Bull. Korean Math. Soc. **46** (2009), No. 4, pp. 803–819
DOI 10.4134/BKMS.2009.46.4.803

ON MULTI-AUTHORITY CIPHERTEXT-POLICY ATTRIBUTE-BASED ENCRYPTION

Sascha Müller, Stefan Katzenbeisser, and Claudia Eckert

Abstract. In classical encryption schemes, data is encrypted under a single key that is associated with a user or group. In Ciphertext-Policy Attribute-Based Encryption(CP-ABE) keys are associated with attributes of users, given to them by a central trusted authority, and data is encrypted under a logical formula over these attributes. We extend this idea to the case where an arbitrary number of independent parties can be present to maintain attributes and their corresponding secret keys. We present a scheme for multi-authority CP-ABE, propose the first two constructions that fully implement the scheme, and prove their security against chosen plaintext attacks.

1. Introduction

Ciphertext-Policy Attribute-Based Encryption (CP-ABE) makes it possible to encrypt data under an access policy, specified as a Boolean formula over attributes. Ciphertexts can only be decrypted by users who possess all attributes required to satisfy the access policy. Decryption is performed by using secret attribute keys; one such attribute key corresponds to one attribute of a user. Thus, only a user who possesses the right number and combination of these secret keys is able to access the data. Common to most previous ABE schemes is the existence of a central trusted authority (*master*) that knows a secret master key and distributes attribute keys to eligible users. However, for many practical scenarios that benefit from the use of attribute-based encryption, there is no central authority that is able to maintain all attributes and distribute secret attribute keys.

In this paper we introduce the concept of Distributed Attribute-Based Encryption (DABE), which allows to mitigate the above-mentioned problem. In this scenario, a central master is only responsible for the distribution of secret user keys. In contrast to standard CP-ABE schemes, this party is *not* involved

Received December 20, 2008.

2000 *Mathematics Subject Classification.* 94A60.

Key words and phrases. attribute-based encryption, bilinear groups.

The preliminary version of this paper appeared in the proceedings of the International Conference on Information Security and Cryptology (ICISC) 2008.

in the creation of secret attribute keys. The latter task can independently be performed by several attribute authorities. 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 user. Furthermore, the attribute authorities determine eligible users and distribute personalized secret attribute keys over an authenticated and trusted channel to them.

To encrypt a message, a user first formulates his access policy. Depending on the construction the form of this policy may be a Boolean formula, a linear secret sharing scheme, or a different formalism. The user can finally encrypt a message under a policy by using the public keys corresponding to the attributes occurring in the policy. To decrypt a ciphertext, a user needs at least access to some set of attributes (and their associated attribute 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 for. A formal definition of a DABE scheme can be found in Section 2.

Attribute-Based Encryption was first proposed by Goyal et al. [9] in the form of *key-policy* attribute-based encryption (KP-ABE), based on the work of Sahai and Waters [11]. 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 [2], followed by some cryptographically stronger CP-ABE constructions that allowed reductions to the Decisional Bilinear Diffie Hellman Problem [6, 8], but imposed restrictions that the original CP-ABE does not have. There is only one attempt at multi-authority CP-ABE, proposed by Chase [5] as an extension of her multi-authority threshold ABE construction. However, the scheme does not provide the full flexibility of a DABE scheme and can only be used for a constrained type of access policies.

In this paper we first formally define the concept of Distributed Attribute-Based Encryption and introduce the attacker model considered in the work (Section 2). Subsequently, we give two DABE constructions (Sections 3 and 4); the first one is very efficient, but its security can be proven only in an idealized model (the generic group model), whereas the second construction, which is a simple extension of a recent work by Waters [14], can be proven secure under a number-theoretic assumption, but is less efficient and requires a weaker attacker model. Finally, we conclude in Section 5.

2. Distributed attribute-based encryption

In this section we formally define the concept of Distributed Attribute-Based Encryption and introduce the attacker model that is used to prove the security of a DABE scheme.

2.1. The DABE scheme

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

Setup: The *Setup* algorithm takes as input the security parameter 1^k. It outputs the public key PK which is used in all subsequent algorithms and the secret master key MK.

CreateUser$(\mathrm{PK}, \mathrm{MK}, u)$**:** 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$(\mathrm{PK}, \mathcal{A}, \mathrm{SK}_a)$**:** The *RequestAttributePK* algorithm is executed by attribute authorities whenever they receive a request for a public attribute key. The algorithm checks whether the authority is responsible for the attribute $\mathcal{A}$. If this is the case, the algorithm outputs a public attribute key $\mathrm{PK}_{\mathcal{A}}$, otherwise `NULL`.

RequestAttributeSK$(\mathrm{PK}, \mathcal{A}, \mathrm{SK}_a, u, \mathrm{PK}_u)$**:** 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 it has authority over attribute $\mathcal{A}$ and whether the user u with public key PK_u is eligible for $\mathcal{A}$. If this is the case, *RequestAttributeSK* outputs a secret attribute key $\mathrm{SK}_{\mathcal{A},u}$ for user u. Otherwise, the algorithm outputs `NULL`.

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

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

Note that this scheme differs from CP-ABE [2] in that the two algorithms *CreateAuthority* and *RequestAttributePK* were added, and the algorithm *KeyGen* of CP-ABE is split into *CreateUser* and *RequestAttributeSK*. It also differs from the construction [5], where all attribute authorities are maintained by the central authority. 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 (by running *CreateUser*).

