Technische Universität Darmstadt
Hochschulkennziffer D-17
Department of Computer Science
Security Engineering Group

Data-Centric Security with Attribute-Based Encryption

Dissertationsschrift in englischer Sprache
zur Erlangung des Grades eines Dr.-Ing.
an der Technischen Universität Darmstadt
eingereicht von

Dipl.-Inform. Sascha Müller
geboren 24.10.1978 in Langen (Hessen)

Erstreferent:	Prof. Dr. Stefan Katzenbeisser
Korreferent:	Prof. Dr. Michael Waidner
Tag der Einreichung:	29. August 2011
Tag der Prüfung:	11. Oktober 2011

Darmstadt, 2011

Zusammenfassung

In dieser Dissertation untersuchen wir verschiedene Aspekte der datenzentrierten Sicherheit. Insbesondere betrachten wir die attributbasierte Verschlüsselung (ABE), ein kryptographisches Primitiv, das es erlaubt, Dokumente mit Policies über Attributen zu verschlüsseln, so dass die Entschlüsselung nur für solche Subjekte möglich ist, deren Attributsmengen die Verschlüsselungspolicies erfüllen.

Das Hauptziel dieser Arbeit ist zu zeigen, auf welche Weise datenzentrierte Sicherheit in praktischen Anwendungsgebieten einsetzbar ist. Hierzu erweitern wir zunächst ABE für dynamische und verteilte Szenarien, indem wir die sogenannte *Distributed Attribute-Based Encryption (DABE)* einführen. DABE erlaubt es nicht nur, dass Subjekte ihre Attribute inkrementell über die gesamte Laufzeit des Systems anfordern (anders als bei konventioneller ABE, wo alle Attribute eines Subjekts gemeinsam angefordert werden müssen); es ist dadurch auch möglich, dass diese Attribute von einer beliebigen Anzahl voneinander unabhängiger *Attribute Authorities* bereitgestellt werden, von denen jede ihr eigenes Universum von Attributen verwaltet. Wir stellen zwei Konstruktionen für das DABE-Schema vor. Eine davon ist zusätzlich effizienter als jede andere heute bekannte ABE Konstruktion.

Die zweite Innovation dieser Arbeit ist ein neuartiges Konzept, das die Privatheit in ABE verbessert, indem es Verschlüsselungspolicies versteckt. Um dies zu erreichen führen wir den Begriff der *Policy-Anonymität* ein, definieren ihn formal und diskutieren ihn. Mithilfe einer Methode aus der Graphentheorie zeigen wir anschließend, wie ein hoher Grad von Policy-Anonymität in der Praxis erreicht werden kann, indem man eine bekannte ABE-Konstruktion erweitert. Die komplette erweiterte Konstruktion wird vorgestellt und ein Sicherheitsbeweis wird geführt.

Wir beschäftigen uns außerdem mit konkreten Anwendungsmöglichkeiten für ABE. So entwickeln wir ein DRM-Framework, das ABE nutzt, um den Prozess der Lizenzerzeugung zu vereinfachen und dabei gleichzeitig die Anforderungen an die Vertrauenswürdigkeit zu verringern. Anschließend beschreiben wir ein Tool, das in der Lage ist, kryptographisch erzwingbare Komponenten aus Policies zu bestimmen, die in der Open Digital Rights Language (ODRL) beschrieben sind.

Zum Schluss der Arbeit demonstrieren wir wie ABE in Service-Orientierte Architekturen (SOA) integriert werden kann. Hierzu identifizieren wir die relevanten

Web Service Standards und erweitern diese, um durch ABE geschützte SOAP Nachrichten zu unterstützen. Ferner beschreiben wir die Implementierung von Web Services, die zur Umsetzung eines vollständigen DABE Frameworks notwendig sind. Dadurch wird ein Framework ermöglicht, das genutzt werden kann um bestehende SOAs so zu erweitern, dass sie die verbesserten Sicherheitsgarantien erfüllen können, die durch datenzentrierte Sicherheitstechnologie ermöglicht werden.

Summary

In this thesis we examine several aspects of data-centric security. In particular, we take a look at Attribute-Based Encryption (ABE), a cryptographic primitive that allows to encrypt documents with policies over attributes and allows decryption only by parties possessing sets of attributes that satisfy the encryption policies.

Our primary goal is to show the applicability of data-centric security to practical scenarios. We first extend ABE to dynamic and distributed settings, introducing what we call *Distributed Attribute-Based Encryption (DABE)*. DABE not only allows parties to claim their attributes incrementally throughout the lifetime of a system (unlike conventional ABE where all attributes must be claimed at once), but also supports these attributes to be managed by an arbitrary number of independent attribute authorities, each of them having control over its own universe of attributes. We give two constructions of DABE schemes, one of which is also more efficient than any ABE scheme known today.

Our second contribution is a novel concept that improves privacy in ABE by hiding the encryption policy. To this end, we introduce, define and discuss *policy anonymity*. Using an idea from graph theory we then show how a high degree of policy anonymity can be achieved by extending a known ABE construction. The complete construction along with security proofs is given.

We also discuss how ABE can be utilized in practical settings. We develop a new DRM framework using ABE that offers a simplified license creation process while requiring less trust. We then describe an extraction tool that is able to determine cryptographically enforceable components of policies in the Open Digital Rights Language (ODRL). Finally, we demonstrate how ABE can be integrated into Service Oriented Architectures (SOA), showing how common Web Service standards can be used to support ABE encrypted SOAP messages and describing implementations of web services to build a complete DABE framework. This resulting framework can be used to extend existing SOAs in order to support the improved security guarantees offered by data-centric security technology.

Acknowledgements

First and foremost I want to thank my supervisor Prof. Dr. Stefan Katzenbeisser. It has been an honour to be the first of his PhD students to graduate. He not only taught me about good research, but also about good thinking and the ability to present results comprehensibly. His invaluable suggestions had a great impact on how I approached and solved the challenges imposed by my work.

I am grateful to have been part of a great working environment at CASED, thanks to my colleagues (in alphabetical order) Sami Alsouri, Sebastian Biedermann, Dr. Wolfgang Böhmer, Martin Franz, Dmitry Kravchenko, Heike Meissner, Dr. Martin Mink, Cuong Hieu Nguyen, Andreas Peter, Bertram Poettering, Andrea Püchner, and Heike Schröder. The Security Engineering group has grown considerably during my time, and it has been a great pleasure to be part of that process.

I am also grateful to my original supervisor Prof. Dr. Claudia Eckert who unfortunately left Darmstadt shortly after my research gained momentum. She was the one who got me interested in research when I was still a student, let me work at Fraunhofer SIT where I did my diploma thesis and later gave me the opportunity to work as a PhD student at TU Darmstadt.

I dedicate my special thanks to my mother for her support through the years and to all my friends for being there to take my mind off work from time to time. Last but not least I want to thank Milena Foerster for a most valuable friendship, understanding and believing in me, as well as her family, whom I have grown more accustomed to than I had anticipated.

Darmstadt, 24. October 2011

Sascha Müller

Contents

Part I.

Introduction

Chapter 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 promising approaches is Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [BSW07]. 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.

In this thesis we are concerned with several aspects of CP-ABE, mainly based on the following papers that were published in international conferences and journals:

- Sascha Müller, Stefan Katzenbeisser, and Claudia Eckert, *Distributed Attribute-Based Encryption*, 11th International Conference on Information Security and Cryptography, ICISC 2008 (Pil Joong Lee and Jung Hee Cheon, eds.), Lecture Notes in Computer Science, vol. 5461, Springer, 2008, pp. 20–36;

- Sascha Müller, Stefan Katzenbeisser, and Claudia Eckert, *On Multi-Authority Ciphertext-Policy Attribute-Based Encryption*, Bulletin of the Korean Mathematical Society (B-KMS) **46** (2009), no. 4, 803–819;

- Sascha Müller and Stefan Katzenbeisser, *Hiding the Policy in Cryptographic Access Control*, 7th International Workshop on Security and Trust Management (STM'11), 2011. to appear;

- Sascha Müller and Stefan Katzenbeisser, *A New DRM Architecture With Strong Enforcement*, ARES, IEEE Computer Society, 2010, pp. 397-403;

as well as some minor contributions from several student theses that were developed under the author's supervision.

Roadmap and Contributions

This thesis is organized as follows: Chapter 2 introduces the basic concepts needed later in the thesis. These include access control, secret sharing, attribute-based encryption and various mathematical notions.

The first major contribution of this thesis is the concept of *Distributed Attribute-Based Encryption (DABE)*, which is introduced and discussed in detail in Chapter 3. In this chapter, we explain the motivation and general concept, give two cryptographic constructions that implement this powerful idea and discuss their security properties. One of these two constructions is very efficient (in fact, more efficient than any other CP-ABE construction known today) but not very expressive, while the other one is very expressive, but not as efficient. We also describe an implementation and give performance results for it.

Chapter 4 approaches another challenge that current CP-ABE constructions face. The contribution of this chapter is twofold: The primary result is a novel CP-ABE construction that allows to obfuscate policies such that they can be considered anonymous in the sense of k-anonymity [CdVFS07]. To this end, we explore a new method from graph theory that we call *Syntax Tree Majors* and show how to use such trees in CP-ABE in order to achieve policy anonymity. As a secondary result, the construction we give also shows how to extend the expressiveness of certain CP-ABE schemes. This chapter concludes the theoretical part of the thesis.

While a lot of different CP-ABE constructions have been published, the practical application of such schemes to real-world scenarios is under developed in the research community. We contribute novel research in this area in Chapter 5, which shows how to use CP-ABE to improve *Enterprise Rights Management (ERM)* scenarios. By shifting parts of the enforcement process to CP-ABE, the personalization of player keys is made easier, while also reducing the trust into the DRM viewer.

Our DABE concept is especially suited for very distributed settings. One such setting are *Service Oriented Architectures (SOA)*. Thus, we describe how to implement core features of DABE in the form of web services in Chapter 6.

Finally, Chapter 7 summarizes the results and concludes the thesis.

Chapter 2

Basics

2.1. Access Control

Access control, one of the most central topics in IT security, is concerned with the question of how to restrict access to objects that are worth protecting (*assets*) such that a need-to-know-principle is adhered to, i.e., no subject learns more than he needs to. To this end, access rules are formulated for each object, usually by its owner. These rules specify under which restrictions access to the object is to be granted.

More formally, let O be the set of all objects of a system, and S the set of all subjects, i.e., objects that want to actively access some of the objects. An access rule for an object $o \in O$ is a function $f_o : S \to \{0, 1\}$ such that $f_o(s)$ returns 1 if $s \in S$ is allowed to access $o \in O$, and 0 otherwise. We call the set $\Pi(o) := \{s \in S \mid f_o(s) = 1\}$ the set of authorized subjects of o. The union of all subjects with access rules $\bigcup_o \Pi(o)$ is called the set of *principals* [Gol06].

Frequently, there is a further restriction describing the access operation, which, for example, could be a read access or a write access. In this work, we are only concerned with read accesses.

The primary focus of this thesis is the enforcement of access rules. Usually, there is a trusted party that has full control over the assets, called the *reference monitor* [GGKL89]. It enforces rules that it receives from the owners of the objects. The reference monitor gets active on every access attempt, determines if the accessing subject has the right to access the object (i.e., $f_o(s) \stackrel{?}{=} 1$), and if so, allows the

access. The high demands imposed on such a party, especially trustworthiness, full control over all objects, and good performance even with many object access requests, motivate the question to construct other ways to enforce access rules.

