

Distributed Attribute-Based Encryption

Sascha Müller, Stefan Katzenbeisser, and Claudia Eckert

Technische Universität Darmstadt

Hochschulstr. 10

D – 64289 Darmstadt

{mueller,eckert}@sec.informatik.tu-darmstadt.de,

katzenbeisser@seceng.informatik.tu-darmstadt.de

Abstract. Ciphertext-Policy Attribute-Based Encryption (CP-ABE) allows to encrypt data under an access policy, specified as a logical combination of attributes. Such ciphertexts can be decrypted by anyone with a set of attributes that fits the policy. In this paper, we introduce the concept of Distributed Attribute-Based Encryption (DABE), where an arbitrary number of parties can be present to maintain attributes and their corresponding secret keys. This is in stark contrast to the classic CP-ABE schemes, where all secret keys are distributed by one central trusted party. We provide the first construction of a DABE scheme; the construction is very efficient, as it requires only a constant number of pairing operations during encryption and decryption.

1 Introduction

Emerging ubiquitous computing environments need flexible access control mechanisms. With a large and dynamic set of users, access rules for objects cannot easily be based on identities, and the conditions under which access to an object is granted need to take into account information like the context and the history of a subject. Due to these shortcomings of traditional access control mechanisms, cryptographically enforced access control receives increasing attention.

One of the most interesting approaches is Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [1]. In this scheme, users possess sets of attributes (and corresponding secret attribute keys) that describe certain properties. Ciphertexts are encrypted according to an access control policy, formulated as a Boolean formula over the attributes. The construction assures that only users whose attributes satisfy the access control policy are able to decrypt the ciphertext with their secret attribute keys. The construction is required to satisfy a *collusion-resistance* property: It must be impossible for several users to pool their attribute keys such that they are able to decrypt a ciphertext which they would not be able to decrypt individually.

Common to most previous ABE schemes is the existence of a central trusted authority (*master*) that knows a secret master key and distributes secret attribute keys to eligible users. However, for many attribute-based scenarios, it is much more natural to support multiple authorities [8,7]. The limitation to a

```

http://db.mycompany.org : isAdmin OR
http://db.mycompany.org : hasFullAccess OR
( http://www.openid.org : is18OrOlder AND
  ( http://www.contprov1.com : article1234.hasPaidFor OR
    http://www.contprov2.com : article4325.hasPaidFor OR
    http://www.contprov3.com : articleABC.hasPurchased ) )

```

Fig. 1. An example policy

single central authority for attribute generation is neither realistic nor desirable in applications where no single entity has the authority to grant secret keys for arbitrary attributes.

We can, for exemplary purposes, illustrate one such scenario as follows. Consider a company that hosts DRM protected media files. Users can purchase licenses from various content providers that issue usage licenses which contain the keys required to decrypt the protected files. Let us assume that three such content providers are `contprov1.com`, `contprov2.com`, and `contprov3.com`. The usage license (see Figure 1) can be expressed as a Boolean formula over attributes. Here, attributes consist of an URL that specifies a party who has authority over an attribute and an identifier describing the attribute itself, both represented as strings and concatenated with a single colon character as separator. The intuition behind this sample policy is that the protected file should only be decrypted by someone who either is an administrator of the company database `db.mycompany.org`, has the rights to download all files, or is at least 18 years old (which is established by an identification service `www.openid.org`) and has purchased licenses from at least one of the given content providers. Note that the same media file might be identified by different product codes in different providers' databases.

It is difficult to use this policy in a standard CP-ABE scheme, since there is no central authority who maintains and controls all attributes; in the above example, `www.contprov1.com` is solely responsible for maintaining the attribute `article1234.hasPaidFor`, while `db.mycompany.org` has authority over the attribute `isAdmin`. While it is possible that a third party is set up to which the maintenance of all attributes is delegated, this solution obviously does not scale. In addition, this solution is problematic if the entities mutually distrust each other.

1.1 Distributed ABE

We propose Distributed Attribute-Based Encryption (DABE) to mitigate this problem. DABE allows an arbitrary number of authorities to independently maintain attributes. There are three different types of entities in a DABE scheme: a master, attribute authorities and users.

The *master* is responsible for the distribution of secret user keys. However, in contrast to standard CP-ABE schemes, this party is *not* involved in the creation of secret attribute keys; the latter task can independently be performed by the attribute authorities.

Attribute authorities are responsible to verify whether a user is eligible of a specific attribute; in this case they distribute a secret attribute key to the user. (Note that determining the users' eligibility is application-dependent and thus beyond the scope of this work.) In our scheme every attribute is associated with a single attribute authority, but each attribute authority can be responsible for an arbitrary number of attributes. Every attribute authority has full control over the structure and semantics of its attributes. An attribute authority generates a *public attribute key* for each attribute it maintains; this public key is available to every participant. Eligible users receive a personalized secret attribute key over an authenticated and trusted channel. This secret key, which is personalized to prevent collusion attacks, is required to decrypt a ciphertext.

Users can encrypt and decrypt messages. To encrypt a message, a user first formulates his access policy in the form of a Boolean formula over some attributes, which in our construction is assumed to be in Disjunctive Normal Form (DNF). The party finally uses the public keys corresponding to the attributes occurring in the policy to encrypt. In DNF, all negations are atomic, so attribute authorities should be able to issue negative attributes as well in order to make use of the full expressive power of DNF formulas.

To decrypt a ciphertext, a user needs at least access to some set of attributes (and their associated secret keys) which satisfies the access policy. If he does not already possess these keys, he may query the attribute authorities for the secret keys corresponding to the attributes he is eligible of.

To illustrate the use of a DABE scheme, we return to the above mentioned example of the protection of media files. Figure 2 shows the policy of Figure 1 in DNF. The policy consists of five conjunctions over different sets of attributes. A user needs all secret attribute keys of at least one of the conjunctive terms to be able to decrypt a ciphertext that was encrypted with this access policy.