2.2. 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 public 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, which the challenger creates by calling *RequestAttributeSK* or *RequestAttributePK*, respectively. During the first request for a public or private key for an attribute of an authority a, the challenger creates the authority by a call to *CreateAuthority*; he stores the secret authority key for future use (but does not make the key available to the attacker).

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 satisfies $\mathbb{A}$. (If any user from Phase 1 satisfies $\mathbb{A}$, the challenger aborts.) 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 $\epsilon = \Pr[b' = b] - \frac{1}{2}$, where the probability is taken over all coin tosses of both challenger and adversary. A DABE scheme is called secure if all polynomial time adversaries have at most a negligible advantage in the above game, i.e., if ϵ, viewed as a function of the security parameter k used for initialization of the scheme, satisfies $\epsilon(k) < 1/p(k)$ for all polynomials $p(\cdot)$ and sufficiently large $k \in \mathbb{N}$.

3. DABE construction

We next give a first and very efficient construction of a DABE scheme, which is based on bilinear groups and requires access policies in the form of Boolean formulas written in Disjunctive Normal Form (DNF); the algorithms introduced in Section 2 are implemented as follows:

***Setup*:** The *Setup* algorithm chooses a pairing $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ of order p [3]. Next it chooses a generator $g \in \mathbb{G}$, and two random group elements $P, Q \in \mathbb{G}$. The public key of the system is $\mathrm{PK} = \{\mathbb{G}, \mathbb{G}_T, e, g, P, e(g, Q)\}$, while the secret master key is given by $\mathrm{MK} = Q$. Note that our construction can easily be modified to work with asymmetric pairings $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, which can be more efficient for high security settings [7]. In this case, let g be a generator of $\mathbb{G}_1$ and choose the random elements $P, Q \in \mathbb{G}_2$.

CreateUser$(\mathrm{PK}, \mathrm{MK}, u)$**:** The algorithm *CreateUser* chooses a secret $\mathrm{mk}_u \in \mathbb{Z}_p$ and outputs the public key $\mathrm{PK}_u := g^{\mathrm{mk}_u}$ and the private key $\mathrm{SK}_u := \mathrm{MK} \cdot P^{\mathrm{mk}_u} = Q \cdot P^{\mathrm{mk}_u}$ 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 large finite family of hash functions, which we model as random oracles. It returns as secret key the index of the hash function $\mathrm{SK}_a := x_a$.

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

$$\mathrm{PK}_A := \left\langle \mathrm{PK}'_{\mathcal{A}} := g^{H_{\mathrm{SK}_a}(\mathcal{A})}, \mathrm{PK}''_{\mathcal{A}} := e(g, Q)^{H_{\mathrm{SK}_a}(\mathcal{A})} \right\rangle .$$

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

RequestAttributeSK$(\mathrm{PK}, \mathcal{A}, \mathrm{SK}_a, u, \mathrm{PK}_u)$**:** After determining that the attribute $\mathcal{A}$ is handled by a, the authority tests whether user u is eligible for the attribute $\mathcal{A}$. (The implementation of this operation is application-dependent and thus outside the scope of this paper.) If this is not the case, *RequestAttributeSK* returns `NULL`, else it outputs the secret attribute key $\mathrm{SK}_{\mathcal{A},u} := {\mathrm{PK}_u}^{H_{\mathrm{SK}_a}(\mathcal{A})} = g^{\mathrm{mk}_u \, H_{\mathrm{SK}_a}(\mathcal{A})}$. Note that the recipient u can check the validity of this secret key by testing if $e(\mathrm{PK}'_{\mathcal{A}}, \mathrm{SK}_u) = \mathrm{PK}''_{\mathcal{A}} \cdot e(\mathrm{SK}_{\mathcal{A},u}, P)$.

Encrypt$(\mathrm{PK}, M, \mathbb{A}, \mathrm{PK}_{\mathcal{A}_1}, \ldots, \mathrm{PK}_{\mathcal{A}_N})$**:** 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 sets $S_1, \ldots, S_N$ denote attributes that occur in the j-th conjunction of $\mathbb{A}$. The encryption algorithm iterates over all $j = 1, \ldots, N$, generates for each conjunction a random value $R_j \in \mathbb{Z}_p$ and constructs a tuple $\mathrm{CT}_j = \langle E_j, E'_j, E''_j \rangle$, where

$$E_j := M \cdot (\prod_{\mathcal{A} \in S_j} \mathrm{PK}''_{\mathcal{A}})^{R_j}, \quad E'_j := P^{R_j}, \text{ and } E''_j := (\prod_{\mathcal{A} \in S_j} \mathrm{PK}'_{\mathcal{A}})^{R_j}. \tag{1}$$

The ciphertext CT is obtained as tuple $\mathrm{CT} := \langle \mathrm{CT}_1, \ldots, \mathrm{CT}_N \rangle$.

Decrypt$(\mathrm{PK}, \mathrm{CT}, \mathbb{A}, \mathrm{SK}_u, \mathrm{SK}_{\mathcal{A}_1,u}, \ldots, \mathrm{SK}_{\mathcal{A}_N,u})$**:** 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 $\mathrm{SK}_{\mathcal{A}_1,u}, \ldots, \mathrm{SK}_{\mathcal{A}_N,u}$ contains at least secret keys for all attributes occurring in a set S_j for some $1 \le j \le N$. If this is not the case, the algorithm outputs `NULL`, otherwise