One promising attempt is *information-centric security*, which binds security directly to data and the people who access it [RSA11]: Instead of having a trusted party in the form of a gateway controlling all accesses, the information essentially protects itself. Note that throughout this work we will use the term *data-centric security* instead, as it appears to be more fitting from a technical point of view.

Cryptographically enforced access control A simple approach to obtain data-centric security is to use encryption: We can encrypt each object o with an encryption key E_o and give the corresponding decryption key D_o to all authorized subjects, i.e., elements of $\Pi(o)$. Thus, the authorized subjects of o can access it (by decrypting it), while all other subjects can not. In such an approach, everybody who releases an object has full control over who is able to access it since he controls the distribution of D_o. Furthermore, there needs to be no trusted party to enforce access rules. They are essentially self-enforcing. Basically, the effect of such an architecture is that instead of protecting the assets, it is the keys D_o that need protection. This also moves the computational cost of access control from the trusted party to the decrypting parties.

However, it is obvious that this approach does not scale very well. The owners of the objects need to create many encryption/decryption key pairs, and the more open and distributed a setting is, the harder it gets to distribute the keys to all allowed subjects. Attribute-Based Encryption attempts to solve these challenges.

2.2. Attribute-Based Encryption

In Attribute-Based Encryption (ABE), ciphertexts and/or user keys are associated with policies that describe who is allowed to access the encrypted information. Specifically, in *Key-Policy Attribute-Based Encryption (KP-ABE)* ciphertexts are encrypted with a set of attributes and each user's secret key is associated with a policy describing which ciphertexts he can decrypt (see Figure 2.1). Such a policy is a predicate over the set of attributes, usually formulated as a Boolean formula.

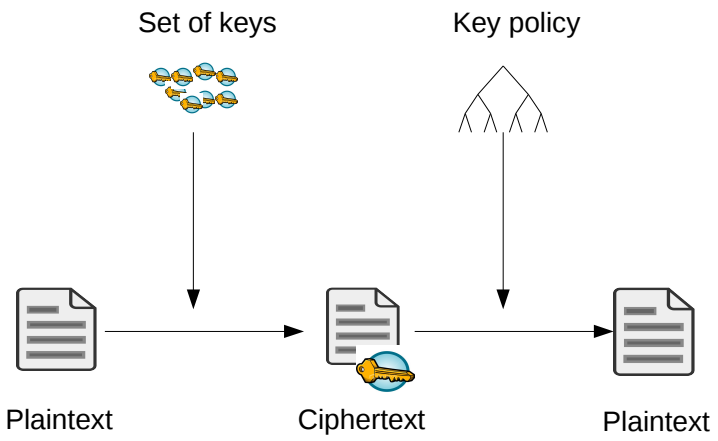


Figure 2.1: Schematic overview of KP-ABE

Conversely, in *Ciphertext-Policy Attribute-Based Encryption (CP-ABE)* a ciphertext is encrypted with a policy. Anyone whose attributes satisfy the policy can decrypt the ciphertext; otherwise the decryption fails (see Figure 2.2). In a nutshell, in CP-ABE a policy is applied during encryption and in KP-ABE a policy is applied during decryption.

An *attribute* is a property or feature that a subject may have. At some point in time, any subject may become *eligible* for a particular attribute, meaning that it now has the respective property or feature. It then receives a token from a trusted party called *attribute authority* that testifies his eligibility and can be used by him to prove that he has the property or feature that the corresponding attribute represents.

An attribute is usually represented as a string. For example, an attribute called `isAdmin` could be used to describe subjects that are administrators of a certain domain. We denote the set of all attributes used in a specific domain as the *universe of attributes*. In CP-ABE, policies over the universe of attributes are formulated for each object to describe what prerequisites a subject must have to access it.

The idea of using policies for encryption was first proposed as a primitive called *Policy-Based Cryptography* by Bagga and Molva [BM05, BMC06], in which – similarly to Ciphertext-Policy Attribute-Based Encryption – policies where used as encryption/decryption keys. However, their schemes have a crucial drawback:

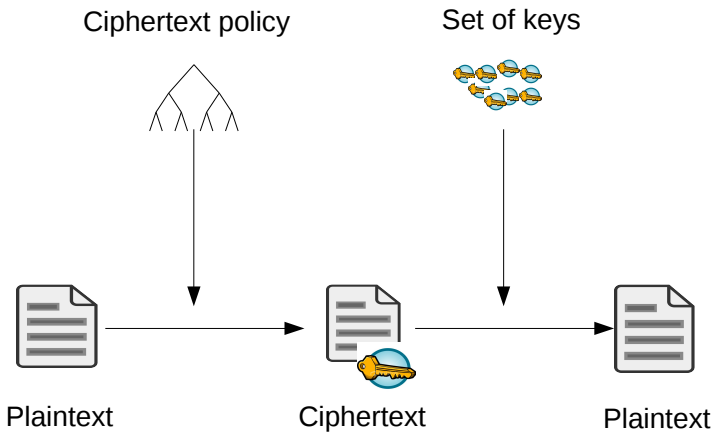


Figure 2.2: Schematic overview of CP-ABE

They are not secure against collusions, i.e., users with different sets of attributes are able to combine their sets to collaboratively decrypt ciphertexts that were encrypted with policies they would not be able to satisfy individually. We note that later the authors managed to circumvent this problem by also encrypting the ciphertexts with the public key of a receiver [BM06]. This, however, neglects the most important property of this approach, namely to encrypt data with a policy *instead* of a user or group key.

The notion of *Attribute-Based Encryption* was first introduced in 2005 by Sahai and Waters [SW05] as an application of their idea of *Fuzzy Identity-Based Encryption*. Here, each user key is associated with a set of attributes ω, and each ciphertext is associated with a set of attributes ω'. A decryption is possible, whenever a user's ω overlaps a ciphertext's ω' in at least d attributes, where d is a fixed value decided on during system setup, i.e., $|\omega \cap \omega'| \geq d$.

In the following year, Goyal et al. [GPSW06] proposed the first expressive KP-ABE scheme (that allowed key policies in the form of logical formulas instead of simple thresholds), which was followed by the first CP-ABE scheme by Bethencourt et al. [BSW07] in the next year. Bethencourt's construction could only be proved secure in the Generic Group Model, and it has since become apparent that CP-ABE constructions with reduction proofs are hard to construct. In 2007 Cheung and Newport [CN07] gave a CP-ABE construction with a reduction proof, but it was restricted to policies in DNF notation. Since then, numerous CP-ABE

schemes have been proposed with varying features that support different types of policy languages [Cha07, CN07, LCLS08, GJPS08, AI09, BKP09, LCLX09, NYO09, LOS+10, LW11, ZH10, YWRL10b].

We will now describe the CP-ABE scheme as a 4-tuple of probabilistic polynomial-time (PPT) algorithms as originally proposed in [BSW07][1]:

$(PK, MK) \leftarrow$ **Setup**(1^k)**.** The setup algorithm takes no input other than the security parameter 1^k. It outputs the public parameters PK and a master key MK.

$CT \leftarrow$ **Encrypt**$(PK, M, \mathbb{A})$**.** The encryption algorithm takes as input the public parameters PK, a message M, and an access structure $\mathbb{A}$ over the universe of attributes. The algorithm will encrypt M and produce a ciphertext CT such that only a user who possesses a set of attributes that satisfies the access structure will be able to decrypt the message. We will assume that the policy $\mathbb{A}$ is usually sent along with the ciphertext CT, as it is needed for decryption.

$SK_L \leftarrow$ **KeyGen**(MK, L)**.** The key generation algorithm takes as input the master key MK and a set of attributes L that describe the key. It outputs a private key SK_L that contains a set of keys corresponding to the attributes of L.

$\{M, \texttt{NULL}\} \leftarrow$ **Decrypt**(PK, CT, SK_L)**.** The decryption algorithm takes as input the public parameters PK, a ciphertext CT, which contains an access policy $\mathbb{A}$, and a private key SK_L that is a private key for a set L of attributes. If the set L of attributes satisfies the access structure $\mathbb{A}$ then the algorithm will decrypt the ciphertext and return a message M; otherwise the algorithm returns `NULL`.

CP-ABE allows to enforce access rules in many practical scenarios. For example, in the popular *Role-based Access Control (RBAC)* approach, users are assigned to *roles* and each user's roles determine which rights he has. CP-ABE can be used to efficiently enforce access rights in an RBAC scenario: For each role there is an attribute, and for each role a user possesses, he receives the corresponding attribute.

[1] We omit the description of the additional algorithm *Delegate* in [BSW07] as it will not be needed in the rest of this thesis and is not implemented in most CP-ABE constructions.

```
(user.adult (= true) OR
creditcard = verified)
    AND (
    (contprov1.article1234.status = purchased AND
    contprov1.account = balanced)
     OR
    (contprov2.article1234.status = purchased AND
    contprov2.account = balanced)
)
```

Figure 2.3: An example policy

Access rights are described as logical formulas over the universe of attributes. For example, if data is encrypted with a policy `RoleA AND (RoleB OR RoleC)`, every user who is active in the role `RoleA` and also in either `RoleB` or `RoleC` (or both) can decrypt the data.

As another example, consider a company that hosts DRM protected media files. Users can purchase licenses from various content providers that issue usage licenses containing keys required to decrypt the protected files. Let us assume that two such content providers are `contprov1` and `contprov2`. A usage license could be expressed as a Boolean formula over attributes. For example, the policy could state that the protected file should only be decrypted by someone who has purchased licenses from at least one of the given content providers and is authenticated as an adult (see Figure 2.3). Here, we use the fact that a subject may become eligible of an attribute at some point in time, i.e., he has to purchase the file before he gets eligible of the attribute `contprov1.article1234.status = purchased`. It is also possible to automatically extract such policies from policies in Open Digital Rights Language (ODRL), as we will show in Chapter 5.

Note that in both examples, rules are enforced automatically by the cryptographic construction, and no trusted entity is required to manually enforce the policies on every access. Also, the access rules may be very complex allowing for elaborate, fine-grained access control if desired by the scenario.

Many variations of CP-ABE are thinkable. For, example, note that the key SK_L that is received via *KeyGen* contains a set of private keys corresponding to the attributes L that the user is eligible for, and the *KeyGen* algorithm is executed once to claim the full set of these private keys for a user. In the open, distributed scenario that we consider interesting for CP-ABE applications, this is not desirable,

as the number of possible attributes is very large, and users may later become eligible of attributes that they were not eligible for before (as seen in the DRM example).

To meet these challenges, we introduce *Distributed Attribute-Based Encryption*, where an arbitrary number of *Attribute Authorities* manage and issue keys, and users can query for keys at any time. In a way, CP-ABE can be seen as a special case of DABE with only a single authority.

Similarly, in other scenarios, Broadcast Encryption (BE) is used to restrict access to media by cryptographic means (see for example [FN93]). In BE, a media is encrypted in a way that only certain subsets of users can decrypt. In contrast to ABE, BE schemes are typically designed to later revoke users that for some reason lose their right to access an object. Clearly, there are some connections between BE and ABE and combinations of the two seem promising. For example, BE has been used to add a revocation mechanism to ABE schemes [AI09]. Another example is Attribute-Based Broadcast Encryption [JK10, ZH10], in which the eligible subsets of users may be refined by attributes. However, all participants must be known by their identity when the system is set up, whereas in ABE, user identities can only be modeled indirectly through attributes (for the purpose of formulating access rules).

