IEICE Transactions*, 92-A(1):22–32, 2009.
15. N. Nishimura, P. Ragde, and D. M. Thilikos. Finding smallest supertrees under minor containment. In P. Widmayer, G. Neyer, and S. Eidenbenz, editors, *WG*, volume 1665 of *Lecture Notes in Computer Science*, pages 303–312. Springer, 1999.
16. K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vision Comput.*, 27(7):950–959, 2009.
17. F. Rosselló and G. Valiente. An algebraic view of the relation between largest common subtrees and smallest common supertrees. *CoRR*, abs/cs/0604108, 2006.
18. A. Sahai and H. Seyalioglu. Worry-free encryption: functional encryption with public keys. In E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 463–472. ACM, 2010.
19. A. Serjantov and G. Danezis. Towards an information theoretic metric for anonymity. In R. Dingledine and P. F. Syverson, editors, *Privacy Enhancing Technologies*, volume 2482 of *Lecture Notes in Computer Science*, pages 41–53. Springer, 2002.
20. E. Shi, J. Bethencourt, H. T.-H. Chan, D. X. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *IEEE Symposium on Security and Privacy* [1], pages 350–364.
21. N. P. Smart. Access control using pairing based cryptography. In M. Joye, editor, *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 111–121. Springer, 2003.
22. G. Valiente. Constrained tree inclusion. *J. Discrete Algorithms*, 3(2-4):431–447, 2005.
23. B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *The 14th IACR International Conference on Practice and Theory of Public Key Cryptography, PKC*, Mar. 2008.
24. S. Yu, K. Ren, and W. Lou. Attribute-based content distribution with hidden policy. In *Secure Network Protocols, 2008, NPSEC*, 2008.

A Embedding a Formula in a Syntax Tree Major

In this section, we construct a mapping $\tilde{M} : V(R) \rightarrow \{\mathscr{F}, \top, \bot, \text{rand}\}$ that allows us to embed a monotonic syntax tree Q in a syntax tree major R such that R computes the same function as Q. This is achieved by mapping some of the leaves of R to the constant values $\bot$ or $\top$. The effect of this is that some inner nodes of R that are connected to these leaves will map to trivial formulas like $x \wedge \top$ or $x \wedge \bot$, which are equal to x or $\bot$. For an example, we refer to Figure 3a. The remaining leaves (that are not set to $\top$ or $\bot$) are then mapped to leaves of Q such that R evaluates the same function as Q.

Finding such a mapping can be done in a two step process. In the first step, we mark all inner nodes of R that are needed to encode nodes of Q with a special symbol that we call $\mathscr{F}$. For convenience, all other nodes will be marked $\emptyset$. Call this mapping $M : V(R) \rightarrow \{\emptyset, \mathscr{F}\}$. As R is a syntax tree major of Q, this marking can be found by the following iterative algorithm: For each leaf of Q, a corresponding leaf in R is selected and the path from the root of R to this node is marked by traversing upwards (see Algorithm 2).

In the second step, the other inner nodes (not marked with $\mathscr{F}$ by Algorithm 2) are marked with $\top$ or $\bot$, such that they do not affect the computation of the encoded policy. Furthermore, we mark all leaves that have no impact on the computation with the value rand. This value shall indicate that the leaf can be chosen randomly from $\{\bot, \top\}$ or

Algorithm 2: MarkRelevantNodes(R, Q, f)

```
Input  : $R$, $Q$, $f : V(R) \to V(Q)$
Output : $M : V(R) \to \{\emptyset, \mathscr{F}\}$

forall $T \in V(R)$ do $M(T) \longleftarrow \emptyset$;
$M(\mathrm{root}(R)) \longleftarrow \mathscr{F}$;
foreach leaf u of $Q$ do
    $T \overset{\mathrm{R}}{\longleftarrow}$ leaf of $f^{-1}(\mathtt{u})$;
    while $M(T) \neq \mathscr{F}$ do
        $M(T) \longleftarrow \mathscr{F}$;
        $T \longleftarrow \mathrm{parent}(T)$;
    end
end
```