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

It is easy to see that the decryption is correct. Let $a_j := \sum_{\mathcal{A} \in S_j} H_{\mathrm{SK}_{a_\mathcal{A}}}(\mathcal{A})$, where for all $\mathcal{A}$, $a_\mathcal{A}$ is the attribute authority of $\mathcal{A}$. Then

$$E_j = M \cdot e(g, Q)^{a_j R_j}, \quad E''_j = g^{a_j R_j} \tag{2}$$

and

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

3.1. Security

We closely follow the structure of the security proof of the CP-ABE scheme introduced in [2]. First we show how any adversary who plays the DABE game of Section 2.2 (denoted $\texttt{Adv}_1$ in the following) can be used to construct an adversary in a slightly modified game (denoted $\texttt{Adv}_2$). Then we prove that no such $\texttt{Adv}_2$ can exist, 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 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 satisfies $\mathbb{A}$. The challenger flips a coin b, and creates a ciphertext for the access policy $\mathbb{A}$ according to Eq. (1), but instead of computing E_j as in Eq. (2), he computes E_j as

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

where all θ_j are uniformly and independently chosen random elements of $\mathbb{Z}_p$. The task of $\texttt{Adv}_2$ is thus to distinguish the two group elements $e(g, Q)^{a_j R_j}$ and $e(g, g)^{\theta_j}$ of $\mathbb{G}_T$.

Lemma 1. *If there exists an adversary* $\texttt{Adv}_1$ *who has advantage of* ϵ *to win the original game, then there exists an adversary* $\texttt{Adv}_2$ *which wins the modified game with advantage* $\epsilon/2$.

Proof. Given an adversary $\texttt{Adv}_1$ that has advantage ϵ in the DABE 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 two messages M_0 and M_1 from $\texttt{Adv}_1$ and the challenge C (which contains elements E_j that are either $e(g,Q)^{a_j R_j}$ or $e(g,g)^{\theta_j}$ for all $1 \le j \le N$) from the challenger. He flips a coin β, multiplies all E_j of C by M_β, and sends the resulting ciphertext as C' to $\texttt{Adv}_1$. When $\texttt{Adv}_1$ outputs a guess β', $\texttt{Adv}_2$ outputs 1 if $\beta' = \beta$, and 0 if $\beta' \neq \beta$. If the components E_j of C satisfy $E_j = e(g,Q)^{a_j R_j}$, then $\texttt{Adv}_2$'s challenge given to $\texttt{Adv}_1$ is a well-formed DABE ciphertext and $\texttt{Adv}_1$ has advantage ϵ of guessing the correct $\beta' = \beta$. If $E_j = e(g,g)^{\theta_j}$, the challenge is independent of the messages M_0 and M_1, so the advantage of $\texttt{Adv}_2$ is 0. Thus, we have

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

and the overall advantage of $\texttt{Adv}_2$ is $\frac{\epsilon}{2}$, as required. □

We prove the security of our DABE construction in the generic group model [12], with the extensions for bilinear groups with a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ developed in [4], which we simplify slightly for our case where $\mathbb{G}_1 = \mathbb{G}_2$. In particular, we show that any polynomial time adversary $\texttt{Adv}_2$, who plays the modified game cannot have non-negligible advantage to distinguish $e(g,Q)^{a_j R_j}$ from $e(g,g)^{\theta_j}$ in his view of the protocol. Lemma 1 finally implies that there exists no efficient successful attacker $\texttt{Adv}_1$ either, which proves the security of the DABE scheme.

In the generic group model, the adversary is given only encoded versions of all group elements, which look like random strings. For groups $\mathbb{G}$ and $\mathbb{G}_T$ of prime order p and a generator $\tilde{g} \in \mathbb{G}$ we use random maps $\xi, \xi_T : \mathbb{Z}_p \to \{0,1\}^m$ for sufficiently large m to encode any element $\tilde{g}^x$ or $e(\tilde{g},\tilde{g})^x$ as a random string $\xi(x)$ or $\xi_T(x)$. The maps ξ and ξ_T must be invertible, so that the representations of group elements can be transformed back to elements of $\mathbb{G}$ and $\mathbb{G}_T$. To manipulate these encoded group elements, the attacker gets access to five oracles, which compute multiplication and division operations in $\mathbb{G}$ and $\mathbb{G}_T$ and the pairing operation e. All oracles take as input string representations of group elements. Given two string representations $\xi(a)$ and $\xi(b)$ of elements $\tilde{g}^a, \tilde{g}^b \in \mathbb{G}$, the adversary can query two different oracles (the multiplication and the division oracle) for the result of the group operations $\tilde{g}^a \cdot \tilde{g}^b$ and $\tilde{g}^a \cdot \tilde{g}^{-b}$.

Both oracles will map the coded inputs $\xi(a)$ and $\xi(b)$ back to the respective elements of $\mathbb{G}$ using ξ^{-1}, execute the group operation and map the result to a string using ξ. From the view of the adversary, the multiplication oracle returns $\xi(a+b)$, while the division oracle returns $\xi(a-b)$. The oracles for computing multiplications and divisions in $\mathbb{G}_T$ operate analogously, by using the encoding ξ_T instead of ξ. Note that no oracle will accept input from different encodings (for example, one cannot feed a value $\xi_T(b)$ into an oracle for a group operation of $\mathbb{G}$). The pairing oracle can be implemented easily: Given two encodings $\xi(a)$ and $\xi(b)$, the encoding of the pairing is given by $\xi_T(ab)$. A scheme proven secure in this model is called *generically secure* and can only be broken by exploiting specific algebraic properties of the groups used in an implementation.