2.3. Mathematical Foundations

2.3.1. Pairing-Based Cryptography

Throughout this work we will use *cryptographic pairings* as building blocks upon which we build our constructions. Generalizing the pairing definition from [Bon07], we define:

Definition 2.1 (Pairings)**.** *Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic groups of order p. A function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a* pairing *if*

1. *the group operations in $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ can be computed efficiently;*
2. *the function e is bilinear, i.e., for all $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$ it holds that:*

$$e(g_1^a, g_2^b) = e(g_1, g_2)^{ab};$$

3. *e can be computed efficiently;*
4. *e is non-degenerate, i.e., there are* $g_1 \in \mathbb{G}$ *and* $g_2 \in \mathbb{G}_2$*, such that* $e(g_1, g_2) \neq 1$.

There are three types of pairings [GPS08]: If $\mathbb{G}_1 = \mathbb{G}_2$, then the pairing is called *symmetric*. Asymmetric pairings, where $\mathbb{G}_1 \neq \mathbb{G}_2$, can be classified further into ones where there is an efficiently computable homomorphism $\phi : \mathbb{G}_1 \to \mathbb{G}_2$ and ones which have no such homomorphism. In this thesis, we will not use pairings of the latter type.

For symmetric pairings, we will usually write all elements of $\mathbb{G} := \mathbb{G}_1 = \mathbb{G}_2$ and $\mathbb{G}_T$ in the form $A = g^a$ or $A = g_T^a$, where $g \in \mathbb{G}$ is a generator of $\mathbb{G}$ and $g_T := e(g, g) \in \mathbb{G}_T$ is a generator of $\mathbb{G}_T$. We assume that for all groups the *Computational Diffie-Hellman Assumption (CDH)* holds, i.e., given (g, g^a, g^b) it is hard to compute g^{ab} and given (g_T, g_T^a, g_T^b) it is hard to compute g_T^{ab}. Note that due to the properties of pairings, the *Decisional Diffie-Hellman Assumption (DDH)* does *not* hold in $\mathbb{G}$, i.e., given (g, g^a, g^b, g^c) it is easy to determine if $c \stackrel{?}{=} ab$ by comparing $e(g^a, g^b) = e(g, g)^{ab}$ with $e(g, g^c) = e(g, g)^c$. There is, however, a variant of the DDH which is considered hard in pairing groups, and often used for security reductions. The *decisional Bilinear Diffie-Hellman (d-(BDH))* problem is defined as follows:

Definition 2.2 (Decisional Bilinear Diffie-Hellman Problem)**.** *For a symmetric pairing* $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ *with generator* $g \in \mathbb{G}$*, given* $g^a, g^b, g^c \in \mathbb{G}$ *and* $Z \in \mathbb{G}_T$*, decide if* $Z = e(g, g)^{abc}$*.*

An algorithm $\mathcal{B}$ that outputs a bit $\{0, 1\}$ has advantage ϵ in solving d-(BDH) in $\mathbb{G}$ if

$$\left|\Pr\left[\mathcal{B}(g, g^a, g^b, g^c, Z = e(g, g)^{abc}) = 0\right] - \Pr\left[\mathcal{B}(g, g^a, g^b, g^c, Z \neq e(g, g)^{abc}) = 0\right]\right| \geq \epsilon\,,$$

where the randomness is over the random bits consumed by $\mathcal{B}$.

We say that the *decisional Bilinear Diffie-Hellman assumption* holds if no PPT algorithm has a non-negligible advantage in solving the decisional BDH problem.

In cryptography, pairings were first used by Alfred Menezes et al. [MOV93] to attack elliptic curve cryptosystems. Later, Joux [Jou00] showed that pairings can

be utilized to implement a one-round Diffie-Hellman-like key exchange protocol for three parties using a pairing. We briefly describe a variant of this protocol using a symmetric pairing: Let $\mathbb{G} := \mathbb{G}_1 = \mathbb{G}_2$ be groups of order p and let $g \in \mathbb{G}$ a generator of $\mathbb{G}$. Each of the three parties chooses a random number, i.e., $a, b, c \in \mathbb{Z}_p$. Each party then raises g to the power of its secret number, i.e., $A := g^a, B := g^b$, and $C := g^c$ and publishes that value. Now each of the parties can compute the same secret key using its secret number and the published values from the other parties, because $e(A, B)^c = e(A, C)^b = e(B, C)^a = e(g, g)^{abc}$. This key is a shared secret of the three parties. The task of an attacker is to recover this shared secret using only the public keys A, B, and C, but neither a, b, or c. It is easy to see that this protocol is secure under the d-(BDH) assumption.

However, the birth of pairing-based cryptography has to be primarily attributed to Dan Boneh and Matt Franklin [BF03], who used pairings to solve a problem that had been open for over 15 years, namely that of *Identity Based Encryption*, thus opening a completely new field of cryptographic research.

2.3.2. Implementations

In this thesis we are not concerned with the implementation and inner workings of pairings, but we will use them in the form of black boxes. However, we need to understand the implications and demands of existing implementations of pairings: Computing pairings is quite expensive. The fastest known pairing to date is the η_T pairing (including some variants of it, like the R-ate pairing), which in its fastest known implementation requires about 2-3 million clock cycles on modern CPUs [BGDM$^+$10]. There also exist fast hardware implementations [BDF$^+$10] and pairings have even been used in smartcards [SCA06] and RFIDs [IOIO07]. The arguably most mature software implementation of pairings is the PBC library [Lyn] that was developed by Ben Lynn during his PhD thesis [Lyn07] and has since been ported to many languages besides the original C implementation, e.g., C++, Java, Python, and Perl. Whenever we implement cryptographic algorithms, we will use this library.

2.4. CP-ABE Proofs

When building CP-ABE constructions we are primarily interested in their security against Chosen Plaintext Attacks (CPA). As noted in – for example – [CN07], CPA secure CP-ABE schemes can be extended to CCA secure schemes by sending a one-time signature along with each encryption using an approach that is similar to [BCHK07]. Recently, Yamada et al. [YAHK11] examined this idea more closely and found that such an extension works for all well-known CP-ABE schemes. We will not go into details regarding this process, and neither do we claim CCA security of our proposed constructions. We will, however, prove CPA security. In this section, we briefly review security definitions specific to CP-ABE constructions, and most importantly introduce the proof framework that we will use throughout this thesis.

2.4.1. CPA Security Game

The CPA-Security game for CP-ABE is defined as follows

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

Phase 1. The adversary queries the challenger for private keys corresponding to lists of attributes L. Each time, the challenger answers with a secret key SK_L for L.

Challenge. The adversary submits two messages M_0 and M_1 of equal length as well as the challenge access structure $\mathbb{A}$, with the restriction that none of the keys he received in Phase 1 satisfies $\mathbb{A}$. The challenger chooses $b \xleftarrow{R} \{0,1\}$ and encrypts M_b with $\mathbb{A}$. The resulting ciphertext CT is given to the adversary.

Phase 2. Same as Phase 1. However, the adversary can only query for keys that do not satisfy $\mathbb{A}$.

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

An algorithm $\mathcal{B}$ that outputs a bit $\{0,1\}$ has advantage ϵ in solving the game if

$$|\Pr[\mathcal{B}(b' = b) = 0] - \Pr[\mathcal{B}(b' \neq b) = 0]| \geq \epsilon\,,$$

where the randomness is over the random bits consumed by $\mathcal{B}$.

It is noteworthy that nearly all CP-ABE constructions are only proven secure in a *selective* version of this game, which has an additional *Init* phase and more restrictive requirements:

Init. The adversary chooses the challenge access structure $\mathbb{A}$.

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

Phase 1. The adversary queries the challenger for private keys corresponding to lists of attributes L. Each time, the challenger answers with a secret key SK_L for L. However, the adversary can only query for keys that do not satisfy $\mathbb{A}$.

Challenge. The adversary submits two messages M_0 and M_1 of equal length. The challenger chooses $b \xleftarrow{R} \{0,1\}$ and encrypts M_b with $\mathbb{A}$. The resulting ciphertext CT is given to the adversary.

Phase 2. Same as Phase 1.

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

However, in all proofs that will be given in the context of this thesis, we use the stronger, non-selective game without the Init phase.

2.4.2. The General Diffie-Hellman Exponent Problem

To prove the security of cryptographic constructions, it is common to use security reductions to simple mathematical assumptions. To do this, a *simulator* maps the elements (i.e., the input and output variables) of the construction to the elements of the assumption, showing that breaking the construction implies breaking the assumption. In CP-ABE this approach turns out to be very difficult, because — as Waters noted in [Wat11] — the elements of CP-ABE include complex access policies that cannot easily be mapped to the short, fixed number of parameters of simple assumptions like the decisional Bilinear Diffie-Hellman assumption (see Definition 2.2).

This problem led to the introduction of several new and often rather complex assumptions that were often tailored to specific crytographic ABE constructions. The elements of these new assumptions mirror the structure of the elements used in the respective CP-ABE constructions. To provide evidence that these new assumptions were correct, the Generic Group Model [Sho97] was used frequently. This helped to demonstrate the hardness of the invented problem. More recently, some constructions were published that use the Generic Group Model directly to show their security. Note that the provability of a protocol in the Generic Group Model does not imply that there is also a security reduction to *any* standard assumption possible for that protocol, as shown in [Pas11].

We will now give a brief introduction to the Generic Group Model developed in [BB08] as it is used in pairing-based cryptography, which we simplify slightly for the symmetric case where $\mathbb{G}_1 = \mathbb{G}_2 =: \mathbb{G}$.

In the generic group model, the adversary is given only encoded versions of all group elements, that 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 $\xi_T(a \cdot b)$. 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.

Due to the random choice of ξ and ξ_T, two polynomials that evaluate to different values over $\mathbb{Z}_p$ yield different encodings when mapped by ξ and ξ_T, except if

1. for a vector $(x_1, \ldots, x_n)$ and some polynomials p_i, p_j that can be constructed using the oracles it holds that $(p_i - p_j)(x_1, \ldots, x_n) = 0$ although $(p_i - p_j) \neq 0$, or

2. due to the choice of the random encodings two different values "accidentally" are mapped to the same string. This can be circumvented by programming the oracles such that they are injective, i.e., $x \neq y \Rightarrow \xi(x) \neq \xi(y)$.

By these preliminaries a construction can be proven secure as follows: If it can be shown that the probability of the first incident occurring is negligible, then an attacker can only compute the terms that he gains by combining the terms that he knows by using the group oracles. If he cannot compute a term that would allow him to win the security game in question, he cannot win it at all.

The General Diffie-Hellman-Exponent assumption as introduced in Appendix A.2 of [BBG05] is a framework that formalizes these ideas and thus allows to show the security of a construction by observing the terms given to the adversary. It is crucial to all our security proofs, so we repeat the important definitions and theorems here.

First we need to capture the properties of the group oracles by the following definition:

Definition 2.3 (Dependent Polynomials)**.** *Let $\hat{P}, \hat{Q} \in \mathbb{F}_p[x_1, \ldots, x_n]^s$ be two s-tuples of n-variate polynomials over $\mathbb{F}_p$. Write $\hat{P} = (p_1, p_2, \ldots, p_s)$ and $\hat{Q} = (q_1, q_2, \ldots, q_s)$, where $p_1 = q_1 = 1$. We say that a polynomial $f \in \mathbb{F}_p[x_1, \ldots, x_n]$ is dependent on the sets $(\hat{P}, \hat{Q})$ if there exist $s^2 + s$ constants $\{a_{i,j}\}_{i,j=1}^s$, $\{b_k\}_{k=1}^s$ such that*

$$f = \sum_{i,j=1}^{s} a_{i,j} p_i p_j + \sum_{k=1}^{s} b_k q_k \,.$$

We say that f is independent of $(\hat{P}, \hat{Q})$ if f is not dependent on $(\hat{P}, \hat{Q})$.