A user who downloads the ciphertext analyzes the policy and tests if he has a sufficient set of attributes to decrypt. The user may contact attribute authorities for secret attribute keys he does not already have in his possession but he is eligible of. For instance, he may query `www.openid.org` for a secret attribute key corresponding to `is180r0lder` and `contprov3.com` for a secret attribute key corresponding to the attribute `articleABC.hasPurchased`. In this case he is able to satisfy the last conjunction. It may be necessary for him to perform additional steps if he is not yet eligible for an attribute. For example, he might decide to buy the article `articleABC` from `contprov3.com` to get the respective attribute.

Note that every attribute authority independently decides on the structure and semantics of its attributes. For instance, the authority `db.mycompany.org` offers the attribute `isAdmin`. The meaning of this attribute and the semantics (i.e., the decision who is eligible of it) is entirely up to `db.mycompany.org`. Whoever includes the attribute in an access policy needs to trust the respective authority to correctly determine eligibility.

Note that a DABE scheme must be collusion-resistant: if a user u has a friend v who possesses an attribute that u does not have, it is not possible for u to

```

http://db.mycompany.org : isAdmin
OR
http://db.mycompany.org : hasFullAccess
OR
( http://www.openid.org : is180rOlder AND
  http://www.contprov1.com : article1234.hasPaidFor )
OR
( http://www.openid.org : is180rOlder AND
  http://www.contprov2.com : article4325.hasPaidFor )
OR
( http://www.openid.org : is180rOlder AND
  http://www.contprov3.com : articleABC.hasPurchased )

```

Fig. 2. Policy of Figure 1 in DNF

use the corresponding secret attribute key of v . Neither can u give any of his attribute keys to v . All secret attribute keys are bound to their owner, making them unusable with keys issued for other users.

1.2 Our Contribution

In this paper we introduce the concept of Distributed Attribute-Based Encryption (DABE), i.e., a fully distributed version of CP-ABE, where multiple attribute authorities may be present and distribute secret attribute keys. Furthermore, we give the first construction of a DABE scheme, which supports policies written in DNF; the ciphertexts grow linearly with the number of conjunctive terms in the policy. Our scheme is very simple and efficient, demonstrating the practical viability of DABE. We furthermore provide a proof of security in the generic group model, introduced by [2]; even though this proof is weaker than the proofs of some more recent CP-ABE schemes [3,4,5], our scheme is much more efficient, requiring only $O(1)$ pairing operations during encryption and decryption.

The remainder of this document is structured as follows: In Section 2 we discuss related work. Section 3 contains a description of DABE as well as a formal definition of the required security property. Our construction is detailed in Section 4. We discuss its security and performance in Section 5. Finally we conclude in Section 6. A detailed security proof in the generic bilinear group model is given in the appendix.

2 Related Work

Attribute-Based Encryption was first proposed by Goyal et al. [6] in the form of *key-policy* attribute-based encryption (KP-ABE), based on the work of Sahai and Waters [7]. In KP-ABE, users are associated with access policies and ciphertexts are encrypted with sets of attributes. The access policies describe which ciphertexts users can decrypt.

The first CP-ABE scheme was presented by Bethencourt, Sahai and Waters [1], followed by some cryptographically stronger CP-ABE constructions that allowed reductions to the Decisional Bilinear Diffie Hellman Problem [5,4], but imposed restrictions that the original CP-ABE does not have. Recently, Waters proposed three CP-ABE schemes that are as expressive as [1], rather efficient and provably secure under strong cryptographic assumptions [3].

There is only one attempt at multi-authority CP-ABE, proposed by Chase [8] as an extension of her multi-authority threshold ABE construction. This extension is rather limited. The policy is written as a set of threshold gates, which are connected by another outer threshold gate. The threshold of the outer gate is fixed. (However, one could run several parallel instances to support different thresholds.) Each of the inner threshold gates is controlled by one of the authorities, and contains only attributes of that authority. The threshold of each inner gate is fixed, even though dummy attributes can be used to support different thresholds, as described in [7]. If the policy can only be formulated in a way where some attributes occur in more than one of the inner threshold gates, these attributes must be copied between the respective authorities, so in this case the involved authorities need to mutually trust each other.

Another restriction of Chase's scheme is that all authorities are managed centrally by a trusted authority. Whenever a new authority is added, the global system key changes and has to be propagated to all users who want to use attributes of the authority. This includes encryptors who want to use attributes from that authority in the policy.

Techniques similar to CP-ABE were proposed for many applications like Attribute-Based Access Control (ABAC, used in SOA) [9], Property-Based Broadcast Encryption (used in DRM) [10], and Hidden Credentials [11,12]. Note that the techniques used in these applications are not collusion-resistant, so they can not be classified as ABE. It remains to be examined if ABE techniques can be used to improve the solutions.

3 DABE

In this section we formally define the concept of DABE and introduce the required keys and algorithms. Our construction will be detailed in Section 4. Table 1 on the next page provides a quick reference of the most relevant keys.

3.1 Users, Attributes and Keys

During setup, a public master key PK and a secret master key MK are generated; PK is available to every party, whereas MK is only known to the master. Every user u maintains a public user key PK_u , which is used by attribute authorities to generate personalized secret attribute keys, and a secret key SK_u , which is used in the decryption operation. Generation and distribution of PK_u and SK_u is the task of the master, who is also required to verify the identity of the users before keys are issued. The keys SK_u and PK_u of a user u are bound to the

Table 1. Summary of DABE keys

Key	Description	Usage
PK	Global key	Input for all operations
MK	Master key	Creation of user keys
SK_a	Secret key of attribute authority a	Creation of attribute keys
$PK_{\mathcal{A}}$	Public key of attribute \mathcal{A}	Encryption
$SK_{\mathcal{A},u}$	Secret key of attribute \mathcal{A} for user u	Decryption
PK_u	Public key of user u	Key Request
SK_u	Secret key of user u	Decryption

identity and/or pseudonyms of the user by the master. This binding is crucial for the verification of the user’s attributes.