Theorem 1. *Let* $\mathtt{Adv}_2$ *be a polynomial time adversary in the generic group model who plays the modified DABE security game and makes q oracle queries. Then* $\mathtt{Adv}_2$ *has advantage at most $O(q^2/p)$ to win the modified game, where p is the order of the bilinear group.*

Proof. Let $\mathtt{Adv}_2$ be a polynomial time adversary against the modified security game. $\mathtt{Adv}_2$ plays against a simulator, who takes over the role of the challenger and manages all oracles. In particular, the simulator operates in the following way:

Setup: The simulator chooses $\mathbb{G}$, $\mathbb{G}_T$, e, $\tilde{g}$ and random exponents $\tilde{p}, \tilde{q} \in \mathbb{Z}_p$. Furthermore the simulator chooses two random encoding functions ξ, ξ_T for the implementation of the group and pairing oracles. The public key is given to the adversary in encoded form, i.e., the adversary obtains $\xi(1), \xi(\tilde{p})$ and $\xi_T(\tilde{q})$ as encoded versions of g, P and $e(g, Q)$.

Phase 1: When the adversary calls *CreateUser* for some u, the simulator chooses a random $\mathrm{mk}_u \in \mathbb{Z}_p$ and returns encoded versions $\xi(\mathrm{mk}_u)$ and $\xi(\tilde{q} + \tilde{p} \cdot \mathrm{mk}_u)$ of the user keys PK_u and SK_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 $\mathrm{mk}_{\mathcal{A}}$, which simulates the term $H_{\mathrm{SK}_a}(\mathcal{A})$ of an attribute $\mathcal{A}$ maintained by attribute authority a; the association between values $\mathrm{mk}_{\mathcal{A}}$ and attributes $\mathcal{A}$ is stored internally by the simulator. If the adversary queries for an attribute $\mathcal{A}$ that was used before, the value $\mathrm{mk}_{\mathcal{A}}$ is retrieved from storage. During every request of a public attribute key for $\mathcal{A}$ (a call to *RequestAttributePK*), the simulator returns $\xi(\mathrm{mk}_{\mathcal{A}})$ and $\xi_T(\tilde{q}\,\mathrm{mk}_{A})$ as encoded versions of the public attribute keys $\mathrm{PK}'_{\mathcal{A}}$ and $\mathrm{PK}''_{\mathcal{A}}$. If queried for a secret attribute key (through a call to *RequestAttributeSK*), the simulator returns $\xi(\mathrm{mk}_u\,\mathrm{mk}_{\mathcal{A}})$ as encoded secret key $\mathrm{SK}_{\mathcal{A},u}$.

Whenever the adversary makes oracle queries for group operations or the pairing, the adversary gets the desired result: on input $\xi_T(a)$ and $\xi_T(b)$, the multiplication oracles returns $\xi_T(a+b)$ and the division oracle returns $\xi_T(a-b)$. Similarly, on input $\xi(a)$ and $\xi(b)$, the multiplication oracle returns $\xi(a+b)$ and the division oracle returns $\xi(a-b)$. On input $\xi(a)$ and $\xi(b)$, the pairing oracle returns $\xi_T(ab)$.

Challenge: When the adversary asks for a challenge by submitting the access policy $\mathbb{A}$, the simulator flips a coin b. Then he chooses a random $R_j \in \mathbb{Z}_p$ for each conjunction in $\mathbb{A}$ and computes $a_j = \sum_{\mathcal{A} \in S_j} \mathrm{mk}_{\mathcal{A}}$. If $b = 0$, he sets θ_j to a random value from $\mathbb{Z}_p$, otherwise $\theta_j := \tilde{q} a_j R_j$. Finally he returns encoded components of the ciphertext CT_j as $\langle \xi_T(\theta_j), \xi(\tilde{p} R_j), \xi(a_j R_j) \rangle$.

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 $\mathbb{A}$.

Due to the restriction of the generic group model, all values that the adversary can access at any time during his attack are either encodings of random values of $\mathbb{Z}_p$ (namely $1, \tilde{p}, \tilde{q}, \mathrm{mk}_u, \mathrm{mk}_{\mathcal{A}}$ and θ), encodings of combinations of these values given by the simulator (such as $\mathrm{mk}_u \, \mathrm{mk}_{\mathcal{A}}$ representing $\mathrm{SK}_{\mathcal{A},u}$), or results of oracle queries, which are encodings of sums and differences of such values. We keep track of the knowledge of an attacker by using symbolic algebraic expressions over these variables which represent inputs of oracle queries. For simplicity of notation, we will often drop the encoding functions; however we stress that the attacker only has encoded values available.

Due to the random choice of ξ and ξ_T, two terms that evaluate to different values over $\mathbb{Z}_p$ yield different encodings when mapped by ξ and ξ_T, except if due to the choice of the random encodings two different values "accidentally" are mapped to the same string. Similar to the proof in [2] 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 on the event that *no* such random collisions occur.

Now, under this assumption consider how the adversary's views differ between the case where the θ_j is random ($b = 0$) and the case where $\theta_j = \tilde{q} a_j R_j$ ($b = 1$). 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 prove this claim, assume the opposite. Since the adversary can only test for equality of strings 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 known to the attacker that evaluate to the same value in the view where $\theta_j = \tilde{q} a_j R_j$ ($b = 1$) for some j, and to a different value in the view corresponding to $b = 0$. Call two such terms ν_1 and ν_2 and fix the index j. Since θ_j only occurs as $\xi_T(\theta_j)$, representing E_j, and elements encoded with ξ_T can not be paired, the adversary can only construct expressions by using the multiplication and division orales for $\mathbb{G}_T$, resulting in queries where θ_j appears as an additive term. Thus, ν_1 and ν_2 can be written as