any policy. This will be useful in the cryptographic construction. Algorithm 3 marks all nodes that were marked with $\emptyset$ to $\perp$, $\top$, or rand. Call this mapping $\tilde{M} : V(R) \to \{\mathscr{F}, \top, \perp, \mathrm{rand}\}$. The algorithm marks the nodes as follows: If a node is an $\wedge$-node marked $\mathscr{F}$, all children marked $\emptyset$ will recursively be set to $\top$. Similarly, if a node is an $\vee$-node marked $\mathscr{F}$, all children marked $\emptyset$ will recursively be set to $\perp$. For nodes that are not marked $\mathscr{F}$, an appropriate mapping of the children to $\perp$, $\top$, or rand is chosen, such that the nodes do not influence the computation.

Algorithm 3: SetNodeValue(T, m)

```
Input  : Tree $R$ with label $M : V(R) \to \{\emptyset, \mathscr{F}\}$, subtree $T$ of $R$ (represented by its root),
         $m \in \{\mathscr{F}, \perp, \top, \mathrm{rand}\}$
Output : $\tilde{M} : V(R) \to \{\mathscr{F}, \perp, \top, \mathrm{rand}\}$

if $T$ is not a leaf then
    $\tilde{M}(T) \longleftarrow m$ ;
    if $T$ is an $\wedge$-node then $t \longleftarrow \perp$ else $t \longleftarrow \top$;
    switch $m$ do
        case $\mathscr{F}$
            foreach child $c, M(c) = \mathscr{F}$ do SetNodeValue(c, $\mathscr{F}$);
            foreach child $c, M(c) \neq \mathscr{F}$ do SetNodeValue(c, not $t$);
        end
        case $t$
            Select a random child $c$;
            SetNodeValue(c, $t$);
            forall other children $c$ do SetNodeValue(c, rand);
        end
        otherwise
            forall children $c$ do SetNodeValue(c, $m$);
        end
    end
end
```

The following theorem states that these algorithms are correct:

Theorem 1. *Let R be a syntax tree major of a monotonic syntax tree Q and let f be the characteristic function of this major, as described in Definition 3. Applying the function*

$\phi : \mathbb{L}(R) \to \mathbb{L}(Q) \cup \{\bot, \top\}$*, defined as*

$$\phi(v) = \begin{cases} f(v), & \textit{if } \tilde{M}(v) = \mathscr{F}, \\ \tilde{M}(v), & \textit{otherwise} \end{cases}$$

to the leaves of R yields a tree $\phi(R)$*, which is semantically equal to Q, i.e., for all configurations* $\psi : \mathbb{L}(Q) \to \{\top, \bot\}$*,* $\psi(\phi(R)) = \psi(Q)$*.*

For every node $q \in Q$, let $T(q) := R[f^{-1}(q)]$ be the subtree of R that q is expanded into. Consider Definition 3. As f is a surjection, all nodes of Q are reached by applying f to the nodes of R. By (1), for each node $q \in Q$, $f^{-1}(q)$ is a connected subtree and by (2), all expanded subtrees $T(q)$ of nodes q are disjunct.

We will show that after applying ϕ to the leaves of R, each subtree of Q starting with any $q \in Q$ is semantically equivalent to the respective subtree of R rooted at the root of $T(q)$, i.e., they compute the same function for all configurations $\psi : \mathbb{L}(Q) \to \{\bot, \top\}$. For any node q and extended subtree $T(q)$ we will write $q \equiv T(q)$ if this is the case.