Every attribute authority maintains a secret key SK_a which is used to issue secret attribute keys to users. An attribute is a tuple consisting of an identifier of an attribute authority (e.g. an URL) and an identifier describing the attribute itself (an arbitrary string). We will denote the public representation of the attribute as \mathcal{A} and use $a_{\mathcal{A}}$ as the identifier of the attribute authority present within \mathcal{A} . For every attribute with representation \mathcal{A} there is a public key, denoted $PK_{\mathcal{A}}$, which is issued by the respective attribute authority and is used to encrypt messages. The corresponding secret attribute keys, personalized for eligible users, are issued by the attribute authorities to users who request them (after determining their eligibility). To prevent collusions, every user gets a different secret attribute key that only he can use. A secret attribute key of an attribute \mathcal{A} , issued for a user u is denoted by $SK_{\mathcal{A},u}$. We call the set of secret keys that a user has (i.e., the key SK_u and all keys $SK_{\mathcal{A},u}$) his *key ring*.

3.2 The DABE Scheme

The DABE scheme consists of seven fundamental algorithms: *Setup*, *CreateUser*, *CreateAuthority*, *RequestAttributePK*, *RequestAttributeSK*, *Encrypt* and *Decrypt*. The description of the seven algorithms is as follows:

Setup. The *Setup* algorithm takes as input the implicit security parameter 1^k .

It outputs the public key PK and the master key MK.

CreateUser(PK, MK, u). The *CreateUser* algorithm takes as input the public key PK, the master key MK, and a user name u . It outputs a public user key PK_u , that will be used by attribute authorities to issue secret attribute keys for u , and a secret user key SK_u , used for the decryption of ciphertexts.

CreateAuthority(PK, a). The *CreateAuthority* algorithm is executed by the attribute authority with identifier a once during initialization. It outputs a secret authority key SK_a .

RequestAttributePK(PK, \mathcal{A} , SK_a). The *RequestAttributePK* algorithm is executed by attribute authorities whenever they receive a request for a public attribute key. The algorithm checks whether the authority identifier $a_{\mathcal{A}}$

of \mathcal{A} equals a . If this is the case, the algorithm outputs a public attribute key for attribute \mathcal{A} , denoted $\text{PK}_{\mathcal{A}}$, otherwise NULL.

RequestAttributeSK($\text{PK}, \mathcal{A}, \text{SK}_a, u, \text{PK}_u$). The *RequestAttributeSK* algorithm is executed by the attribute authority with identifier a whenever it receives a request for a secret attribute key. The algorithm checks whether the authority identifier $a_{\mathcal{A}}$ of \mathcal{A} equals a and whether the user u with public key PK_u is eligible of the attribute \mathcal{A} . If this is the case, *RequestAttributeSK* outputs a secret attribute key $\text{SK}_{\mathcal{A},u}$ for user u . Otherwise, the algorithm outputs NULL.

Encrypt($\text{PK}, M, \mathbb{A}, \text{PK}_{\mathcal{A}_1}, \dots, \text{PK}_{\mathcal{A}_N}$). The *Encrypt* algorithm takes as input the public key PK , a message M , an access policy \mathbb{A} and the public keys $\text{PK}_{\mathcal{A}_1}, \dots, \text{PK}_{\mathcal{A}_N}$ corresponding to all attributes occurring in the policy \mathbb{A} . The algorithm encrypts M with \mathbb{A} and outputs the ciphertext CT .

Decrypt($\text{PK}, \text{CT}, \mathbb{A}, \text{SK}_u, \text{SK}_{\mathcal{A}_1,u}, \dots, \text{SK}_{\mathcal{A}_N,u}$). The *Decrypt* algorithm takes as input a ciphertext produced by the *Encrypt* algorithm, an access policy \mathbb{A} , under which CT was encrypted, and a key ring $\text{SK}_u, \text{SK}_{\mathcal{A}_1,u}, \dots, \text{SK}_{\mathcal{A}_N,u}$ for user u . The algorithm *Decrypt* decrypts the ciphertext CT and outputs the corresponding plaintext M if the attributes were sufficient to satisfy \mathbb{A} ; otherwise it outputs NULL.

Note that this scheme differs from CP-ABE [1] in that the two algorithms *CreateAuthority* and *RequestAttributePK* were added, and CP-ABE's algorithm *KeyGen* is split up into *CreateUser* and *RequestAttributeSK*. It is crucial that *RequestAttributeSK* does not need any components of the master key MK as input, so that every attribute authority is able to independently create attributes. However, we still require that a trusted central party maintains users (executes *CreateUser*), as otherwise collusion attacks would be possible.

3.3 Security Model

We model the security of DABE in terms of a game between a challenger and an adversary, where the challenger plays the role of the master and all attribute authorities.

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

Phase 1. The adversary asks the challenger for an arbitrary number of user keys. The challenger calls *CreateUser* for each requested user and returns the resulting public and private user keys to the adversary. For each user the adversary can request an arbitrary number of secret and public attribute keys, that the challenger creates by calling *RequestAttributeSK* or *RequestAttributePK*, respectively. Whenever the challenger receives a request for an attribute \mathcal{A} of authority a , he tests whether he has already created a secret key SK_a for a . If not, he first calls *CreateAuthority* to create the appropriate authority key (note that SK_a will not be made available to the adversary).

Challenge. The adversary submits two messages M_0 and M_1 and an access policy \mathbb{A} such that none of the users that he created in Phase 1 satisfy \mathbb{A} . (If any user from Phase 1 satisfies \mathbb{A} , the challenger aborts.) As before, the challenger may have to call *CreateAuthority* to initialize attribute authorities. The challenger flips a coin b , encrypts M_b under \mathbb{A} , and gives the ciphertext CT to the adversary.

Phase 2. Like in Phase 1, the adversary may create an arbitrary number of users. He can also request more secret attribute keys for the users he created in Phase 1 and 2, but if any secret attribute key would give the respective user a set of attributes needed to satisfy \mathbb{A} , the challenger aborts. As before, the adversary can always request any public attribute key.