$$\nu_1 = \gamma_1\theta_j + \nu_1',$$
$$\nu_2 = \gamma_2\theta_j + \nu_2'$$

for some ν_1' and ν_2' that do not contain θ_j. Since by assumption $\theta_j = \tilde{q}a_jR_j$ results in $\nu_1 = \nu_2$, we have $\gamma_1\tilde{q}a_jR_j + \nu_1' = \gamma_2\tilde{q}a_jR_j + \nu_2'$. Rearranging the equation yields

$$\nu_1' - \nu_2' = (\gamma_2 - \gamma_1)\tilde{q}a_jR_j.$$

Thus, the adversary can construct an oracle query for a term $\gamma\tilde{q}a_jR_j$ with $\gamma \neq 0$, by using terms ν_1' and ν_2' in his possession. (We can, without loss of generality, add the query $\gamma\tilde{q}a_jR_j$ to the queries made by the attacker). 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 $\xi_T(\gamma\tilde{q}a_jR_j)$ for any $\gamma \in \mathbb{Z}_p$ and j from the information that he has. This contradicts the assumption that the views in the modified game are not identically distributed and proves the theorem.

After Phase 2, the adversary received the following information from the simulator, all in encoded form:

- The tuple PK, i.e., $\xi(1), \xi(\tilde{p})$ and $\xi_T(\tilde{q})$.
- PK_u and SK_u for an arbitrary number of users, i.e., $\xi(\mathrm{mk}_u)$ and $\xi(\tilde{q} + \tilde{p} \cdot \mathrm{mk}_u)$.
- $\mathrm{PK}_{\mathcal{A}}'$ and $\mathrm{PK}_{\mathcal{A}}''$ for an arbitrary number of attributes, i.e., $\xi(\mathrm{mk}_{\mathcal{A}})$ and $\xi_T(\tilde{q}\,\mathrm{mk}_A)$.
- $\mathrm{SK}_{\mathcal{A},u}$ for an arbitrary number of attributes and users, i.e., $\xi(\mathrm{mk}_u\mathrm{mk}_{\mathcal{A}})$, with the restriction that for no user 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, i.e., $\xi_T(\theta_j), \xi(\tilde{p}R_j)$ and $\xi(a_jR_j)$.

Furthermore, he possibly obtained arbitrary combinations of these encodings through queries to the five oracles that implement group operations and the pairing.

Since $\tilde{q}$ and all R_j are random, and a_j is defined as a sum of random values over $\mathbb{Z}_p$, the only way to construct $\xi_T(\gamma\tilde{q}a_jR_j)$ is to pair two representations of terms of $\mathbb{G}$ by querying the pairing oracle, so that each of the components $\tilde{q}, a_j$, and R_j is contained in any of the terms.

First we show how the adversary can find terms containing $a_j = \sum_{\mathcal{A}\in S_j} \mathrm{mk}_{\mathcal{A}}$ in $\mathbb{G}$, since a_j is contained in the required term and thus needs to be contained in an input term for the pairing oracle. Aside from E_j and E_j'', a_j can only be constructed by querying the multiplication oracles for encodings of terms containing $\mathrm{mk}_{\mathcal{A}}$ for all $\mathcal{A} \in S_j$ and some j with $1 \leq j \leq n$. These values occur only in $\mathrm{PK}_{\mathcal{A}}'$, $PK_{\mathcal{A}}''$, and $\mathrm{SK}_{\mathcal{A},u}$. Since $PK_A'' \in \mathbb{G}_T$, it cannot be used as input of the pairing. Thus, the only possibility for the attacker is to combine any $\mathrm{PK}_{\mathcal{A}}'$ and $\mathrm{SK}_{\mathcal{A},u}$. Multiplying representations of $\mathrm{PK}_{\mathcal{A}}'$ (by calling the multiplication oracle) yields representations of terms of the form $\gamma \sum_{\mathcal{A}} \mathrm{mk}_{\mathcal{A}}$ for some γ, and multiplying this by $\mathrm{SK}_{\mathcal{A},u}$ for some u and $\mathcal{A}$ yields representations of terms of the form

$$\sum_{u} \left(\gamma_u \, \mathrm{mk}_u \sum_{\mathcal{A}} \gamma_{\mathcal{A},u} \, \mathrm{mk}_{\mathcal{A}} \right) + \gamma \sum_{\mathcal{A}} \mathrm{mk}_{\mathcal{A}}$$

for some γ, γ_u and $\gamma_{\mathcal{A},u}$. Since the adversary does not have all secret attribute keys corresponding to one user u to satisfy any conjunction of $\mathbb{A}$, no sum $\sum_{\mathcal{A}} \gamma_{\mathcal{A},u} \, \mathrm{mk}_{\mathcal{A}}$ will evaluate to the required a_j. Furthermore, the simulator chooses all $\mathrm{mk}_{\mathcal{A}}$ randomly, so any oracle query involving any sum over $\sum_{\mathcal{A}} \mathrm{mk}_{\mathcal{A}}$ with a set of attributes that does not precisely correspond to the attributes of the challenge $\mathbb{A}$ will not yield a term containing a_j. Thus, the first sum of the above term can not yield a_j. The only way that the sum $\gamma \sum_{\mathcal{A}} \mathrm{mk}_{\mathcal{A}}$ 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 $\mathrm{PK}'_{\mathcal{A}}, \mathcal{A} \in S_j$, 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 $\prod_{\mathcal{A} \in S_j} \mathrm{PK}'_{\mathcal{A}}$. Other terms containing $\mathrm{mk}_{\mathcal{A}}$ are not useful for him.