We need the following lemmas:

Lemma 1. *For the root* r_R *of R and the root* r_Q *of Q, it holds that* $f(r_R) = r_Q$*.*

Proof. Suppose not. Then $f(r_R)$ has a predecessor x and is connected to this predecessor by an edge (x, r_R). By (3) of Definition 3, this edge is preserved in R, so r_R also has a predecessor. This contradicts the assumption that r_R is a root. □

Lemma 2. *Let R be a syntax tree major of Q, and let Q and R be equally rooted (i.e., both are* $\wedge$*-rooted or both are* $\vee$*-rooted). Then for each node* $q \in Q$*, the root of the subtree* $T(q)$ *has the same (implicit) label as q.*

Proof. We prove this lemma by induction over the structure of the tree. Lemma 1 implies that the root r_Q of Q has the same label as the root of $T(r_Q)$ (because the root of $T(r_Q)$ is the root of R and both trees are equally rooted). Now let $q \in Q$ be a node for which Lemma 2 holds, i.e., the root of $T(q)$ has the same label as q. By (1) of Definition 3, $T(q)$ has even height, so the leaves of $T(q)$ also have the same label as q. Suppose that the label of q is $\wedge$ and let r be a child of q in Q. Then the label of r must be $\vee$ (by Definition 1). By the induction hypothesis, the root of $T(q)$ is labeled $\wedge$. Then the leaves of $T(q)$ are also labeled $\wedge$. Again, by (3), the edge $(r,q) \in E(Q)$ is preserved in R, so there is an edge from a leaf of $T(q)$ to the root of $T(r)$. This root must be labeled $\vee$. If the label of q is $\vee$, a similar argument holds. □

We are now in a position to prove Theorem 1 by structural induction.

Proof. Let $\mathfrak{u}$ be a leaf of Q. Algorithm 2 randomly marked a leaf $\mathfrak{v} \in T(\mathfrak{u})$ of R with $\mathscr{F}$, as well as all nodes of the single path from $\mathfrak{v}$ to the root of $T(\mathfrak{u})$. Algorithm 3 configured all nodes on this path such that they return the value of their single marked child (this means that they compute the identify function). By this, the value of the marked leaf is propagated along the path to the root. It follows that for all configurations $\psi : \mathbb{L}(Q) \to \{\bot, \top\}$, $\psi(T(\mathfrak{u})) = \psi(\phi(\mathfrak{v})) = \psi(f(\mathfrak{v})) = \psi(\mathfrak{u})$.

Now suppose that for all children $w_i, 1 \leq i \leq n_q$ of a node q, $w_i \equiv T(w_i)$ (where n_q is the number of children of q). We need to show that this implies that $q \equiv T(q)$. By (3) of Definition 3, the unexpanded edges of Q are preserved in R. This means that for each edge $(q, w_i) \in E(Q)$ there is an edge in R that that connects a leaf of $T(q)$ to the root of

$T(w_i)$. From Lemma 2 we know that the root of $T(q)$ and all leaves of $T(q)$ have the same label as q. From (4) of Definition 3 we conclude that marked paths meet only at nodes that have the same label as the root of $T(q)$. All other nodes on the marked paths are set by Algorithm 3 such that they compute the identity function, and all other nodes that are unmarked are configured to have no influence on the computation.

If q is an $\wedge$-gate (q computes $w_1 \wedge w_2 \wedge \ldots \wedge w_{n_q}$) then the roots of all $T(w_i)$ will be connected by $\wedge$-gates in $T(q)$ and $T(q)$ computes $T(w_1) \wedge T(w_2) \wedge \ldots \wedge T(w_{n_q})$ But we know that $w_i \equiv T(w_i)$, so $q \equiv T(q)$. Similarly, if q is an $\vee$-gate (q computes $w_1 \vee w_2 \vee \ldots \vee w_{n_q}$) then the roots of all $T(w_i)$ will be connected by $\vee$-gates in $T(q)$ and $T(q)$ computes $T(w_1) \vee T(w_2) \vee \ldots \vee T(w_{n_q})$. But we know that $w_i \equiv T(w_i)$, so $q \equiv T(q)$. □

A direct consequence of this Theorem is:

Corollary 1. *Every syntax tree major R of a monotonic syntax tree Q semantically contains Q.*

B Finding a Small Common Syntax Tree Major

We will now briefly discuss one way to find a common syntax tree major of a set of syntax tree. Ideally, we would like such combinations of trees to be as small as possible. This problem is known as the smallest common supertree problem and is well-studied for tree majors [17, 22, 16]. Generally, this problem is NP-hard, but by adding some reasonable constraints on the input trees, it becomes tractable. We can adopt the algorithm for finding the smallest common major of two trees described by Nishimura et al. [15] to our definition of syntax tree majors. The only constraint of Nishimura's algorithm is that the input trees must have a bounded degree. This is reasonable for our settings as a sufficient upper bound for all realistic syntax trees of a particular scenario can easily be estimated.