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

The advantage of the adversary in this game is defined as $\Pr[b' = b] - \frac{1}{2}$, where the probability is taken over all coin tosses of both challenger and adversary. A DABE scheme is secure if all polynomial time adversaries have at most a negligible advantage in the above game.

4 Our Construction

We construct an efficient DABE scheme as follows:

Setup. The *Setup* algorithm chooses a bilinear group \mathbb{G} of order p and a pairing $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ [13]. Next it chooses a generator $g \in \mathbb{G}$, a random point $P \in \mathbb{G}$ and a random exponent $y \in \mathbb{Z}_p$. The public key of the system is $\text{PK} = \{\mathbb{G}, \mathbb{G}_T, e, g, P, e(g, g)^y\}$, while the secret master key is given by $\text{MK} = g^y$.

CreateUser(PK, MK, u). The algorithm *CreateUser* chooses a secret $\text{mk}_u \in \mathbb{Z}_p$ and outputs the public key $\text{PK}_u := g^{\text{mk}_u}$ and the private key $\text{SK}_u := \text{MK} \cdot P^{\text{mk}_u} = g^y \cdot P^{\text{mk}_u}$ for user u .

CreateAuthority(PK, a). The algorithm *CreateAuthority* chooses uniformly and randomly a hash function $H_{x_a} : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ from a finite family of hash functions, which we model as random oracles. It returns as secret key the index of the hash function $\text{SK}_a := x_a$.

RequestAttributePK($\text{PK}, \mathcal{A}, \text{SK}_a$). If \mathcal{A} is handled by the attribute authority a (i.e., $a_{\mathcal{A}} = a$), *RequestAttributePK* returns the public attribute key of \mathcal{A} , which consists of two parts:

$$\text{PK}_{\mathcal{A}} := \langle \text{PK}'_{\mathcal{A}} := g^{H_{\text{SK}_a}(\mathcal{A})}, \quad \text{PK}''_{\mathcal{A}} := e(g, g)^{y H_{\text{SK}_a}(\mathcal{A})} \rangle .$$

This public key can be requested from the attribute authority by anyone, but *RequestAttributePK* can only be executed by the respective authority, because it requires the index of the hash function SK_a as input.

RequestAttributeSK($\text{PK}, \mathcal{A}, \text{SK}_a, u, \text{PK}_u$). After determining that the attribute \mathcal{A} is handled by a (i.e., $a_{\mathcal{A}} = a$), the authority tests whether user u is eligible for the attribute \mathcal{A} . If this is not the case, *RequestAttributeSK* returns NULL, else it outputs the secret attribute key

$$\text{SK}_{\mathcal{A},u} := \text{PK}_u^{H_{\text{SK}_a}(\mathcal{A})} = g^{\text{mk}_u H_{\text{SK}_a}(\mathcal{A})}.$$

Note that the recipient u can check the validity of this secret key by testing if

$$e(\text{SK}_u, \text{PK}'_{\mathcal{A}}) = \text{PK}''_{\mathcal{A}} \cdot e(P, \text{SK}_{\mathcal{A},u}) .$$

Encrypt($\text{PK}, M, \mathbb{A}, \text{PK}_{\mathcal{A}_1}, \dots, \text{PK}_{\mathcal{A}_N}$). A policy in DNF can be written as

$$\mathbb{A} = \bigvee_{j=1}^n \left(\bigwedge_{\mathcal{A} \in S_j} \mathcal{A} \right),$$

where n (not pairwise disjoint) sets S_1, \dots, S_n denote attributes that occur in the j -th conjunction of \mathbb{A} . The encryption algorithm iterates over all $j = 1, \dots, n$, generates for each conjunction a random value $R_j \in \mathbb{Z}_p$ and constructs CT_j as

$$\begin{aligned} \text{CT}_j &:= \langle E_j := M \cdot \left(\prod_{\mathcal{A} \in S_j} \text{PK}''_{\mathcal{A}} \right)^{R_j}, \\ E'_j &:= P^{R_j}, \\ E''_j &:= \left(\prod_{\mathcal{A} \in S_j} \text{PK}'_{\mathcal{A}} \right)^{R_j} \rangle . \end{aligned} \quad (1)$$

The ciphertext CT is obtained as tuple $\text{CT} := \langle \text{CT}_1, \dots, \text{CT}_n \rangle$.

Decrypt($\text{PK}, \text{CT}, \mathbb{A}, \text{SK}_u, \text{SK}_{\mathcal{A}_1,u}, \dots, \text{SK}_{\mathcal{A}_N,u}$). To decrypt a ciphertext CT, *Decrypt* first checks whether any conjunction of \mathbb{A} can be satisfied by the given attributes, i.e., whether the input $\text{SK}_{\mathcal{A}_1,u}, \dots, \text{SK}_{\mathcal{A}_N,u}$ contains secret keys for all attributes occurring in a set S_j for some $1 \leq j \leq n$. If this is not the case, the algorithm outputs NULL, otherwise

$$M = E_j \cdot \frac{e(E'_j, \prod_{i \in S_j} \text{SK}_{i,u})}{e(E''_j, \text{SK}_u)} .$$

It is easy to see that the decryption is correct. Let $a_j := \sum_{\mathcal{A} \in S_j} H_{\text{SK}_a}(\mathcal{A})$. Then $E_j = M \cdot e(g, g)^{y a_j R_j}$, $E'_j = g^{a_j R_j}$ and

$$\begin{aligned} E_j \cdot \frac{e(E'_j, \prod_{i \in S_j} \text{SK}_{i,u})}{e(E''_j, \text{SK}_u)} &= M \cdot e(g, g)^{y a_j R_j} \cdot \frac{e(P^{R_j}, g^{\text{mk}_u a_j})}{e(g^{a_j R_j}, g^y \cdot P^{\text{mk}_u})} \\ &= M \cdot e(g, g)^{y a_j R_j} \cdot \frac{e(P, g)^{R_j \text{mk}_u a_j}}{e(P, g)^{R_j \text{mk}_u a_j} \cdot e(g, g)^{y a_j R_j}} = M . \end{aligned}$$

5 Discussion

In this section, we first comment on the performance of the proposed DABE scheme, give a security proof in the generic group model and finally comment on the delegation property.