Next we consider how to obtain terms containing $\tilde{q} \cdot R_j$. None of the values that the adversary has contains both $\tilde{q}$ and R_j. Thus to get a representation of a term containing the product $\tilde{q} \cdot R_j$, the adversary needs to pair two terms from $\mathbb{G}$, where each of the values is contained in any term. The only values in $\mathbb{G}$ that contain $\tilde{q}$ are SK_u. We examine all possible results from pairing some $\gamma \, \mathrm{SK}_u$ with some other value. As shown above, we need not consider terms containing $\mathrm{mk}_{\mathcal{A}}$, since these are not useful for the adversary.

Table 1. Results of pairings

Source	Term	Pairing with SK_u	Pairing with E'_j
$\mathrm{PK}_{u'}$	$\mathrm{mk}_{u'}$	$\mathrm{mk}_{u'} \, \tilde{q} + \tilde{p} \, \mathrm{mk}_u \, \mathrm{mk}_{u'}$	$\mathrm{mk}_u \, \tilde{p} R_j$
$\mathrm{SK}_{u'}$	$\tilde{q} + \tilde{p} \, \mathrm{mk}_{u'}$	$\tilde{q}^2 + \tilde{q}\tilde{p}(\mathrm{mk}_{u'} + mk_u)$ $+ \tilde{p}^2 \, \mathrm{mk}_u \, \mathrm{mk}_{u'}$	$\tilde{q}\tilde{p} R_j + \tilde{p}^2 \, \mathrm{mk}_u \, R_j$
$\prod_{\mathcal{A} \in S_j} \mathrm{PK}'_{\mathcal{A}}$	a_j	$\tilde{q} a_j + \tilde{p} \, \mathrm{mk}_u \, a_j$	$a_j \tilde{p} R_j$
E'_j	$\tilde{p} R_j$	$\tilde{q}\tilde{p} R_j + \tilde{p}^2 \, \mathrm{mk}_u \, R_j$	$\tilde{p}^2 R_j^2$
E''_j	$a_j R_j$	$\tilde{q} a_j R_j + \tilde{p} \, \mathrm{mk}_u \, a_j R_j$	$a_j \tilde{p} R_j^2$

The first three columns of Table 1 list all remaining combinations. It can be seen that the only result that contains all $\tilde{q}$, a_j and R_j is the pairing of some SK_u and some E''_j which results in a representation of

$$\xi_T(\tilde{p} R_j \, \mathrm{mk}_u \, a_j + \tilde{q} a_j R_j).$$

In order to obtain the required term $\xi_T(\gamma \tilde{q} a_j R_j)$, the adversary has to eliminate the first term, $\tilde{p} R_j \, \mathrm{mk}_u \, a_j$, which can be done by a call to the division oracle of

$\mathbb{G}_T$ if the adversary has a value $\xi_T(\tilde{p}R_j\,\mathrm{mk}_u\,a_j)$ available. 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 1 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 $\xi_T(\tilde{p}R_j\,\mathrm{mk}_u\,a_j)$ can be constructed through oracle calls, so $\xi_T(\gamma\tilde{q}a_jR_j)$ cannot be constructed either. □

3.2. 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 $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, 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.

4. Waters' construction

It is easy to modify the CP-ABE constructions from [14] in order to obtain a DABE construction since the structure of the secret attribute keys is similar to the DABE construction given in the preceding section. For this construction the access policy $\mathbb{A}$ must be given as a linear secret sharing scheme. For a formal definition of secret sharing schemes and access structures we refer the reader to [13]. We define a linear secret sharing scheme (LSSS) as follows [1]:

Definition 1. A secret-sharing scheme Π over a set of parties $\mathcal{P}$ is called linear (over $\mathbb{Z}_p$) if

(1) The shares for each party form a vector over $\mathbb{Z}_p$.
(2) There exists a matrix $\mathcal{M}$ called the share-generating matrix for Π. The Matrix $\mathcal{M}$ has ℓ rows and n columns. Let $\rho : \{1, \ldots, \ell\} \to \mathcal{P}$ a function that maps each row of $\mathcal{M}$ to a party. When we consider the column vector $v = (s, r_2, \ldots, r_n)$ where $s \in \mathbb{Z}_p$ is the secret to be shared, and $r_2, \ldots, r_n \in \mathbb{Z}_p$ are randomly chosen, then $\mathcal{M}v$ is the vector of ℓ shares of the secret s according to Π. The share $\lambda_i := (\mathcal{M}v)_i$ belongs to party $\rho(i)$.

It has been shown in [1] that if S is a set of parties that is allowed to receive the secret according to Π (i.e., an *authorized subset* of the access structure realized by Π) and $I = \{i \,|\, \rho(i) \in S\}$ is the set of rows of $\mathcal{M}$ corresponding to the elements of S, then there exist constants $\{\omega_i \in \mathbb{Z}_p\}_{i\in I}$, such that $\sum_{i\in I} \omega_i\lambda_i = s$ for the shares $\lambda_i = (\mathcal{M}v)_i = \mathcal{M}_i v$. These constants can be found in polynomial time. In our setting, the parties $\mathcal{P}$ resemble attributes, so an encryptor first needs to formulate his access policy as a secret sharing scheme Π and construct a matrix $\mathcal{M}$ along with a row-labeling function ρ for it. As in the original scheme, the encryptor also needs the public keys $\mathrm{PK}_{\mathcal{A}}$ of all attributes used in the access policy as input to the encryption algorithm. Using the notation from Section 3, the construction is as follows:

***Setup*:** The *Setup* algorithm chooses a pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ of order p. Next it chooses a generator $g \in \mathbb{G}$, and two random group elements $P, Q \in \mathbb{G}$. The public key of the system is $\mathrm{PK} = \{\mathbb{G}, \mathbb{G}_T, e, g, P, e(g, Q)\}$, while the secret master key is given by $\mathrm{MK} = Q$.