As our definition of syntax tree majors is based on the definition of tree majors used in [15], we can modify Nishimura's algorithm to work for our definition of syntax tree majors. We briefly describe the necessary modifications: The most important restriction in our definition compared to the original one is that nodes of tree R can only be combined with nodes of Q that have equal labels. As the algorithm of [15] always combines the roots of Q and R, both trees must be either $\wedge$-rooted or $\vee$-rooted. If they are not equally rooted, construct a new node N and attach either Q or R as a subtree. The main loop of the algorithm tries all possible combinations of nodes of Q with nodes of R. For syntax tree majors, we must restrict these combinations to combinations that map nodes of equal labels, i.e., the distances of the combined nodes to their respective root must both be even or odd. Finally, the remainder of the algorithm must be restricted such that only mappings that preserve the labels and furthermore adhere to (4) are tried.

With this modification the algorithm can be utilized by the encryptor to systematically construct a tree R that is a syntax tree major of a set of trees: In addition to the monotonic syntax tree Q, select some monotonic syntax trees $P_1, \ldots, P_n$. Find the smallest common tree major of all Q and P_i as follows: Set $Q_0 := Q$ and for all $i \in \{1, \ldots, n\}$ let Q_i be the smallest common major of Q_{i-1} and P_i. The resulting tree Q_n is a syntax tree major of $P_1, \ldots, P_n$ and Q and can be used to encode any of the formulas encoded by Q, any P_i, and numerous combinations.

C CPA-Security

As the system uses the same structures as Nishide's construction, its CPA-security can directly be derived from it. We can model the security using a standard security game as for example used in [2]. As we only claim security in the generic group model, the more powerful non-selective version can be used:

Setup The challenger runs the Setup algorithm and gives the public key PK to the adversary.

Phase 1 The adversary queries the challenger for private keys corresponding to lists of attributes $L = [L_1, L_2, \ldots, L_n] = [v_{1,t_1}, v_{2,t_2}, \ldots, v_{n,t_n}]$.

Challenge The adversary declares two messages M_0 and M_1 and a policy W. The policy must not be satisfied by any of the queried attribute lists. The challenger flips a random coin $b \in \{0, 1\}$ and encrypts M_b under W, producing CT. It gives CT to the adversary.

Phase 2 The adversary quieries the challenger for private keys corresponding to lists of attributes $L = [L_1, L_2, \ldots, L_n] = [v_{1,t_1}, v_{2,t_2}, \ldots, v_{n,t_n}]$ with the added restriction that none of these lists must satisfy W.

Guess The adversary outputs a guess b' for b. The advantage for the adversary in this game is defined to be $Pr[b = b'] - \frac{1}{2}$.

The CP-ABE system is said to be secure if all polynomial time adversaries have at most negligible advantage in this security game.

Theorem 2. *The construction given in Section 3 is secure in the generic group model.*

Proof. We show how any adversary who plays the CP-ABE game (denoted $\texttt{Adv}_1$) can be used to construct an adversary in a slightly modified game ($\texttt{Adv}_2$). Then we prove that no such $\texttt{Adv}_2$ can exists, so no $\texttt{Adv}_1$ can exist, either. We define the modified game in the following manner: The phases **Setup**, **Phase 1**, and **Phase 2** are equal to the CP-ABE game. In the **Challenge** phase, the adversary declares a policy W. The policy must not be satisfied by any of the queried attribute lists. The challenger flips a random coin $b \in \{0, 1\}$ and encrypts M_b under W, producing CT, but instead of computing $\tilde{C}$ as described in the algorithm, he computes $\tilde{C}$ as

$$\tilde{C} = \begin{cases} Y^r, & \text{if } b = 1 \\ e(g_1, g_2)^{\theta}, & \text{if } b = 0, \end{cases}$$

where θ is chosen randomly from $\mathbb{Z}_p^*$. The task of $\texttt{Adv}_1$ is to distinguish the two group elements $Y^r = e(g_1, g_2)^{\omega r}$ and $e(g_1, g_2)^{\theta}$.