5.1 Performance

Compared to other ABE schemes, the proposed DABE construction is very efficient. Nearly all operations are group operations in \mathbb{G} and \mathbb{G}_T . The only computationally expensive operation—the pairing $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ —is needed during decryption exactly two times, no matter how complex the access policy is. No pairings are needed for any other algorithms. In all other known ABE schemes, the number of pairings grows at least linearly with the minimum number of distinct attributes needed for decryption.

5.2 Security

We first give an intuitive security argument. Clearly, to decrypt a ciphertext CT without having access to a sufficient set of secret attribute keys, an adversary needs to find $e(g, g)^{y a_j R_j}$ for some $1 \leq j \leq n$ which allows him to obtain M from E_j (note that M only occurs in E_j). He thus must compute a pairing of g^α and $g^{y\beta}$ for some $\alpha, \beta \in \mathbb{Z}_p$, such that $\alpha\beta = a_j R_j$.

To create such a pairing, the adversary can only use keys that he has obtained before in a security game as defined in Section 3.3. We will show that, assuming the adversary has not enough secret keys to satisfy \mathbb{A} , he is not able to compute this value.

The only occurrence of g^y (aside from $e(g, g)^y$) is in the secret user keys, so the adversary has to use some SK_u for the pairing, yielding

$$e(g^\alpha, \gamma \text{SK}_u) = e(g^\alpha, g^y) \cdot e(g^\alpha, P^{\text{mk}_u}) \cdot e(g^\alpha, \gamma) ,$$

for some γ . Given all values that the adversary knows, the only useful choice for g^α is $E_j'' = g^{a_j R_j}$ for some conjunction $\bigwedge_{\mathcal{A} \in S_j} \mathcal{A}$. Pairing E_j'' with SK_u gives:

$$e(E_j'', \text{SK}_u) = e(g, g)^{y a_j R_j} \cdot e(g, P)^{\text{mk}_u a_j R_j} .$$

To obtain $e(g, g)^{y a_j R_j}$, the second factor has to be eliminated. However, all three exponents of $e(g, P)^{\text{mk}_u a_j R_j}$ are unknown to the adversary, and no combination of two publicly known values or secret user keys holds exactly the desired components (assuming that the adversary does not have all required secret attribute keys), so this value cannot be computed by the adversary.

For a more thorough security proof of our construction, we will use the generic group model [2]. In this model, the elements of \mathbb{G} and \mathbb{G}_T are encoded as arbitrary strings that (from the adversary's point of view) appear random. All operations are computed by oracles, so the adversary is limited to the group operations, the

pairing, and equality tests. A scheme proven secure this way is called *generically secure* and can only be broken by exploiting specific properties of the groups that are used to implement it.

Theorem 1. *Let Adv be a generic adversary who plays the DABE security game and makes q oracle queries. Then Adv has advantage at most $O(q^2/p)$ in the generic group model, where p is the order of the bilinear group.*

A proof of this theorem is given in the appendix.

5.3 Delegation

The CP-ABE schemes [1] and [4] support an additional mechanism called *Delegate* that allows a user to create a new key ring that contains a subset of his secret attribute keys. In a DABE scenario with separate *CreateUser* and *RequestAttributeSK* algorithms, delegation between users cannot be supported since it allows collusions. To see this, consider a user u who is eligible of a set of attributes S_u and a user v who is eligible of a set of attributes $S_v \neq S_u$. Now let S be a set of attributes such that $S \subset S_u \cup S_v$, but $S_u \not\subseteq S$ and $S_v \not\subseteq S$. To decrypt a ciphertext encrypted with a conjunction consisting of all attributes in S , the user u would use *CreateUser* and *RequestAttributeSK* to get all attributes of S_u , then call *Delegate* to create a key ring for v that contains $S \cap S_u$. Finally, v would then use *RequestAttributeSK* to add all remaining attributes $S'_v := S_v \setminus (S \cap S_u)$. This is possible as in a DABE scheme, the *RequestAttributeSK* can be called at any time to add private attribute keys to key rings. Subsequently, v could decrypt any ciphertext encrypted with S . For this reason, delegation is not allowed in DABE. In our scheme, key rings can be re-randomized in the same way as [1] and [3], so a new keyring containing a subset of the attributes of the old keyring can be generated that is usable for decryption. However, all values of the resulting keyring will contain a random mk_u that is not bound to any identity, so the user will not be able to add new attributes to it.

6 Conclusion

In this paper, we proposed the concept of Distributed Attribute-Based Encryption (DABE) as an extension of Ciphertext-Policy Attribute-Based Encryption (CP-ABE) that supports an arbitrary number of attribute authorities and allows to dynamically add new users and authorities at any time. We provided an efficient construction of DABE that uses only two pairing operations in the decryption algorithm and no pairing operation in any other algorithm.

A limitation of our construction is that access policies need to be in DNF form. We leave it as an open question to design a more expressive DABE scheme, while preferably maintaining the $O(1)$ number of pairings that our construction offers.

Acknowledgements

The authors wish to thank the reviewers of this paper on the ICISC 2008 program committee for some very helpful comments and suggestions.