This definition describes how an adversary can combine the terms that he knows by calling the group oracles for multiplication, division and the pairing operation.

In our proofs, $\hat{P}$ and $\hat{Q}$ are all polynomials that are given to the attacker, and f is a polynomial of $\mathbb{G}_T$ that he has to construct to break the security of the respective construction. His task is to distinguish between $\xi_T(f(x_1,\ldots,x_n))$ and a random value. If he succeeds then, under the Generic Group Assumption, he can distinguish between $g_T^{f(x_1,\ldots,x_n)}$ and a random value of $\mathbb{G}_T$. Formally:

Definition 2.4 (Decision $(\hat{P},\hat{Q},f)$-Diffie-Hellman Problem)**.** *Let $\hat{P},\hat{Q} \in \mathbb{F}_p[x_1,\ldots,x_n]^s$ be two s-tuples of n-variate polynomials over $\mathbb{F}_p$. Given a generator $g \in \mathbb{G}, g_T = e(g,g)$ and the vector*

$$H(x_1,\ldots,x_n) = \left(g^{\hat{P}(x_1,\ldots,x_n)}, g_T^{\hat{Q}(x_1,\ldots,x_n)}\right) \in \mathbb{G}^s \times \mathbb{G}_T^s$$

distinguish $g_T^{f(x_1,\ldots,x_n)}$ (with $f \in \mathbb{F}_p[x_1,\ldots,x_n]$) from a random value $T \in \mathbb{G}_T$.

We say that an algorithm $\mathcal{B}$ that outputs $b \in \{0,1\}$ has advantage ϵ in solving the Decision $(\hat{P},\hat{Q},f)$-Diffie-Hellman Problem in $\mathbb{G}$ if

$$\left|\Pr\left[\mathcal{B}(H(x_1,\ldots,x_n), g^{f(x_1,\ldots,x_n)}) = 0\right] - \Pr\left[\mathcal{B}(H(x_1,\ldots,x_n), T) = 0\right]\right| > \epsilon\,,$$

where the probability is over the random choice of the generator $g \in \mathbb{G}$, the random choice of $x_1,\ldots,x_n$ in $\mathbb{F}_p$, the random choice of $T \in \mathbb{G}_T$, and the random bits consumed by $\mathcal{B}$.

In [BBG05], the following theorem is shown which shows that the adversary advantage is negligible if f is independent of $(\hat{P},\hat{Q})$:

Theorem 2.1 ([BBG05])**.** *For a generic bilinear group of order p with oracles ξ, ξ_T let $\hat{P},\hat{Q}$, and f defined as above and let $d = \max(2\deg(\hat{P}), \deg(\hat{Q}), \deg(f))$. If f is independent of $(\hat{P},\hat{Q})$, then for any adversary A that makes a total of at most q queries to the oracles computing the group operations in $\mathbb{G}$, $\mathbb{G}_T$ and the pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ we have:*

$$\left| Pr\left[A\begin{pmatrix} p, \xi(\hat{P}(x_1,\ldots,x_n)), \\ \xi_T(\hat{Q}(x_1,\ldots,x_n)), \\ \xi_T(t_0), \xi_T(t_1) \end{pmatrix} = b : \begin{matrix} x_1,\ldots,x_n, y \xleftarrow{R} \mathbb{F}_p, \\ b \xleftarrow{R} \{0,1\}, \\ t_b \longleftarrow f(x_1,\ldots,x_n), \\ t_{1-b} \longleftarrow y \end{matrix} \right] - \frac{1}{2} \right| \leq \frac{(q+2s+2)^2 \cdot d}{2p}$$

Proof. See Theorem $A.2$ in [BBG05]. □

Corollary 2.1 ([BBG05]). *Let $\hat{P}, \hat{Q} \in \mathbb{F}_p[x_1, \ldots, x_n]^s$ be two s-tuples of n-variate polynomials over $\mathbb{F}_p$ and let $F \in \mathbb{F}_p[x_1, \ldots, x_n]$. Let $d = \max(2\deg(\hat{P}), \deg(\hat{Q}), \deg(f))$. If f is independent of $(\hat{P}, \hat{Q})$, then any $\mathcal{A}$ that has advantage $1/2$ in solving the decision $(\hat{P}, \hat{Q}, f)$-Diffie-Hellman-Problem in a generic bilinear group $\mathbb{G}$ must take time at least $\Omega(\sqrt{p/d} - s)$.*

In Chapter 4 we will also use the asymmetric case:

Definition 2.5 (Dependant Polynomials, Asymmetric case). *Let P, Q, R be three s-tuples of n-variate polynomials over $\mathbb{F}_p$, i.e., $P, Q, R \in \mathbb{F}_p[x_1, \ldots, x_n]^s$. Write $P = (p_1, p_2, \ldots, p_s)$, $Q = (q_1, q_2, \ldots, q_s)$ and $R = (r_1, r_2, \ldots, r_s)$ where $p_1 = q_1 = r_1 = 1$. We say that a polynomial $f \in \mathbb{F}_p[x_1, \ldots, x_n]$ is dependent on the sets (P, Q, R) if there exist $2s^2 + s$ constants $\{a_{i,j}\}_{i,j=1}^s, \{b_{i,j}\}_{i,j=1}^s, \{c_k\}_{k=1}^s$ such that*

$$f = \sum_{i=1}^{s}\sum_{j=1}^{s} a_{i,j} p_i q_j + \sum_{i=1}^{s}\sum_{j=1}^{s} b_{i,j} q_i q_j + \sum_{k=1}^{s} c_k r_k$$

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

Boneh et al. prove the following theorem:

Theorem 2.2 ([BBG05]). *Let $P, Q, R \in \mathbb{F}_p[x_1, \ldots, x_n]^s$ be three s-tuples of n-variate polynomials over $\mathbb{F}_p$ and let $f \in \mathbb{F}_p[x_1, \ldots, x_n]$. Let $d = \max(\deg(P) + \deg(Q),\ 2\deg(Q), \deg(R), \deg(f))$. If f is independent of (P, Q, R) then any $\mathcal{A}$ that has advantage $1/2$ in solving the decision (P, Q, R, f)-Diffie-Hellman Problem in a generic bilinear group $(\mathbb{G}_1, \mathbb{G}_2)$ must take time at least $\Omega(\sqrt{p/d} - s)$.*

Part II.

CP-ABE Constructions

Chapter 3

Distributed Attribute-Based Encryption

In this chapter we formalize the concept of Distributed Attribute-Based Encryption, give a security model and describe two possible constructions. The first construction will be shown to be very efficient, surpassing all other known CP-ABE constructions in terms of efficiency.

3.1. Motivation

Common to most previous CP-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 [Cha07, SW05]. The limitation to a 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 that 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 3.1) can be expressed as a Boolean formula over attributes.

```
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 ) )
```

Figure 3.1: An example policy

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. Note that we consider the URL part of the attribute name.

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.

Outline The remainder of the chapter is structured as follows: Section 3.2 describes the intuition behind DABE, which is then described more formally in Section 3.3. In the final Sections 3.4 and 3.5 we describe two DABE constructions, including proofs and evaluations. Section 3.6 concludes this chapter.

3.2. DABE Intuition

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 entites in a DABE scheme: a master, attribute authorities and users.

A central *trusted authority* 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 chapter. However, we will discuss this process later in Chapter 6.) 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. 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. 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, i.e., there should be attributes attesting that a user does *not* have a specific property.

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. Figure 3.2 visualizes the interaction between the participants.

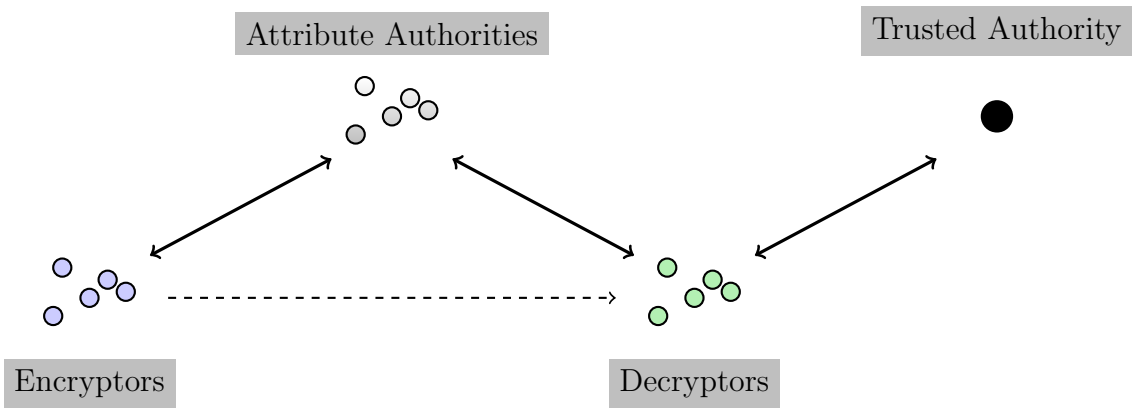


Figure 3.2: Parties involved in DABE

To illustrate the use of a DABE scheme, we return to the above mentioned example of the protection of media files. Figure 3.3 shows the policy of Figure 3.1 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 `is18OrOlder` 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 should not be possible for u to use the corresponding secret attribute key of v. Neither should u be able to give

```
   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.openid.org : is18OrOlder AND
     http://www.contprov2.com : article4325.hasPaidFor)
OR
   ( http://www.openid.org : is18OrOlder AND
     http://www.contprov3.com : articleABC.hasPurchased )
```

Figure 3.3: Policy of Figure 3.1 in DNF

any of his attribute keys to v in a form that allows v to use them together with its own keys. All secret attribute keys are bound to their owner, making them unusable with keys issued for other users.

3.3. Formal Description

3.3.1. DABE Algorithms

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

$(PK, MK) \leftarrow$ **Setup**(1^k) The *Setup* algorithm takes as input the security parameter 1^k. This parameter is often implicitly determined by the application and implementation. *Setup* outputs the public key PK which is used in all subsequent algorithms, and the secret master key MK.

$(PK_u, SK_u) \leftarrow$ **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, that will be used in conjunction with secret attribute keys to decrypt ciphertexts. The user must later be able to prove the connection between his public user key PK_u and

his identity, for example by the use of a PKI. We will elaborate on this in Chapter 6, where we describe an implementation of a DABE scheme for a practical scenario.

$SK_a \leftarrow$ **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.

$\{PK_{\mathcal{A}}, \texttt{NULL}\} \leftarrow$ **RequestAttributePK**$(PK, \mathcal{A}, SK_a)$ The *RequestAttributePK* algorithm is executed by attribute authorities whenever they receive a request for a public attribute key corresponding to an attribute $\mathcal{A}$. 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 $PK_{\mathcal{A}}$, otherwise `NULL`.

$\{SK_{\mathcal{A},u}, \texttt{NULL}\} \leftarrow$ **RequestAttributeSK**$(PK, \mathcal{A}, SK_a, u, PK_u)$ 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 of $\mathcal{A}$. If this is the case, *RequestAttributeSK* outputs a secret attribute key $SK_{\mathcal{A},u}$ for user u. Otherwise, the algorithm outputs `NULL`.