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

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

RequestAttributePK$(\mathrm{PK}, \mathcal{A}, \mathrm{SK}_a)$**:** If $\mathcal{A}$ is handled by the attribute authority a, *RequestAttributePK* returns the public attribute key of $\mathcal{A}$: $\mathrm{PK}_{\mathcal{A}} := g^{H_{\mathrm{SK}_a}(\mathcal{A})}$.

RequestAttributeSK$(\mathrm{PK}, \mathcal{A}, \mathrm{SK}_a, u, \mathrm{PK}_u)$**:** After determining that the attribute $\mathcal{A}$ is handled by 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 $\mathrm{SK}_{\mathcal{A},u} := {\mathrm{PK}_u}^{H_{\mathrm{SK}_a}(\mathcal{A})} = g^{\mathrm{mk}_u \, H_{\mathrm{SK}_a}(\mathcal{A})}$.

Encrypt$(\mathrm{PK}, M, \mathbb{A}, \mathrm{PK}_{\mathcal{A}_1}, \ldots, \mathrm{PK}_{\mathcal{A}_N})$**:** Given an access policy of the form $\mathbb{A} = \langle \mathcal{M}, \rho \rangle$, *Encrypt* chooses a random vector $\vec{v} = (s, y_2, \ldots, y_n) \in \mathbb{Z}_p^{n+1}$. For $i = 1$ to ℓ, it calculates the share $\lambda_i = \mathcal{M}_i \cdot v$. In addition, it chooses random $r_1, \ldots, r_\ell \in \mathbb{Z}_p$. The ciphertext is

$$\begin{aligned}\mathrm{CT} = \big\langle & C = Me(g,Q)^s, C' = g^s, \\ & C_1 = P^{\lambda_1} {\mathrm{PK}_{\rho(1)}}^{-r_1}, D_1 = g^{r_1}, \\ & \ldots \\ & C_\ell = P^{\lambda_\ell} {\mathrm{PK}_{\rho(\ell)}}^{-r_\ell}, D_\ell = g^{r_\ell} \big\rangle.\end{aligned}$$

Decrypt$(\mathrm{PK}, \mathrm{CT}, \mathbb{A}, \mathrm{SK}_u, \mathrm{SK}_{\mathcal{A}_1,u}, \ldots, \mathrm{SK}_{\mathcal{A}_N,u})$**:** If the attributes of $\mathrm{SK}_{\mathcal{A}_1,u}, \ldots, \mathrm{SK}_{\mathcal{A}_N,u}$ satisfy the access structure $\mathbb{A}$, let I be the set of all row indices of $\mathcal{M}$ that are associated with the attributes of $\mathrm{SK}_{\mathcal{A}_1,u}, \ldots, \mathrm{SK}_{\mathcal{A}_N,u}$, i.e., $I = \{i \mid \rho(i) \in \{\mathcal{A}_1, \ldots, \mathcal{A}_N\}\}$. As noted above, there are constants $\{\omega_i \mid i \in I\}$, so that $\sum_{i \in I} \omega_i \lambda_i = s$, which can be found in polynomial time. The algorithm computes

$$e(C', \mathrm{SK}_u) / \left(\prod_{i \in I} (e(C_i, \mathrm{PK}_u) \cdot e(D_i, SK_{\rho(i),u}))^{\omega_i} \right) = e(g, Q)^s$$

and divides the ciphertext component C by this value, thus retrieving M.

The only major difference to the construction from Section 3 is in the algorithms *Encrypt* and *Decrypt*.

4.1. Security

To prove the security of his construction, Waters uses a security assumption that he introduces in his paper, the decisional q-parallel Bilinear Diffie Hellman Exponent assumption. He shows that any adversary who breaks the scheme can be used to construct an algorithm that breaks the decisional q-parallel BDHE assumption.

This proof also works for the DABE modification. However, we have to weaken the attack model which is considered. In particular we allow only *non-adaptive key queries*, similar to the Multi-Authority Threshold ABE scheme by Chase [5]: In this scheme, a sequence of calls of *CreateUser* to request user keys for a user u and an arbitrary number of calls to *RequestAttributeSK* to request all attributes for u are treated as a single call. The adversary will only get the user keys and secret attribute keys after having submitted the complete set of attributes for the user. This means that an adversary is not able to adaptively request secret attribute keys to users, but needs to define the whole set of attributes that a user has at once. If an adversary wants to add attributes to a user he has already created, he can create a new key ring for the user by submitting another sequence of *CreateUser*, *RequestAttributeSK* calls. Note that this constraint prevents the attacker from utilizing the complete flexibility of the DABE scheme.

We also need to change the scheme to a selectively secure scheme, where the adversary submits the challenge access structure to the challenger in advance during an additional *Initialization* phase. The whole attack model is as follows:

Initialization: The adversary submits a challenge access structure $\mathbb{A}$ to the challenger that he wants to be challenged on.

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 public attribute keys, which the challenger creates by calling *RequestAttributePK*. The adversary also requests an arbitrary number of users by calling *CreateUser* and secret attribute keys for each user by calling *RequestAttributeSK*. However, he will only receive the secret user keys and secret attribute keys for each user after he has called *RequestAttributeSK* for all attributes that he wants the user to have. The adversary cannot request a set of secret attribute keys that would give a user the ability to satisfy $\mathbb{A}$. During the first request for a public or private key for an attribute of an authority a, the challenger creates the authority by a call to *CreateAuthority*; he stores the secret authority key for future use (but does not make the key available to the attacker).