Lemma 3. *If there exists an adversary* $\texttt{Adv}_1$ *who has advantage of* ε *to win the original CP-ABE game, then there exists an adversary* $\texttt{Adv}_2$ *who wins the modfied game with advantage* $\varepsilon/2$.

Proof. Given an adversary $\texttt{Adv}_1$ that has advantage ε in the CP-ABE game, we can construct an adversary $\texttt{Adv}_2$ as follows: $\texttt{Adv}_2$ simulates $\texttt{Adv}_1$. In the phases **Setup**, **Phase 1**, and **Phase 2**, $\texttt{Adv}_2$ forwards all messages he receives from $\texttt{Adv}_1$ to the challenger, and all messages from the challenger to $\texttt{Adv}_1$. In the **Challenge** phase, $\texttt{Adv}_2$

receives to messages M_0 and M_1 from $\texttt{Adv}_1$ and the challenge CT (which contains $\tilde{C}$ that is either $e(g_1,g_2)^{\omega r}$ or $e(g_1,g_2)^{\theta}$ for a random θ) from the challenger. He flips a coin b and multiplies $\tilde{C}$ by M_b and sends the resulting ciphertext CT$'$ to $\texttt{Adv}_1$. When $\texttt{Adv}_1$ outputs a guess b', $\texttt{Adv}_2$ outputs 1 if $b' = b$ and 0 if $b' \neq b$. If for the original $\tilde{C} = e(g_1,g_2)^{\omega}$, then $\texttt{Adv}_2$'s challenge given to $\texttt{Adv}_1$ is a well-formed CP-ABE ciphertext and $\texttt{Adv}_1$ has advantage ε of guessing the correct $b' = b$. Otherwise, the challenge is independent of the message M_0 and M_1, so the advantage of $\texttt{Adv}_2$ is 0. Thus, we have

$$\begin{aligned}\Pr[\texttt{Adv}_2 \text{ succeeds}] &= \Pr[\tilde{C} = Y^r]\Pr[\beta' = \beta \,|\, \tilde{C} = Y^r] + \\ &\quad \Pr[\tilde{C} = e(g_1,g_2)^{\theta}]\Pr[\beta' \neq \beta \,|\, \tilde{C} = e(g_1,g_2)^{\theta}] \\ &\leq \frac{1}{2}(\frac{1}{2}+\varepsilon) + \frac{1}{2}\cdot\frac{1}{2} = \frac{1+\varepsilon}{2}\end{aligned}$$

To prove Theorem 2, we use the asymmetric case of the Generic Diffie-Hellman Exponent problem [4]:

Definition 4 (Dependant Polynomials, Asymmetric case). *Let $P,Q,R \in \mathbb{F}_p[x_1,\dots,x_n]^s$ be three s-tuples of n-variate polynomials over $\mathbb{F}_p$. Write $P = (p_1,p_2,\dots,p_s)$, $Q = (q_1,q_2,\dots,q_s)$ and $R = (r_1,r_2,\dots,r_s)$ where $p_1 = q_1 = r_1 = 1$. We say that a polynomial $f \in \mathbb{F}_p[x_1,\dots,x_n]$ is dependent on the sets (P,Q,R) if there exist $2s^2+s$ constants $\{a_{i,j}\}_{i,j=1}^{s}, \{b_{i,j}\}_{i,j=1}^{s}, \{c_k\}_{k=1}^{s}$ such that*

$$f = \sum_{i=1}^{s}\sum_{j=1}^{s} a_{i,j}p_iq_j + \sum_{i=1}^{s}\sum_{j=1}^{s} b_{i,j}q_iq_j + \sum_{k=1}^{s} c_kr_k$$

We say that f is independent of (P,Q,R) if f is not dependent on (P,Q,R).