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

$\{M, \texttt{NULL}\} \leftarrow$ **Decrypt**$(PK, CT, \mathbb{A}, SK_u, SK_{\mathcal{A}_1,u}, \ldots, SK_{\mathcal{A}_N,u})$ 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 SK_u, $SK_{\mathcal{A}_1,u}, \ldots, SK_{\mathcal{A}_N,u}$ for user u, which includes the secret user key SK_u and secret attribute keys $SK_{\mathcal{A}_1,u}, \ldots, SK_{\mathcal{A}_N,u}$ 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 $\mathcal{A}_1, \ldots, \mathcal{A}_N$ are sufficient to satisfy $\mathbb{A}$; otherwise it outputs `NULL`.

Note that, as mentioned, the correct binding between u and PK_u must be ensured. A trusted certificate may be used for this purpose. Alternatively, the attribute authority may query the trusted authority for the correct public user key for user u. However, in this solution the central authority may become a performance bottleneck, so a decentralized solution like a PKI is preferable.

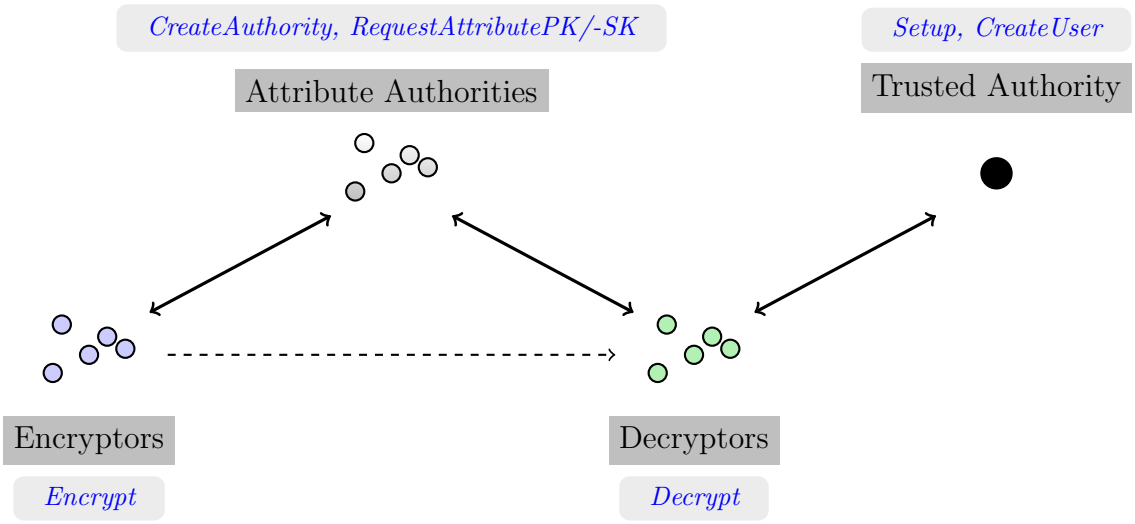


Figure 3.4: Algorithms used in DABE

Figure 3.4 visualizes the algorithms and their relations to the involved parties. Note that this scheme differs from CP-ABE (see Section 2.2) 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 [Cha07], 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*).

3.3.2. Security Model

Similarly to CP-ABE, 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. We follow the CP-ABE security game (see Section 2.4.1), but extend it taking into account the added features of DABE.

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. In this game, eligibility of attributes is not modeled, and the adversary is implicitly considered eligible to all attributes during Phase 1. In practice the adversary is weaker since he may not be eligible to some of the attributes. 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 sufficient to satisfy $\mathbb{A}$, the challenger aborts. As in Phase 1, the adversary is considered eligible for all other secret attribute keys and he 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 $k \in \mathbb{N}$.

3.4. Construction

3.4.1. Description

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 3.3.1 are implemented as follows:

Setup(1^k) The *Setup* algorithm chooses a symmetric pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ of order p (see Section 2.3.1). 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 $PK = \{\mathbb{G}, \mathbb{G}_T, e, g, P, e(g, Q)\}$, while the secret master key is given by $MK = Q$. Note that our construction can easily be modified to work with asymmetric pairings $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, which can be more efficient for high security settings [GPS08]. In this case, g is a generator of $\mathbb{G}_1$ and one chooses the random elements $P, Q \in \mathbb{G}_2$.

CreateUser(PK, MK, u) The algorithm *CreateUser* chooses a secret $mk_u \in \mathbb{Z}_p$ and outputs the public key $PK_u := g^{mk_u}$ and the private key $SK_u := MK \cdot P^{mk_u} = Q \cdot P^{mk_u}$ 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 $SK_a := x_a$.

RequestAttributePK$(PK, \mathcal{A}, SK_a)$ If $\mathcal{A}$ is handled by the attribute authority a, *RequestAttributePK* returns the public attribute key of $\mathcal{A}$, which consists of two parts: $PK_A := \left\langle PK'_{\mathcal{A}} := g^{H_{SK_a}(\mathcal{A})}, PK''_{\mathcal{A}} := e(g, Q)^{H_{SK_a}(\mathcal{A})} \right\rangle$, otherwise the algorithm returns `NULL`. The public key can be requested from the attribute authority by anyone, but *RequestAttributePK* can only be executed by the authority responsible for $\mathcal{A}$, as it requires the secret index of the hash function SK_a as input.

RequestAttributeSK$(PK, \mathcal{A}, SK_a, u, PK_u)$ 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 work.) If this is not the case, *RequestAttributeSK* returns `NULL`, else it outputs the secret attribute key $SK_{\mathcal{A},u} := {PK_u}^{H_{SK_a}(\mathcal{A})} = g^{mk_u H_{SK_a}(\mathcal{A})}$. Note that the recipient u can check the validity of this secret key by testing if $e(PK'_{\mathcal{A}}, SK_u) = PK''_{\mathcal{A}} \cdot e(SK_{\mathcal{A},u}, P)$.

Encrypt$(PK, M, \mathbb{A}, PK_{\mathcal{A}_1}, \ldots, PK_{\mathcal{A}_N})$ A policy in DNF can be written as

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

where N sets $S_1, \ldots, S_k$ denote attributes that occur in the j-th conjunction of $\mathbb{A}$. The encryption algorithm iterates over all $j = 1, \ldots, k$, generates for each conjunction a random value $R_j \in \mathbb{Z}_p$ and constructs a tuple $CT_j = \langle E_j, E'_j, E''_j \rangle$, where

$$E_j := M \cdot (\prod_{\mathcal{A} \in S_j} PK''_{\mathcal{A}})^{R_j}, \quad E'_j := P^{R_j}, \text{ and } E''_j := (\prod_{\mathcal{A} \in S_j} PK'_{\mathcal{A}})^{R_j}. \tag{3.1}$$

The ciphertext CT is obtained as tuple $CT := \langle CT_1, \ldots, CT_k \rangle$.

Decrypt$(PK, CT, \mathbb{A}, SK_u, SK_{\mathcal{A}_1,u}, \ldots, SK_{\mathcal{A}_N,u})$ 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 $SK_{\mathcal{A}_1,u}, \ldots, SK_{\mathcal{A}_k,u}$ contains at least secret keys for all attributes occurring in a set S_j for some $1 \leq j \leq k$. If this is not the case, the algorithm outputs `NULL`, otherwise

$$M = E_j \cdot \frac{e(\prod_{i \in S_j} SK_{i,u}, E'_j)}{e(E''_j, SK_u)}.$$

Correctness. Let $a_j := \sum_{\mathcal{A} \in S_j} H_{SK_{a_{\mathcal{A}}}}(\mathcal{A})$, where we denote by $a_{\mathcal{A}}$ the attribute authority of $\mathcal{A}$. Then

$$E_j = M \cdot e(g, Q)^{a_j R_j}, \quad E''_j = g^{a_j R_j} \tag{3.2}$$

and

$$E_j \cdot \frac{e(\prod_{i \in S_j} SK_{i,u}, E'_j)}{e(E''_j, SK_u)} = M \cdot e(g,Q)^{a_j R_j} \cdot \frac{e(g^{mk_u a_j}, P^{R_j})}{e(g^{a_j R_j}, Q \cdot P^{mk_u})}$$
$$= M \cdot e(g,Q)^{a_j R_j} \cdot \frac{e(g,P)^{R_j mk_u a_j}}{e(g,Q)^{a_j R_j} \cdot e(g,P)^{R_j mk_u a_j}}$$
$$= M.$$

The same is true for the asymmetric case, as $SK_{i,u} = g^{mk_u H_{SK_a}(\mathcal{A})} \in \mathbb{G}_1$, $E'_j = P^{R_j} \in \mathbb{G}_2$, E''_j is product of some $PK'_{\mathcal{A}} = g^{H_{SK_a}(\mathcal{A})} \in \mathbb{G}_1$, and $SK_u = Q \cdot P^{mk_u} \in \mathbb{G}_2$.

3.4.2. CPA-Security

To prove the security of our construction, we basically follow the structure of the security proof of the CP-ABE scheme introduced in [BSW07]. First we show how any adversary who plays the DABE game of Section 3.3.2 (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. Thus, no $\texttt{Adv}_1$ can exist, either. We define the modified game that is used by $\texttt{Adv}_2$ 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. (3.1), but instead of computing E_j as in Eq. (3.2), 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 3.1. *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 $1 \leq j \leq k$) 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

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

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 using the Generic Diffie Hellman Exponent assumption (GDHE) as introduced in Section 2.4.2. 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 3.1 finally implies that there exists no efficient successful attacker $\texttt{Adv}_1$ either, which proves the security of the DABE scheme.

Theorem 3.1. *Let $\texttt{Adv}_2$ be a polynomial time adversary in the generic group model who plays the modified DABE security game and makes q oracle queries. Then $\texttt{Adv}_2$ has advantage at most $O(q^2/p)$ to win the modified game, where p is the order of the bilinear group.*

Proof. Let $\texttt{Adv}_2$ be a polynomial time adversary against the modified security game. $\texttt{Adv}_2$ plays against a simulator, who takes over the role of the challenger

and manages all oracles as outlined in Section 2.4.2. 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 $mk_u \in \mathbb{Z}_p$ and returns encoded versions $\xi(mk_u)$ and $\xi(\tilde{q} + \tilde{p} \cdot mk_u)$ 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 $mk_{\mathcal{A}}$, which simulates the term $H_{SK_a}(\mathcal{A})$ of an attribute $\mathcal{A}$ maintained by attribute authority a; the association between values $mk_{\mathcal{A}}$ 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 $mk_{\mathcal{A}}$ is retrieved from storage. During every request of a public attribute key for $\mathcal{A}$ (a call to *RequestAttributePK*), the simulator returns $\xi(mk_{\mathcal{A}})$ and $\xi_T(\tilde{q}mk_A)$ as encoded versions of the public attribute keys $PK'_{\mathcal{A}}$ and $PK''_{\mathcal{A}}$. If queried for a secret attribute key (through a call to *RequestAttributeSK*), the simulator returns $\xi(mk_u mk_{\mathcal{A}})$ as encoded secret key $SK_{\mathcal{A},u}$.

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 $a_j = \sum_{\mathcal{A} \in S_j} mk_{\mathcal{A}}$. 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 $1, \tilde{p}, \tilde{q}, mk_u, mk_{\mathcal{A}}$ and θ), encodings of combinations of these values given by the simulator (such as the term $mk_u mk_{\mathcal{A}}$ which represents $SK_{\mathcal{A},u}$), 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.