Challenge: The adversary submits two messages M_0 and M_1. 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 along with secret attribute keys for them. He can not add secret attribute keys to users that he has already created, and as before, if any secret attribute key would give the respective user a set of attributes needed to satisfy $\mathbb{A}$, the challenger aborts.

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

As in the original security definition, the advantage of the adversary is defined as $\epsilon = \Pr[b' = b] - \frac{1}{2}$, where the probability is taken over all coin tosses of both challenger and adversary. The scheme is called non-adaptive key query secure if all polynomial time adversaries have at most a negligible advantage in the non-adaptive DABE game.

This modified model is from an adversary's point of view equal to the model of [14]. Thus, the proof of [14], Section 4.1, can directly be applied to the above construction.

4.2. Performance

Since ℓ pairings are needed for decryption, the complexity is obviously linear in the number of rows of the secret sharing matrix $\mathcal{M}$. Since $\ell \geq n$ (linear dependant columns can be eliminated), ℓ is a measure for the complexity of the matrix. In general, the size of $\mathcal{M}$ can grow exponentially in the size of the policy, but depending on the structure of the desired access policy, smaller matrices can be found.

5. Conclusion

CP-ABE is a promising concept for next-generation access control. To be usable in a pervasive environment, the extension of CP-ABE to settings which support multiple authorities is necessary. In this paper, we proposed a scheme where an arbitrary, non-static set of independent attribute authorities is able to issue attributes to users, taking as input only public user keys. A central trusted authority is only needed for the creation of users.

We also proposed a DABE construction that supports every possible access policy expressed in DNF and proved its CPA security in the generic group model. Furthermore, we showed how a recent CP-ABE construction can easily be extended to fit the DABE scheme, but can be proven secure only under a weaker attacker model than the one used in the first construction.

In both constructions, the size of the ciphertext might be exponential in the size of the policy, depending on its structure. However, in practical settings both constructions are likely to achieve small ciphertexts.

The proofs of both constructions put constraints on the attacker. The proof of the first constructions uses Shoup's generic group model, where the attacker cannot exploit weaknesses of the underlying groups. The second construction could only be proven non-adaptive key query secure. A construction that can be proven secure under the strong attacker model of Section 2.2 and which does not utilize the generic group model is left for future research.

References

[1] A. Beimel, *Secure schemes for secret sharing and key distribution*, Ph. D. thesis, Dept. of Computer Science, Technion, 1996.

[2] J. Bethencourt, A. Sahai, and B. Waters, *Ciphertext-policy attribute-based encryption*, IEEE Symposium on Security and Privacy, 321–334, 2007.

[3] D. Boneh, *A brief look at pairings based cryptography*, FOCS, 19–26, IEEE Computer Society, 2007.

[4] D. Boneh and X. Boyen, *Short signatures without random oracles and the SDH assumption in bilinear groups*, J. Cryptology **21** (2008), no. 2, 149–177.

[5] M. Chase, *Multi-authority attribute based encryption*, Theory of cryptography, 515–534, Lecture Notes in Comput. Sci., 4392, Springer, Berlin, 2007.

[6] L. Cheung and C. C. Newport, *Provably secure ciphertext policy ABE*, ACM Conference on Computer and Communications Security (Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, eds.), 456–465, ACM, 2007.

[7] S. D. Galbraith, K. G. Paterson, and N. P. Smart, *Pairings for cryptographers*, Discrete Appl. Math. **156** (2008), no. 16, 3113–3121.

[8] V. Goyal, A. Jain, O. Pandey, and A. Sahai, *Bounded ciphertext policy attribute based encryption*, ICALP, 2008.

[9] V. Goyal, O. Pandey, A. Sahai, and B. Waters, *Attribute-based encryption for fine-grained access control of encrypted data*, ACM Conference on Computer and Communications Security (Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, eds.), 89–98, ACM, 2006.

[10] S. Müller, S. Katzenbeisser, and C. Eckert, *Distributed attribute-based encryption*, 11th International Conference on Information Security and Cryptology, 2008, to appear.

[11] A. Sahai and B. Waters, *Fuzzy identity-based encryption*, Advances in cryptology—EUROCRYPT 2005, 457–473, Lecture Notes in Comput. Sci., 3494, Springer, Berlin, 2005.

[12] V. Shoup, *Lower bounds for discrete logarithms and related problems*, Advances in cryptology—EUROCRYPT '97 (Konstanz), 256–266, Lecture Notes in Comput. Sci., 1233, Springer, Berlin, 1997.

[13] D. R. Stinson, *An explication of secret sharing schemes*, Des. Codes Cryptography **2** (1992), no. 4, 357–390.

[14] B. Waters, *Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization*, Tech. report, SRI International, 2008, work in progress.

Sascha Müller
Technische Universität Darmstadt
Hochschulstr. 10
D – 64289 Darmstadt
E-mail address: mueller@sec.informatik.tu-darmstadt.de

Stefan Katzenbeisser
Technische Universität Darmstadt
Hochschulstr. 10
D – 64289 Darmstadt
E-mail address: katzenbeisser@seceng.informatik.tu-darmstadt.de

Claudia Eckert
Technische Universität Darmstadt
Hochschulstr. 10
D – 64289 Darmstadt
E-mail address: eckert@sec.informatik.tu-darmstadt.de