References

1. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: IEEE Symposium on Security and Privacy, pp. 321–334. IEEE Computer Society, Los Alamitos (2007)
2. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (1997)
3. Waters, B.: Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. Technical report, SRI International (2008) (to appear)
4. Goyal, V., Jain, A., Pandey, O., Sahai, A.: Bounded ciphertext policy attribute based encryption. In: ICALP (2008)
5. Cheung, L., Newport, C.C.: Provably secure ciphertext policy ABE. In: Ning, P., di Vimercati, S.D.C., Syverson, P.F. (eds.) ACM Conference on Computer and Communications Security, pp. 456–465. ACM, New York (2007)
6. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Juels, A., Wright, R.N., di Vimercati, S.D.C. (eds.) ACM Conference on Computer and Communications Security, pp. 89–98. ACM, New York (2006)
7. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005)
8. Chase, M.: Multi-authority attribute based encryption. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 515–534. Springer, Heidelberg (2007)
9. Yuan, E., Tong, J.: Attributed based access control (ABAC) for web services. In: ICWS, pp. 561–569. IEEE Computer Society, Los Alamitos (2005)
10. Adelsbach, A., Huber, U., Sadeghi, A.R.: Property-based broadcast encryption for multi-level security policies. In: Won, D.H., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 15–31. Springer, Heidelberg (2006)
11. Kapadia, A., Tsang, P.P., Smith, S.W.: Attribute-based publishing with hidden credentials and hidden policies. In: Proceedings of The 14th Annual Network and Distributed System Security Symposium (NDSS), pp. 179–192 (March 2007)
12. Bradshaw, R.W., Holt, J.E., Seamons, K.E.: Concealing complex policies with hidden credentials. In: Atluri, V., Pfitzmann, B., McDaniel, P.D. (eds.) ACM Conference on Computer and Communications Security, pp. 146–157. ACM, New York (2004)
13. Boneh, D.: A brief look at pairings based cryptography. In: FOCS, pp. 19–26. IEEE Computer Society, Los Alamitos (2007)
14. Boneh, D., Boyen, X.: Short signatures without random oracles and the SDH assumption in bilinear groups. *J. Cryptology* 21(2), 149–177 (2008)

A Security Proof

We closely follow the structure of the proof of [1]: First we show how to reduce any adversary who plays the DABE game of Section 3.3 (denoted Adv_1 in the

following) to an adversary in a modified game (denoted Adv_2). Then we show that no Adv_2 has non-negligible advantage, so there can be no Adv_1 with non-negligible advantage, either.

Let adversary Adv_1 be an adversary who plays the DABE game defined in Section 3.3 using our construction from Section 4. We define a modified game as follows: The phases **Setup**, **Phase 1**, and **Phase 2** are equal to the DABE game. In the **Challenge** phase, the adversary submits an access policy \mathbb{A} such that none of the users that he created in Phase 1 satisfy \mathbb{A} . The challenger flips a coin b , and creates a ciphertext for the access policy \mathbb{A} according to Equation 1, but instead of computing $E_j := M \cdot e(g, g)^{y_{a_j} R_j}$, he computes E_j as

$$E_j = \begin{cases} e(g, g)^{y_{a_j} R_j}, & \text{if } b = 1 \\ e(g, g)^{\theta_j}, & \text{if } b = 0 \end{cases},$$

where all θ_j are uniformly and independently chosen random elements of \mathbb{Z}_p .

Given an adversary Adv_1 that has advantage ϵ in the DABE game, we can construct Adv_2 as follows: In the phases **Setup**, **Phase 1**, and **Phase 2**, Adv_2 forwards all messages he receives from Adv_1 to the challenger and all messages from the challenger to Adv_1 . In the **Challenge** phase, Adv_2 receives two messages M_0 and M_1 from Adv_1 and the challenge C (which is either $e(g, g)^{y_{a_j} R_j}$ or $e(g, g)^{\theta_j}$) from the challenger. He flips a coin β and sends $C' := M_\beta \cdot C$ to Adv_1 . When Adv_1 outputs a guess β' , Adv_2 outputs as its guess 1 if $\beta' = \beta$, or 0 if $\beta' \neq \beta$. If $C = e(g, g)^{y_{a_j} R_j}$, then Adv_2 's challenge is a well-formed DABE ciphertext and Adv_1 has advantage ϵ of guessing the correct $\beta' = \beta$. If $C = e(g, g)^{\theta_j}$, the challenge is independent of the messages M_0 and M_1 , so the advantage of Adv_2 is 0. Thus, we have

$$\begin{aligned} \Pr[\text{Adv}_2 \text{ succeeds}] &= \Pr[C = e(g, g)^{y_{a_j} R_j}] \Pr[\beta' = \beta \mid C = e(g, g)^{y_{a_j} R_j}] + \\ &\quad \Pr[C = e(g, g)^{\theta_j}] \Pr[\beta' \neq \beta \mid C = e(g, g)^{\theta_j}] \\ &\leq \frac{1}{2} \left(\frac{1}{2} + \epsilon \right) + \frac{1}{2} \cdot \frac{1}{2} = \frac{1 + \epsilon}{2} \end{aligned}$$

and the overall advantage of Adv_2 is $\frac{\epsilon}{2}$, as required. The existence of any successful Adv_1 implies the existence of an adversary Adv_2 who succeeds with non-negligible advantage as well.

Our next step is to show that no Adv_2 can distinguish between $e(g, g)^{y_{a_j} R_j}$ and $e(g, g)^{\theta_j}$ in polynomial time. A combination of both results implies that no Adv_1 can have non-negligible advantage, either.

To show this, we use the generic group model from [2], with the extensions for bilinear groups with a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ developed in [14], which we simplify slightly for our case where $\mathbb{G}_1 = \mathbb{G}_2$. For groups \mathbb{G} and \mathbb{G}_T of prime order p , the simulator chooses an arbitrary generator $\tilde{g} \in \mathbb{G}$ and uses random maps $\xi, \xi_T : \mathbb{Z}_p \rightarrow \{0, 1\}^{\lceil \log p \rceil}$ that encode any \tilde{g}^x or $e(\tilde{g}, \tilde{g})^x$ as a random string. The maps ξ and ξ_T must be invertible, so the simulator can map representations back to elements of \mathbb{G} and \mathbb{G}_T . The simulator gives the adversary two oracles that compute the group operations of \mathbb{G} and \mathbb{G}_T and another oracle that computes the