We will now use the General Diffie-Hellman Exponent Problem as introduced in Section 2.4.2. By showing that the target polynomial $f = \tilde{q} a_j R_j$ is independent of the terms the adversary is given according to Definition 2.3, the proof follows from Theorem 2.1.

After Phase 2, the adversary received the following information from the simulator, all in encoded form. Since we are only interested in the polynomials and not their representations, we write $x \in \hat{P}$ instead of $\xi(x)$ and $y \in \hat{Q}$ instead of $\xi_T(y)$:

- The public system key PK: $\tilde{p} \in \hat{P}$ and $\tilde{q} \in \hat{Q}$ (not $\tilde{q} \in \hat{P}$, since the adversary only receives $e(g, Q) = e(g, g)^{\tilde{q}}$ and not $g^{\tilde{q}}$).
- PK_u and SK_u for an arbitrary number of users. Let N be the number of users, denote the jth secret user key $mk_u^{(j)}$: $mk_u^{(j)}$, $\tilde{q} + \tilde{p} \cdot mk_u^{(j)} \in \hat{P}$.
- $PK'_{\mathcal{A}}$ and $PK''_{\mathcal{A}}$ for an arbitrary number of attributes: $mk_{\mathcal{A}} \in \hat{P}$ and $\tilde{q} mk_A \in \hat{Q}$.
- $SK_{\mathcal{A},u}$ for an arbitrary number of attributes and users. Let $L^{(j)}$ be the attribute list for the jth user: $\{mk_u^{(j)} mk_{\mathcal{A}}\}_{\mathcal{A} \in L^{(j)}} \in \hat{P}$, 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., $a_j R_j$, $\tilde{p} R_j \in \hat{P}$ and $\theta_j \in \hat{Q}$, where θ_j may be equal to $a_j R_j$ for all j.

Hence, for N users and a DNF of k conjunctions, the polynomials that he knows are:

$$\hat{P} = \begin{pmatrix} 1, \tilde{p}, \{mk_{\mathcal{A}}\}_{\mathcal{A}}, \\ \{mk_u^{(j)}, \tilde{q} + \tilde{p} \cdot mk_u^{(j)}\}_{1 \leq j \leq N}, \\ \left\{mk_u^{(j)} mk_{\mathcal{A}}\right\}_{\mathcal{A} \in L^{(j)}, 1 \leq j \leq N}, \\ \{\tilde{p} R_i, a_i R_i\}_{1 \leq i \leq k}, \end{pmatrix} \qquad \hat{Q} = \begin{pmatrix} 1, \tilde{q}, \{\tilde{q} mk_{\mathcal{A}}\}_{\mathcal{A}}, \\ \{\theta_i\}_{1 \leq i \leq k} \end{pmatrix}$$

where the subset $L^{(j)}$ does not satisfy $\mathbb{A}$ for any $1 \leq j \leq k$. The polynomial that he needs to construct is $f = \tilde{q} a_j R_j$ for any $1 \leq j \leq k$. We show that he cannot do this because f is independent of $(\hat{P}, \hat{Q})$ according to Definition 2.3.

As a_j and R_j are not contained in $\hat{Q}$, f must contain a product of two polynomials of $\hat{P}$ such 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 polynomials containing $a_j = \sum_{\mathcal{A} \in S_j} mk_{\mathcal{A}}$ in $\hat{P}$, since a_j is contained in f. Aside from E_j and E_j'', a_j can only be constructed by multiplying polynomials of $\hat{P}$ 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 $PK''_A \in \mathbb{G}_T$ (the polynomial is an element of $\hat{Q}$), it cannot be used as input of the pairing. Thus, the only possibility for the attacker is to combine any $PK'_{\mathcal{A}}$ and $SK_{\mathcal{A},u}$. Multiplying representations of $PK'_{\mathcal{A}}$ yields polynomials of the form $\gamma \sum_{\mathcal{A}} mk_{\mathcal{A}}$ for some γ, and multiplying this by $SK_{\mathcal{A},u}$ for some u and $\mathcal{A}$ yields polynomials of the form

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

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 $\sum_{\mathcal{A}} \gamma_{\mathcal{A},u} mk_{\mathcal{A}}$ will evaluate to the required 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 $\mathbb{A}$ 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 $\gamma \sum_{\mathcal{A}} mk_{\mathcal{A}}$ evaluates to a_j for some j is as a product of corresponding public attribute keys, which is obtained by multiplying all representations of $PK'_{\mathcal{A}}, \mathcal{A} \in S_j$, yielding $\xi(a_j)$. It follows, that to construct a polynomial containing a_j, the adversary has no other option than to use either E_j,

Table 3.1: Results of pairings

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

E''_j, or $\prod_{\mathcal{A}\in S_j} PK'_{\mathcal{A}}$. Other polynomials containing $mk_{\mathcal{A}}$ are not useful for him.

Next we consider how to obtain terms containing $\tilde{q} \cdot R_j$. None of the values of $\hat{P}$ or $\hat{Q}$ contains both $\tilde{q}$ and R_j. Thus to get a polynomial containing the product $\tilde{q} \cdot R_j$, two polynomials of $\hat{P}$ need to be multiplied, where each of the values is contained in any term. The only polynomials in $\hat{P}$ that contain $\tilde{q}$ are SK_u. We examine all possible results from multiplying γSK_u with some other value. As shown above, we need not consider polynomials containing $mk_{\mathcal{A}}$, since these are not useful for the adversary.

The first three columns of Table 3.1 list all remaining combinations. It can be seen that the only result that contains all $\tilde{q}$, a_j and R_j is the product of some SK_u and some E''_j which results in a polynomial of the form

$$\tilde{p}R_j mk_u a_j + \tilde{q}a_j R_j.$$

In order to obtain the required term $\tilde{q}a_j R_j$, the adversary has to eliminate $\tilde{p}R_j mk_u a_j$, which can be done by a division with another term of $\hat{Q}$ if $\hat{Q}$ contains a polynomial $\tilde{p}R_j mk_u a_j$. To construct this, he needs to multiply a polynomial from $\hat{P}$ containing $\tilde{p}$ with another polynomial from $\hat{P}$. Thus we need to examine all possible results from pairing SK_u or E'_j (the only elements of $\hat{P}$ depending on $\tilde{p}$) with another value. Once again, Table 3.1 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.)

From this case analysis it follows that no term of the form $\tilde{p}R_j mk_u a_j$ can be constructed by the adversary $\mathtt{Adv}_2$, so f is independent of the terms $(\hat{P}, \hat{Q})$ by

our Definition 2.3. This allows us to apply Theorem 2.1 to conclude that the adversary's advantage is $O(q^2/p)$, proving the theorem. □

3.4.3. Implementation and 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. Furthermore, each conjunction requires only three group elements in the ciphertext, regardless of the number of attributes involved in the conjunction.

To further demonstrate the high efficiency of our DABE construction, we implemented it in C using the PBC library [Lyn] and compared it to the implementation of Bethencourt's original CP-ABE scheme [BSW07] (henceforth simply called CP-ABE) which is also built on PBC. To make the comparison as fair as possible, both implementations were linked against the same PBC instance and both used similar curves, i.e., curves of the PBC-predefined type denoted *Type A* that are based on the equation $y^2 = x^3 + x$.

We created 1 000 000 random alphanumeric strings of length 10-20. These strings were used as attribute descriptors and all tests were run with subsets of this attribute list. As a first test, we created a large set of attribute keys (10 000 and 100 000) using both implementations. For DABE, both *RequestAttributePK* and *RequestAttributeSK* were executed for each attribute, and for CP-ABE, the algorithm *KeyGen* was called to create a key ring containing keys for all given attributes. In the next step, a random group element was encrypted and then decrypted with a large conjunction of attributes (100, 2 500, and 5 000). For CP-ABE, the function `bswabe_enc` was used. Note that when given policies that are not simple conjunctions, both implementations will need to find a conjunction of attributes that satisfies the policy and apply computations to all attributes of the conjunction. Thus, testing only one large conjunction is sufficient for a representative performance comparison. Also, note that in real world settings, a

	KeyGen		*Encrypt*			*Decrypt*		
# of attributes:	*10 000*	*100 000*	*100*	*2 500*	*5 000*	*100*	*2 500*	*5 000*
CP-ABE [BSW07]	101	988	1.084	25.585	56.106	0.401	24.305	80.941
Our construction	52	465	0.006	0.036	0.066	0.004	0.029	0.056

Table 3.2: Performance comparison of DABE and Bethencourt's CP-ABE (all values given in seconds)

hybrid encryption will usually be utilized, where a single random group element will be chosen for ABE encryption and a binary representation of it (for example computed by a hash function $H : \mathbb{G}_T \to \{0,1\}^k$ for an integer k) is used to symmetrically encrypt an actual message. We will use such an approach in our Web Service implementation later in Chapter 6.

All tests were performed on a standard desktop PC equipped with an Intel Core 2 Duo processor at 3 GHz and 4 GB bytes of RAM running Kubuntu Linux. Table 3.2 summarizes the results (all time values are given in seconds).

As it can be seen, DABE outperforms CP-ABE in every aspect. The large difference in the `KeyGen` phase can be explained by the fact that CP-ABE is probabilistic and requires the computation of a pseudorandom value for each attribute. DABE, on the other hand, creates two keys (whereas in CP-ABE the public attribute key is the attribute name itself), but altogether only three exponentiations and two hashes are needed for each attribute. For `Decrypt`, the large differences (factor 10^3) are obviously caused by the different numbers of pairings required for decryption: In CP-ABE two pairings must be computed for each attribute of the conjunction, resulting in a superlinear growth of the time required for decryption, while DABE requires only two pairing operations regardless of the number of attributes involved. Instead, the only action that is dependent on the number of attributes is the multiplication of the private attribute keys associated with each attribute.

3.5. Enhancing Waters' Construction

3.5.1. Modified Construction

It is easy to modify the CP-ABE constructions from [Wat11] 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 [Sti92]. We define a linear secret sharing scheme (LSSS) as follows [Bei96]:

Definition 3.1. *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* $\rho(i)$*.*

We will not discuss share-generating matrices in detail here, but give an intuition. Consider the following example:

$$\rho := \begin{Bmatrix} (1, & A), \\ (2, & B), \\ (3, & C) \end{Bmatrix} \qquad \mathcal{M} := \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & -1 \end{pmatrix}$$

As described above, the column vector $v = (s, r_1)$, so the first share is $\lambda_1 = (\mathcal{M}v)_1 = s + r_1$ and is given to $\rho(1) = A$, the second share is $\lambda_2 = s$ and is given to $\rho(2) = B$, and the third share is $\lambda_3 = -r_1$ which is given to $\rho(3) = C$. Note that $\mathcal{M}$ and ρ are publicy known. With these shares, s can be reconstructed by B alone and by A and C together (by adding their shares), so the matrix represents the formula $B \vee (A \wedge C)$.

It has been shown in [Bei96] 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 $I = \{i \mid \rho(i) \in S\}$ 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.

In general, a LSSS matrix can be efficiently generated from any policy. A simple approach is sketched in Appendix D of [LW11]. More recently, a very efficient algorithm was proposed in [LC10]. Moreover, as that algorithm is deterministic, the LSSS matrix needs not be included along with the ciphertext. If both parties (encryptor and decryptor) use the same algorithm to create the $\mathcal{M}$ and ρ, it suffices to send the policy. In fact, this is the preferred approach as the policy is a lot more readable than a LSSS matrix, and the decryptor should be able to easily understand the rules under which decryption is possible.