Boneh et al. prove the following theorem:

Theorem 3. *Let $P,Q,R \in \mathbb{F}_p[x_1,\dots,x_n]^s$ be three s-tuples of n-variate polynomials over $\mathbb{F}_p$ and let $f \in \mathbb{F}_p[x_1,\dots,x_n]$. Let $d = \max(\deg(P)+\deg(Q), 2\deg(Q), \deg(R), \deg(f))$. If f is independent of (P,Q,R) then any $\mathscr{A}$ that has advantage $1/2$ in solving the decision (P,Q,R,f)-Diffie-Hellman Problem in a generic bilinear group $(\mathbb{G}_1,\mathbb{G}_2)$ must take time at least $\Omega(\sqrt{p/d}-s)$.*

We need to show that the adversary's target term $f = \omega r$ is independent of (P,Q,R), where (P,Q,R) are the polynomials from $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ that he knows. After the challenge phase, the adversary has the following terms:

- The public system key PK: $\{a_{i,t} \in \mathbb{Z}_p^*\}_{1\leq t\leq n_i, 1\leq i\leq n}, \beta, \overline{\beta} \in P$, $\omega, \overline{\omega} \in R$,
- Private keys SK$_L$ for several queries L. Let the number of these sets be N and denode the jth query $L^{(j)}$. For each query $1 \leq j \leq N$, the challenger created a secret value $s^{(j)}$ and random values $\lambda_i^{(j)}, 1 \leq i \leq n$ (n being the number of attributes): $\beta^{-1}(\omega - s^{(j)}), \overline{\beta}^{-1}(\overline{\omega} - s^{(j)}), s^{(j)} + a_{i,t_i}\lambda_i^{(j)}, \lambda_i^{(j)} \in Q$. No $L^{(j)}$ satisfies the policy per the definition of the security game.
- The ciphertext components
  - C_0: $\beta r \in P$ for a randomly chosen r

- For all leaves that contain genuine $\wedge$-gates and are satisfied for an $L^{(j)}$ $C_{i,1}^{(\mathfrak{v})}, 1 \leq i \leq n$ and $C_{i,t,2}^{(\mathfrak{v})}, 1 \leq t \leq n_i, 1 \leq i \leq n$: $r_i^{(\mathfrak{v})}$, $a_{i,t} r_i^{(\mathfrak{v})} \in P$
- For all leaves the unblinding values of the secondary cryptosystem $H^{(\mathfrak{v})}$ and $\overline{C_0}^{(\mathfrak{v})}$: $\overline{\omega} m^{(\mathfrak{v})} \in R$, $\overline{\beta} m^{(\mathfrak{v})} \in P$

From this examination we follow that the polynomial vectors P, Q, and R are as follows:

$$P = \begin{pmatrix} 1, \beta, \overline{\beta}, \beta r, \\ \{a_{i,t} \in \mathbb{Z}_p^*\}_{1 \leq t \leq n_i, 1 \leq i \leq n}, \\ \{r_i^{(\mathfrak{v})}, a_{i,t} r_i^{(\mathfrak{v})}\}_{\forall \mathfrak{v}, 1 \leq t \leq n_i, 1 \leq i \leq n}, \\ \{\overline{\beta} m^{(\mathfrak{v})}\}_{\forall \mathfrak{v}} \end{pmatrix}$$

$$Q = \begin{pmatrix} 1, \\ \{\beta^{-1}(\omega - s^{(j)})\}_{1 \leq j \leq N}, \\ \{\overline{\beta}^{-1}(\overline{\omega} - s^{(j)})\}_{1 \leq j \leq N}, \\ \{s^{(j)} + a_{i,t_i} \lambda_i^{(j)}, \lambda_i^{(j)}\}_{L_i^{(j)} = v_{i,t_i}, 1 \leq j \leq N, 1 \leq i \leq n} \end{pmatrix} \quad R = \begin{pmatrix} 1, \omega, \overline{\omega}, \\ \{\overline{\omega} m^{(\mathfrak{v})}\}_{\forall \mathfrak{v}} \end{pmatrix}$$

and $f = \omega r$.