pairing e . All oracles take as input string representations of group elements. The adversary can only perform operations on group elements by interacting with the oracles. For example, given two string representations $A := \xi_T(a)$ and $B := \xi_T(b)$ of elements of group \mathbb{G}_T , the adversary can query the group oracle for the result of the group operation $A \cdot B$ in \mathbb{G}_T . The simulator will map A and B back to the respective elements of \mathbb{G}_T using ξ_T^{-1} , then execute the group operation and map the result to a string using ξ_T . From the view of the adversary, the simulator can simply return $\xi(a) \cdot \xi(b) = \xi(a + b)$ or $\xi_T(a) \cdot \xi_T(b) = \xi_T(a + b)$ for multiplication and $\xi(a)/\xi(b) = \xi(a - b)$ or $\xi_T(a)/\xi_T(b) = \xi_T(a - b)$ for division. Note that no oracle will accept input from different encodings (for example, one cannot compute $\xi(a) \cdot \xi_T(b)$). The oracle for e can be implemented easily: Given two encodings $A = \xi(a)$ and $B = \xi(b)$ for some $a, b \in \mathbb{Z}_p$, the encoding of the pairing of A and B is $\xi_T(ab)$. We assume that the adversary only makes oracle queries on strings previously obtained from the simulator.

The simulator plays the DABE game as follows:

Setup. The simulator chooses \mathbb{G} , \mathbb{G}_T , e , \tilde{g} and random exponents $y, \tilde{p} \in \mathbb{Z}_p$, as well as two encoding functions ξ, ξ_T and oracles for the group operations in \mathbb{G} , \mathbb{G}_T , and the pairing as described above. The public parameters are $g := \xi(1)$, $P := \xi(\tilde{p})$, and $Y := \xi_T(y)$.

Phase 1. When the adversary calls *CreateUser* for some u , the simulator chooses a random $\text{mk}_u \in \mathbb{Z}_p$ and returns $\text{PK}_u := \xi(\text{mk}_u)$ and $\text{SK}_u := \xi(y + \tilde{p} \cdot \text{mk}_u)$. Whenever the simulator gets a request involving an attribute \mathcal{A} that the adversary has not used before, he chooses a new, unique random value $\text{mk}_{\mathcal{A}}$, which simulates the term $H_{\text{SK}_u}(\mathcal{A})$ of an attribute \mathcal{A} with attribute authority $a = a_{\mathcal{A}}$. (The association between values $\text{mk}_{\mathcal{A}}$ and attributes \mathcal{A} is stored for future queries). For every public attribute key request for an attribute \mathcal{A} , the simulator returns $\text{PK}'_{\mathcal{A}} := \xi(\text{mk}_{\mathcal{A}})$ and $\text{PK}''_{\mathcal{A}} := \xi_T(y \text{mk}_{\mathcal{A}})$. If queried for a secret attribute key, the simulator returns $\text{SK}_{\mathcal{A},u} = \xi(\text{mk}_u \text{mk}_{\mathcal{A}})$.

Challenge. When the adversary asks for a challenge, the simulator flips a coin b . Then he chooses a random $R_j \in \mathbb{Z}_p$ for each conjunction \mathbb{A}_j and computes $a_j = \sum_{\mathcal{A} \in \mathcal{S}_j} \text{mk}_{\mathcal{A}}$. If $b = 0$, he sets θ_j to a random value from \mathbb{Z}_p , otherwise $\theta_j := y a_j R_j$. Finally he computes the ciphertext components of CT_j as

$$\langle E_j := \xi_T(\theta_j), E'_j := \xi(\tilde{p} R_j), E''_j := \xi(a_j R_j) \rangle ,$$

which he returns as ciphertext.

Phase 2. The simulator behaves as in Phase 1. However, the simulator refuses any secret attribute key that would give the respective user a set of attributes satisfying \mathbb{A} .

All values that the adversary knows are either encodings of random values of \mathbb{Z}_p (namely $1, \tilde{p}, y, \text{mk}_u, \text{mk}_{\mathcal{A}}$ and θ), combinations of these values given to him by the simulator (for example $\text{SK}_{\mathcal{A},u} = \xi(\text{mk}_u \text{mk}_{\mathcal{A}})$), or results of oracle queries on combinations of these values. We keep track of the algebraic expressions used to query the oracles; all queries can thus be written as rational functions. We

assume that different terms always result in different string representations. This assumption can only be false if due to the choice of the random encodings two different terms “accidentally” result in the same string representation. Similar to the proof in [1] it can be shown that the probability of this event is $O(q^2/p)$ where q is the number of oracle queries that the adversary makes. In the following we will condition that no such event occurs.

Now, under this assumption consider how the adversary’s views differ between the case where the θ_j are random ($b = 0$) and the case where $\theta_j = ya_jR_j$ ($b = 1$). We claim that the views are identically distributed for both cases and therefore any adversary has no advantage to distinguish between them in the generic group model. To proof this claim, assume the opposite. Since the adversary can only test for equality of string representations he receives (and all representations of group elements are random), the only possibility for the views to differ is that there exist two different terms that result in the same answer in the view where $\theta_j = ya_jR_j$ ($b = 1$), for at least one j , and in different answers in the view corresponding to $b = 0$. Call two such terms ν_1 and ν_2 and fix one relevant j . Since θ_j only occurs as $E_j := \xi_T(\theta_j)$ and elements of ξ_T cannot be paired, the adversary can only construct queries where θ_j appears as an additive term. Thus, ν_1 and ν_2 can be written as

$$\begin{aligned}\nu_1 &= \gamma_1\theta_j + \nu'_1 \\ \nu_2 &= \gamma_2\theta_j + \nu'_2 \ ,\end{aligned}$$

for some ν'_1 and ν'_2 that do not contain θ_j . Since by assumption $\theta_j = ya_jR_j$ results in $\nu_1 = \nu_2$, we have $\gamma_1ya_jR_j + \nu'_1 = \gamma_2ya_jR_j + \nu'_2$. Rearranging the equation yields

$$\nu'_1 - \nu'_2 = (\gamma_2 - \gamma_1)ya_jR_j \ .$$

Thus, the adversary can construct an oracle query for a term γya_jR_j (which we can, without loss of generality, add to the queries of the adversary).