After the encryptor has created $\mathcal{M}$ and ρ for his policy, he also needs the public keys $PK_{\mathcal{A}}$ of all attributes used in the access policy as input to the encryption algorithm. Using the notation from Section 3.4, the construction is as follows:

Setup(1^k) 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 $PK = \{\mathbb{G}, \mathbb{G}_T, e, g, P, e(g, Q)\}$, while the secret master key is given by $MK = Q$.

CreateUser(PK, MK, u) The algorithm *CreateUser* chooses a secret $mk_u \in \mathbb{Z}_p$ and outputs the public key $PK_u := g^{mk_u}$ and the private key $SK_u := MK \cdot P^{mk_u} = Q \cdot P^{mk_u}$ 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 $SK_a := x_a$.

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

RequestAttributeSK$(PK, \mathcal{A}, SK_a, u, PK_u)$ 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 $SK_{\mathcal{A},u} := {PK_u}^{H_{SK_a}(\mathcal{A})} = g^{mk_u H_{SK_a}(\mathcal{A})}$.

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

Figure 3.5: Ciphertext in Water's construction

Encrypt$(PK, M, \mathbb{A}, PK_{\mathcal{A}_1}, \dots, PK_{\mathcal{A}_N})$ Given an access policy of the form $\mathbb{A} = \langle \mathcal{M}, \rho \rangle$, *Encrypt* chooses a random vector $\vec{v} = (s, y_2, \dots, y_n) \in \mathbb{Z}_p^{n+1}$. For $i = 1$ to ℓ, it calculates the share $\lambda_i = \mathcal{M}_i \cdot v$. In addition, it chooses random elements $r_1, \dots, r_\ell \in \mathbb{Z}_p$. The complete ciphertext is given in Figure 3.5

Decrypt$(PK, CT, \mathbb{A}, SK_u, SK_{\mathcal{A}_1,u}, \dots, SK_{\mathcal{A}_N,u})$ The algorithm first verifies that the access structure $\mathbb{A}$ can be satisfied by the attributes of $SK_{\mathcal{A}_1,u}$, …, $SK_{\mathcal{A}_N,u}$. If not, it aborts, returning `NULL`, else it proceeds as follows: Let I be the set of all row indices of $\mathcal{M}$ that are associated with the attributes of $SK_{\mathcal{A}_1,u}$, …, $SK_{\mathcal{A}_N,u}$, i.e., $I = \{i \,|\, \rho(i) \in \{\mathcal{A}_1, \dots, \mathcal{A}_N\}\}$. As noted above, there are constants $\{\omega_i \,|\, i \in I\}$ so that $\sum_{i \in I} \omega_i \lambda_i = s$, that can be found in polynomial time. The algorithm computes

$$
e(C', SK_u) / \left(\prod_{i \in I} (e(C_i, PK_u) \cdot e(D_i, SK_{\rho(i),u}))^{\omega_i} \right) = e(g,Q)^s \tag{3.3}
$$

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

The only major difference to the construction from Section 3.4 is in the algorithms *Encrypt* and *Decrypt*.

Correctness. Decryption works if the set of attributes given as input to the *Decrypt* algorithm satisfies the access structure $\mathbb{A}$. In the CPA security proof in Section 3.5.2 we will show that decryption is not possible if the attribute set is not sufficient. If, however, it is sufficient, Equation (3.3) yields the correct result as follows:

$$
\begin{aligned}
& e(C', SK_u) / \left(\prod_{i \in I} (e(C_i, PK_u) \cdot e(D_i, SK_{\rho(i),u}))^{\omega_i} \right) \\
&= e(g^s, QP^{mk_u}) / \left(\prod_{i \in I} (e(P^{\lambda_i} PK_{\rho(i)}^{-r_i}, PK_u) \cdot e(g^{r_i}, PK_u^{H_{SK_a}(\mathcal{A})})^{\omega_i} \right) \\
&= e(g^s, QP^{mk_u}) / \\
&\quad \left(\prod_{i \in I} (e(P^{\lambda_i} g^{-r_i H_{SK_a}(\mathcal{A})}, g^{mk_u}) \cdot e(g^{r_i}, g^{mk_u H_{SK_a}(\mathcal{A})}))^{\omega_i} \right) . \qquad (3.4)
\end{aligned}
$$

For any pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with generators g_1, g_2 and a, b, $c \in \mathbb{Z}_p$ the following equations hold:

$$
\begin{gathered}
e(g_1^a, g_2^b \cdot g_2^c) = e(g_1^a, g_2)^{b+c} = e(g_1^a, g_2^b) \cdot e(g_1^a, g_2^c) \quad \text{and} \\
e(g_1^a, g_2^b) = e(g_1^b, g_2^a) .
\end{gathered}
$$

For symmetric groups ($\mathbb{G} = \mathbb{G}_1 = \mathbb{G}_2$), the last equation implies that the pairing operation is commutative, i.e., $\forall A, B \in \mathbb{G}$, $e(A, B) = e(B, A)$. Using these rules, Equation (3.4) can be split up and rearranged as follows:

$$
\begin{aligned}
(3.4) &= e(g, Q)^s \cdot e(g, P)^{smk_u} / \\
&\quad \left(\prod_{i \in I} (e(g, P)^{\lambda_i mk_u} \cdot e(g, g)^{-r_i H_{SK_a}(\mathcal{A}) mk_u} \cdot e(g, g)^{r_i H_{SK_a}(\mathcal{A}) mk_u})^{\omega_i} \right) \\
&= e(g, Q)^s \cdot e(g, P)^{smk_u} / \left(\prod_{i \in I} (e(g, P)^{\lambda_i mk_u})^{\omega_i} \right) \\
&= e(g, Q)^s \cdot e(g, P)^{smk_u} / e(g, P)^{\sum_{i \in I} \omega_i \lambda_i mk_u} \\
&= e(g, Q)^s \cdot e(g, P)^{smk_u} / e(g, P)^{smk_u} \\
&= e(g, Q)^s .
\end{aligned}
$$

As seen in the *Decrypt* algorithm, if the ciphertext component C is divided by this value, the result is the plaintext message M, so the decryption succeeds. □

3.5.2. CPA-Security

We prove the security of this construction in two ways. First we show how to apply Waters' original proof to the DABE version. This proof allows a security reduction to a mathematical assumption. However, the reduction requires a very restrictive security model. We call this *non-adaptive security*. The construction can also be proven secure in the stricter *adaptive security* model that was used in the previous DABE construction by using the General Diffie Hellman Exponent Assumption (Section 2.4.2).

Non-adaptive Security. To prove the security of his construction, Waters [Wat11] 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 that is considered. In particular we allow only *non-adaptive key queries*, similar to the Multi-Authority Threshold ABE scheme by Chase [Cha07]: 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 instead create a new user by submitting another sequence of *CreateUser*, *RequestAttributeSK* calls. Note that due to the CP-ABE property of collusion-resistance he cannot mix the new keys with the old ones, so he has to create the new key ring containing all required attributes for the user. 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 *Init* phase (see Section 2.4.1). The 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, each sequence of attribute claims for a user is considered a single call, so the dispatch of the secret user keys and secret attribute keys for each user is stalled until the adversary 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 request is made for a set of user keys that would 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 [Wat11]. Thus, the proof of [Wat11], Section 4.1, can directly be applied to the above construction.

Adaptive Security. However, using generic groups we can also prove security in the strict security game (see Section 3.3.2) similar to our original DABE construction. Similarly to the proof in Section 3.4.2, we define a modified game that is used by an adversary $\mathtt{Adv}_2$:

The phases **Setup**, **Phase 1**, and **Phase 2** are equal to the original 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. (3.1), but instead of computing the ciphertext component C as in the construction, he computes C as

$$C = \begin{cases} e(g,Q)^s, & \text{if } b = 1 \\ e(g,g)^\theta, & \text{if } b = 0\,. \end{cases}$$

Lemma 3.2. *If there exists an adversary* $\mathtt{Adv}_1$ *who has advantage of* ϵ *to win the original game, then there exists an adversary* $\mathtt{Adv}_2$ *which wins the modified game with advantage* $\epsilon/2$*.*

The proof is similar to the one of Lemma 3.1.

Theorem 3.2. *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. The challenger plays the modified game by using the oracles to compute representations of all terms similarly to the challenger that is described in the proof of the original DABE construction. To emulate the random group elements P and Q, we write $P = g^{\tilde{p}}$ and $Q = g^{\tilde{q}}$ for $\tilde{p}$, $\tilde{q} \xleftarrow{R} \mathbb{Z}_p^*$, so $e(g,Q)^s = e(g,g)^{\tilde{q}s}$.

As before, we use the GDHE assumption to prove this theorem. In order to prove the security, we need to show that the adversary's target polynomial $f = \tilde{q}s$ is independent of $(\hat{P}, \hat{Q})$, where $\hat{P}$ and $\hat{Q}$ are the polynomials he receives from the challenger. We begin by examining $\hat{P}$ and $\hat{Q}$.

After the challenge phase, the adversary has the following polynomials:

- The public system key PK: $\tilde{p} \in \hat{P}$ and $\tilde{q} \in \hat{Q}$ (not $\tilde{q} \in \hat{P}$, since the adversary only receives $e(g,Q) = e(g,g)^{\tilde{q}}$ and not $g^{\tilde{q}}$).
- Public attribute keys for all attributes: $\{H_{SK_a}(\mathcal{A})\}_{\mathcal{A}} \in \hat{P}$. Since we model H as a random oracle, all hash values are independent from each other and from any other values in the system.
- For an arbitrary number of users u the public and secret user keys and for each u secret attribute keys for a number of attributes: Let N be the number of users, denote the jth user $u^{(j)}$, his secret user key $mk_u^{(j)}$ and his set of attributes $L^{(j)}$. Then the polynomials are $mk_u^{(j)}$, $\tilde{q} + \tilde{p}mk_u^{(j)} \in \hat{P}$ and $\{mk_u H_{SK_a}(\mathcal{A})\}_{\mathcal{A} \in L^{(j)}} \in \hat{P}$.

  However, for no j does $L^{(j)}$ satisfy the challenge access structure $\mathbb{A}$.
- The ciphertext components:
  - The component C': $s \in \hat{P}$.
  - The component C: $\theta \in \hat{Q}$, where θ is either $\tilde{q}s$ or a random value.
  - For $1 \leq i \leq \ell$ (with ℓ the number of rows of the LSSS matrix $\mathcal{M}$), the components C_i and D_i: $\tilde{p}\lambda_i - H_{SK_a}(\rho(i))r_i$, $r_i \in \hat{P}$ for random r_i and the shares $\lambda(i)$.

Thus the vectors are:

$$\hat{P} = \begin{pmatrix} 1, \tilde{p}, s, \\ \left\{ mk_u^{(j)}, \tilde{q} + \tilde{p}mk_u^{(j)} \right\}_{1 \leq j \leq N}, \\ \{H_{SK_a}(\mathcal{A})\}_{\mathcal{A}}, \\ \left\{ mk_u^{(j)} H_{SK_a}(\mathcal{A}) \right\}_{\mathcal{A} \in L^{(j)}, 1 \leq j \leq N}, \\ \{\tilde{p}\lambda_i - H_{SK_a}(\rho(i))r_i,\, r_i\}_{1 \leq i \leq \ell} \end{pmatrix},$$

$\hat{Q} = (1, \tilde{q}, \theta)$ and $f = \tilde{q}s$.