The adversary needs to build the target polynomial ωr from P, Q, R using only the combinations implied by Definition 4. The product ωr is not contained in any term, so he needs to multiply a polynomial of Q with a polynomial of P or Q such that both factors ω and r are contained in either term. First observe which terms contain ω. There is only $\omega \in R$ which cannot be multiplied with any other polynomial as it is in R and $\beta^{-1}(\omega - s^{(j)}) \in Q$ for any $1 \leq j \leq N$. As this has β^{-1} as additional factor, the adversary must eliminate β^{-1} from any of these polynomials by multiplying it with a polynomial from P or Q that contains β. There are two possibilities to achieve this: Multiplying with $\beta \in P$ yields $(\omega - s^{(j)}) \in R$, and multiplying with $\beta r \in P$ yields $(\omega r - r s^{(j)}) \in R$. These are the only two methods to create a polynomial that contains ω, but not β^{-1}. We now examine which of these two combinations can be used to build the target polynomial ωr.

Case 1: $(\omega - s^{(j)}) \in R$. Since the adversary has to build ωr, he is required to have r as another factor. However, only terms of P and Q can be multiplied, and the current polynomial is already in R, so there is no way to build ωr.

Case 2: $(\omega r - r s^{(j)}) \in R$ for some (any) $1 \leq j \leq N$, which contains ωr. Note that this is an element of R and no other multiplication is possible. It follows that this is the only way to create the product ωr. The adversary now has to build $r s^{(j)}$ in order to remove that term from the polynomial.

To construct a polynomial that contains $r s^{(j)}$, the adversary now needs to multiply a polynomial that contains $s^{(j)}$ with a polynomial that contains r or at least $r_i^{(\mathfrak{v})}$ (as r can be constructed using a sum over some $r_i^{(\mathfrak{v})}$). Table 1 lists all results.

The only useful term is $r_i^{(\mathfrak{v})} s^{(j)} + a_{i,t_i} \lambda_i^{(j)} r_i^{(\mathfrak{v})}$. As explained in the correctness proof, summing up all attribute keys for an appropriate set of leaves, the sum becomes equal to $r s^{(j)} + \sum_{\mathfrak{v}} \sum_i a_{i,t_i} \lambda_i^{(j)} r_i^{(\mathfrak{v})}$, and adding that to our current term, we get

	βr	$r_i^{(\mathfrak{v})}$	$a_{i,t}^{(\mathfrak{v})} r_i^{(j)}$
$\beta^{-1}(\omega - s^{(j)})$	$\omega r - s^{(j)} r$	$\beta^{-1} r_i^{(\mathfrak{v})}(\omega - s^{(j)})$	$\beta^{-1} a_{i,t}^{(\mathfrak{v})} r_i^{(\mathfrak{v})} \omega - \beta^{-1} a_{i,t}^{(\mathfrak{v})} r_i^{(\mathfrak{v})} s^{(j)})$
$\overline{\beta}^{-1}(\omega - s^{(j)})$	$\overline{\beta}^{-1} \beta r(\omega - s^{(j)})$	$\overline{\beta}^{-1} r_i^{(\mathfrak{v})}(\omega - s^{(j)})$	$\overline{\beta}^{-1} r(\omega - s^{(j)}) a_{i,t}^{(\mathfrak{v})}$
$s^{(j)} + a_{i,t_i} \lambda_i^{(j)}$	$\beta r(s^{(j)} + a_{i,t_i} \lambda_i^{(j)})$	$r_i^{(\mathfrak{v})} s^{(j)} + a_{i,t_i} \lambda_i^{(j)} r_i^{(\mathfrak{v})}$	$r_i^{(\mathfrak{v})} (a_{i,t_i} s^{(j)} + a_{i,t_i}^2 \lambda_i^{(j)})$

Table 1: Pairing a term containing r with a term containing $s^{(j)}$

$$\left(\omega r - r s^{(j)}\right) + \left(r s^{(j)} + \sum_{\mathfrak{v}} \sum_{i} a_{i,t_i} \lambda_i^{(j)} r_i^{(\mathfrak{v})}\right) = \omega r + \sum_{\mathfrak{v}} \sum_{i} a_{i,t_i} \lambda_i^{(j)} r_i^{(\mathfrak{v})}$$

The adversary can then cancel out $\sum_{\mathfrak{v}} \sum_i a_{i,t_i} \lambda_i^{(j)} r_i^{(\mathfrak{v})}$ by multiplying the ciphertext components $a_{i,t}^{(\mathfrak{v})} r_i^{(j)}$ with his secret key components $\lambda_i^{(j)}$.