It remains to be shown that, without having a sufficient set of attributes satisfying \mathbb{A} , the adversary *cannot* construct a query of the form $\xi_T(\gamma ya_jR_j)$ for any γ and j from the information that he has. This contradicts the assumption that the views in the modified game are not identically distributed.

After Phase 2, the adversary has received the following information from the simulator:

- The tuple PK.
- PK_u and SK_u for an arbitrary number of users.
- $\text{PK}'_{\mathcal{A}}$ and $\text{PK}''_{\mathcal{A}}$ for an arbitrary number of attributes.
- $\text{SK}_{\mathcal{A},u}$ for an arbitrary number of attributes and users, with the restriction that for no u , he has a sufficient set of secret attributes keys that satisfies \mathbb{A} .
- E_j , E'_j , and E''_j of the challenge ciphertext.

Furthermore, he possibly obtained encodings of arbitrary combinations of these values through queries to the five oracles that implement the group operations and the pairing. Since all R_j and y are random, the only way to construct

Table 2. Results of pairings

Source	Term	Pairing with SK_u	Pairing with E'_j
$PK_{u'}$	$mk_{u'}$	$mk_{u'} y + \tilde{p} mk_u mk_{u'}$	$mk_u \tilde{p} R_j$
$SK_{u'}$	$y + \tilde{p} mk_{u'}$	$y^2 + y\tilde{p}(mk_{u'} + mk_u) + \tilde{p}^2 mk_u mk_{u'}$	$y\tilde{p} R_j + \tilde{p}^2 mk_u R_j$
$\prod_{\mathcal{A} \in S_j} PK'_{\mathcal{A}}$	a_j	$ya_j + \tilde{p} mk_u a_j$	$a_j \tilde{p} R_j$
E'_j	$\tilde{p} R_j$	$y\tilde{p} R_j + \tilde{p}^2 mk_u R_j$	$\tilde{p}^2 R_j^2$
E''_j	$a_j R_j$	$ya_j R_j + \tilde{p} mk_u a_j R_j$	$a_j \tilde{p} R_j^2$

$\xi_T(\gamma ya_j R_j)$ is to pair two values from \mathbb{G} by querying the pairing oracle, so that each of the components is contained in any of the terms.

First we show how the adversary can find representations of terms that contain a_j . Aside from E_j and E''_j , a_j can only be constructed by querying the multiplication oracle for encodings of the terms containing $mk_{\mathcal{A}}$ for all $\mathcal{A} \in S_j$ and some j with $1 \leq j \leq n$. These values occur only in $PK'_{\mathcal{A}}$, $PK''_{\mathcal{A}}$, and $SK_{\mathcal{A},u}$. Since $PK''_{\mathcal{A}} \in \mathbb{G}_T$, it cannot be used as input of the pairing. Multiplying representations of $PK'_{\mathcal{A}}$ for some \mathcal{A} and $SK_{\mathcal{A},u}$ for some u and \mathcal{A} yields terms with exponents of the form

$$\sum_u \left(\gamma_u mk_u \sum_{\mathcal{A}} \gamma_{\mathcal{A},u} mk_{\mathcal{A}} \right) + \gamma' \sum_{\mathcal{A}} mk_{\mathcal{A}} ,$$

for some γ_u , $\gamma_{\mathcal{A},u}$ and γ' . Since the adversary does not have all secret attribute keys corresponding to any one user u to satisfy any conjunction, no sum $\sum_{\mathcal{A}} \gamma_{\mathcal{A},u} mk_{\mathcal{A}}$ will yield any a_j . Furthermore, the simulator chooses all $mk_{\mathcal{A}}$ randomly, so any oracle query involving any sum over $\sum_{\mathcal{A}} mk_{\mathcal{A}}$ with a set of attributes that does not precisely correspond to the attributes of the challenge \mathbb{A} gives no information about a_j . The only way that the sum $\gamma' \sum_{\mathcal{A}} mk_{\mathcal{A}}$ evaluates to a_j for some j is as a product of corresponding public attribute keys, which is obtained by querying the multiplication oracle with all representations of $PK'_{\mathcal{A}}$, $\mathcal{A} \in S_j$, yielding $\xi(a_j)$. It follows that to construct a term containing a_j , the adversary has no other option than to use either E_j , E''_j , or $\prod_{\mathcal{A} \in S_j} PK_{\mathcal{A}}$ for any j . Other terms containing $mk_{\mathcal{A}}$ are not useful for him.

Next we consider how to obtain terms containing y and R_j . All R_j and y are random, so the only way to construct a relevant pairing is to pair two representations of terms from \mathbb{G} by querying the pairing oracle, such that both y and one R_j are contained in one of the terms. The only values in \mathbb{G} that contain y are SK_u . We examine all possible results from pairing some γSK_u with some other value. As shown above, we need not consider terms where the adversary has some $mk_{\mathcal{A}}$, but not all to create a value a_j .

The first three columns of Table 2 list all remaining combinations. It can be seen that the only result that contains all y , a_j and R_j is the pairing of some SK_u and some E''_j which results in

$$\xi_T(\tilde{p} R_j mk_u a_j + ya_j R_j) .$$

In order to obtain the required term $\xi_T(\gamma y a_j R_j)$, the adversary will have to eliminate the first term, $\tilde{p} R_j \text{mk}_u a_j$. To construct this, he needs to pair a term containing \tilde{p} with another term. Thus we need to examine all possible results from pairing SK_u or E'_j (the only terms depending on \tilde{p}) with another value. Once again, Table 2 on the previous page lists all possible combinations not containing terms involving results of the hash oracle. (Including terms given by the oracles one gets terms of the above form that will not help, either.) We can conclude from the case analysis that no term of the form $\xi_T(\tilde{p} R_j \text{mk}_u a_j)$ can be constructed.

Thus, the term $\xi_T(y a_j R_j)$ cannot be constructed by the adversary, which contradicts the assumption that the views in the modified game are not identically distributed. Thus, any Adv_2 will have negligible success to win the game. In turn, a successful Adv_1 cannot exist either, which proves the theorem. \square