However, note that none of the adversary's keyrings satisfies the access policy. This means that for each j there is at least one $\mathfrak{v}$, for which he does not have a sufficient set $s^{(j)} + a_{i,t_i} \lambda_i^{(j)}$ to build the partial secret $m^{(\mathfrak{v})}$. If he uses such a wrong attribute set, he will not be able to cancel out the term $\sum_{\mathfrak{v}} \sum_i a_{i,t_i} \lambda_i^{(j)} r_i^{(\mathfrak{v})}$, as the corresponding ciphertext components are not $a_{i,t}^{(\mathfrak{v})} r_i^{(j)} \in P$ but were set by the encryptor to random values.

Without a sufficient set of partial secrets, he cannot build r and thus not $r s^{(j)} + \sum_{\mathfrak{v}} \sum_i a_{i,t_i} \lambda_i^{(j)} r_i^{(\mathfrak{v})}$ for any j.

As this is the only remaining possibility to remove the unneeded term $r s^{(j)}$ from the polynomial, it can now be seen that $f = \omega r$ is independent of (P, Q, R), and from Theorem 3 it follows that the adversary cannot build f and thus cannot break the system. □

D Indistinguishability within an Anonymity Set

We need to show that an attacker cannot distinguish between two policies that are in the same anonymity set. As shown in section 4.1, from an adversary's view the anonymity set is defined by the tree major and the set of nodes that he can decrypt. The latter set is in turn defined by his attribute set.

The game works as follows: The adversary decides on two policies W_0 and W_1 as well as a common syntax tree major R of both W_0 and W_1. Note that there is only a fixed number of ways that each policy can be embedded in R, as the process is defined by our algorithms given in Section 2. Thus, the adversary is able to determine all possible anonymity sets given an attribute list. For each of these anonymity sets he can further determine if exactly one of W_0 and W_1 is an element of the anonymity set. This can only be the case if one of the leaves of R can be decrypted for one of the two policies W_0 and W_1 and cannot be decrypted for the other one. As we only claim policy anonymity if either both or neither of W_0 and W_1 are in the anonymity set, we restrict the adversary to only query attribute lists that do not have this property.

Setup: The challenger runs the *Setup* algorithm and gives the public key PK to the adversary.

Challenge: The adversary commits to the challenge ciphertext policies W_0, W_1 as well as a common syntax tree major R of W_0 and W_1 and a message M. The challenger flips a random coin b and passes the ciphertext E *:= Encrypt(PK, M,* W_b*)* to the adversary.

Query: The adversary sends a number of attribute list $L_i, 0 \leq i \leq n$ for any polynomial n. For each attribute list L_i, the challenger verifies that either both W_0 and W_1 are in the anonymity set derived from L_i or that neither W_0 or W_1 are in that anonymity set. If this is the case, the challenger gives the adversary the secret key SK_{L_i}. Note that these queries can be adaptive.

Guess: The adversary outputs a guess b' of b.

The adversary wins the game if his advantage $\left|Pr[b = b'] - \frac{1}{2}\right|$ is negligible.

Theorem 4. *If Nishide's construction is secure, the adversary cannot win the game.*

Proof sketch As the leaves of the tree are encrypted using Nishide's construction (without the unblinding step), the adversary's views are identical for both $b = 0$ and $b = 1$. This is the case because by our restriction each leaf that can be decrypted if $b = 0$ can also be decrypted if $b = 1$ and each leaf that cannot be decrypted if $b = 0$ can also not be decrypted if $b = 1$. Thus he cannot distinguish between the two policies, proving the theorem. □