First we examine how the adversary can build polynomials containing $\tilde{q}$. This value occurs in $\hat{Q}$ as $\tilde{q} \in \hat{Q}$ and as part of the polynomials $\tilde{q} + \tilde{p}mk_u^{(j)} \in \hat{P}$ for any user $u^{(j)}$. A term in $\hat{Q}$ can not be multiplied with any other term, so $f = \tilde{q}s \in \hat{Q}$

can not be computed from $\tilde{q} \in \hat{Q}$, and the only way to have $\tilde{q}s$ as an addend is to multiply $\tilde{q} + \tilde{p}mk_u^{(j)}$ with a value that contains s.

Now we examine how to construct such a polynomial containing s, but with the restriction that s does not have an additional factor x. If s was part of sx for some x, then the multiplication with $\tilde{q} + \tilde{p}mk_u^{(j)}$ would yield a polynomial of $\hat{Q}$ containing $\tilde{q}sx$, which still contains x as a term. However, x cannot be removed from this polynomial, as after one multiplication the result is in $\hat{Q}$ and no further multiplication is possible. The vector $\hat{P}$ contains $s \in \hat{P}$, which could be used, but there is potentially another way to construct a polynomial containing s: Since $\sum_{i \in I} \omega_i \lambda_i = s$, a polynomial containing s can be computed using the terms $\tilde{p}\lambda_i - H_{SK_a}(\rho(i))r_i$. However, the resulting polynomial would have $\tilde{p}$ as factor of s which cannot be removed.

Thus, $f = \tilde{q}s$ can only be computed by multiplying $\tilde{q} + \tilde{p}mk_u^{(j)}$ (for any j) with $s \in \hat{P}$, resulting in $s\tilde{q} + s\tilde{p}mk_u^{(j)}$, so the adversary needs to build $s\tilde{p}mk_u^{(j)}$ in order to remove it and arrive at the desired term $f = \tilde{q}s$.

Again, s occurs only in two polynomials: $s \in \hat{P}$ and in the form $\tilde{p}\lambda_i - H_{SK_a}(\rho(i))r_i$. To compute $s\tilde{p}mk_u^{(j)}$ from $s \in \hat{P}$, the adversary needs a term of $\hat{P}$ containing both $\tilde{p}$ and $mk_u^{(j)}$. However, these two values are only both contained in $\tilde{q} + \tilde{p}mk_u^{(j)}$, but when using this to remove $s\tilde{p}mk_u^{(j)}$, the adversary would also cancel out the target polynomial $\tilde{q}s$.

So the adversary needs to multiply a suitable linear combination of $\tilde{p}\lambda_i - H_{SK_a}(\rho(i))r_i$ for some i with a polynomial containing $mk_u^{(j)}$. All possible multiplication results are:

$$(\tilde{p}\lambda_i - H_{SK_a}(\rho(i))r_i) \cdot mk_u^{(j)} = \tilde{p}\lambda_i mk_u^{(j)} - H_{SK_a}(\rho(i))r_i mk_u^{(j)} \tag{3.5}$$

$$\begin{aligned}(\tilde{p}\lambda_i - H_{SK_a}(\rho(i))r_i) \cdot (\tilde{q} + \tilde{p}mk_u^{(j)}) = \tilde{p}^2 mk_u^{(j)}\lambda_i + \tilde{p}\tilde{q}\lambda_i - \tilde{q}H_{SK_a}(\rho(i))r_i \\ - \tilde{p}mk_u^{(j)}H_{SK_a}(\rho(i))r_i\end{aligned} \tag{3.6}$$

$$(\tilde{p}\lambda_i - H_{SK_a}(\rho(i))r_i) \cdot mk_u^{(j)}H_{SK_a}(\mathcal{A}) = \tilde{p}\lambda_i mk_u^{(j)}H_{SK_a}(\mathcal{A}) - H_{SK_a}^2(\rho(i))r_i mk_u^{(j)} \tag{3.7}$$

Only equation (3.5) on the first line contains a term that can be used to construct $\tilde{p}smk_u^{(j)}$. Determining a set I and coefficients ω_i such that $\sum_{i \in I} \omega_i \lambda_i = s$, the adversary can create a linear combination of the product (3.5) that contains the

desired polynomial:

$$\begin{aligned}
&\sum_{i\in I}\left(\omega_i\cdot(\tilde{p}\lambda_i - H_{SK_a}(\rho(i))r_i)\cdot mk_u^{(j)}\right)\\
&=\sum_{i\in I}\tilde{p}\omega_i\lambda_i mk_u^{(j)} - (\omega_i\cdot H_{SK_a}(\rho(i))r_i mk_u^{(j)})\\
&=\tilde{p}s mk_u^{(j)} - \sum_{i\in I}(\omega_i\cdot H_{SK_a}(\rho(i))r_i mk_u^{(j)})\,.
\end{aligned}$$

(Note that scalar multiplication is still possible in $\hat{Q}$ because addition is possible in $\hat{Q}$.)

In order to eliminate the right-hand term, the adversary finally needs to build polynomials of the form $H_{SK_a}(\rho(i))r_i mk_u^{(j)}$ for the same j that was used for the rest of the computation and for all $i \in I$. The only other term that contains r_i is $r_i \in \hat{P}$, so the adversary needs to multiply r_i with a polynomial that contains both $H_{SK_a}(\mathcal{A})$ and $mk_u^{(j)}$. The only term that qualifies is $H_{SK_a}(\mathcal{A})mk_u^{(j)} \in \hat{P}$, which represents a private attribute key. Now the important observation is that no user satisfies the LSSS matrix. This means that for each possible I, with $\sum_{i\in I}\omega_i\lambda_i = s$, there is at least one non-null coefficient ω_i where $\rho(i)$ points to an attribute that is not element of $L^{(j)}$, i.e., there is no corresponding secret attribute key for the jth user. Thus, the adversary will arrive at a polynomial of the form

$$\tilde{q}s - \sum_i H_{SK_a}(\rho(i))r_i mk_u^{(j)}\,,$$

where the right-most sum contains at least one element that cannot be removed by the adversary. It follows that $f = \tilde{q}s$ is independent of $(\hat{P}, \hat{Q})$. □

3.5.3. 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 [LC10, LW11].

3.6. Conclusion

CP-ABE is a promising concept for next-generation access control and arguably the most interesting technology to implement data-centric security at the moment. However, to be usable in a pervasive and highly distributed environment, its extension to settings with multiple authorities is necessary. In this chapter, 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 and the names of the attributes. A central trusted authority is only needed for the creation of users. Later in Chapters 5 and 6 we will discuss the practical implications of such an architecture and demonstrate how this can be used in real-world settings.

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. Our performance analysis of an implementation of that construction showed that all algorithms are much more efficient than conventional CP-ABE. This can be explained by the fact that the construction uses only a constant number of pairing operations, while in conventional CP-ABE, that number is linearly dependent on the size of the policy. Furthermore, we showed how a recent CP-ABE construction can be extended to fit the DABE scheme. We proved the security of that second construction both in the Generic Group Model and – by using a relaxed, weaker security model – in the form of a reduction proof.

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.

Chapter 4

Hiding the Policy

4.1. Introduction

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

4.1.1. Towards Policy Privacy

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

The second approach, which we are concerned with here, is to modify common

CP-ABE constructions to somehow hide the policy while still allow an eligible user to decrypt. We first examine that a policy can never been *completely* hidden in a ciphertext, as it has to be stored in a finite space and a known format, so there is always a limited, finite set of possible policies that can be encoded in a particular ciphertext. This motivates to introduce the notion of *policy anonymity*, which is similar to the established notion of anonymity sets [SD02] and k-anonymity [CdVFS07]: Given a number of candidates for a policy, the anonymity set, an attacker cannot determine which actual policy was used for the encryption.

Extending a CP-ABE construction to support a policy hiding feature has been attempted by Nishide et al. [NYO09] and Yu et al. [YRL08], both of which extend the CP-ABE scheme of [CN07], where the policy consists of a single AND-gate. Simply speaking, in these extensions the policy is still an AND-gate, but the decryptor does not now the particular configuration and has to apply all his attribute keys to decrypt. In both cases the anonymity set consists of all policies that consists of a single AND-gate over a subset of all attributes of the system.

In this chapter we show that one of Nishide's CP-ABE constructions [NYO09] can be modified in order to support the encryption with every Boolean formula by combining several AND-gates in a specific way and using a novel idea from graph theory. This in turn allows the encryptor to choose a particular anonymity set which contains – among with the original policy – many more.

The idea of the construction is as follows: Given a policy, represented by a Boolean syntax tree with $\wedge$ and $\vee$-gates, we construct a *major* of this tree, i.e., a supertree that is built by expanding nodes of the original tree into new subtrees. Such a major can be used to express many different policies by assigning different expressions to its leaves. The set of all such policies makes up the anonymity set. The decryptor knows only that the used policy is among all policies that can be encoded by the supertree. The leaves of this major are encryptions of blinded partial secrets that represent $\wedge$-gates. As these $\wedge$-gates are hidden, an adversary does not know which of the possible policies of the anonymity set is used in the encryption, but by our construction he is still able to decrypt the message if he fulfills the hidden policy. He will determine which of the leaves he is able to satisfy, obtain some of the encoded partial secrets, combine them according to the tree structure using his private key, and unblind the resulting combination to retrieve the secret. Our application of Nishide's construction takes collusion attacks into

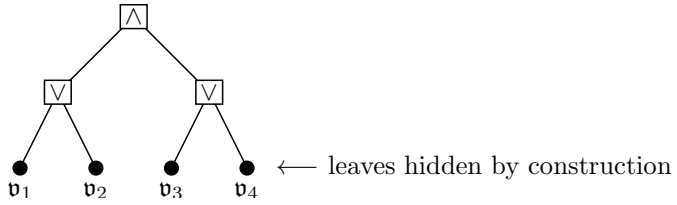


Figure 4.1: Sample obfuscated policy

P_1 : `RoleA` $\wedge$ `(RoleB` $\vee$ `RoleC)`

P_2 : `(adult` $\vee$ `cc = verified)` $\wedge$
`((contprov1.article1 = purchased` $\wedge$ `account1 = balanced)` $\vee$
`(contprov2.article1 = purchased` $\wedge$ `account2 = balanced))`

P_3 : `userrole = surgeon` $\wedge$ `employer = hospitalx`

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

Figure 4.2: Example policies for Figure 4.1

account, so no group of users (who fulfill different parts of a policy) can decrypt the policy unless one member of the group fulfills the complete policy.

Example To give an intuition of the hiding property of our system, examine the tree in Figure 4.1 which may be sent along with the ciphertext. It represents the structure of a policy anonymity set, and its topology is known to everyone, but the leaves are hidden using the ideas of Nishide's construction. Each leaf hides an $\wedge$-gate with an unknown configuration. Each $\wedge$-gate could also represent the constant values $\bot$ (false) or $\top$ (true). Figure 4.2 shows some policies that might be encoded with this tree. Consider, for example, policy P_4. In our construction, each of $x_1, \ldots, x_{10}$ may represent an expression of the form $A = x$ for an attribute A and an attribute value x. Here, the leaf $\mathfrak{v}_1$ could encode the expression $\mathfrak{v}_1 \equiv x_1 \wedge x_2 \wedge x_3$, $\mathfrak{v}_2$ could encode $\mathfrak{v}_2 \equiv x_4$, $\mathfrak{v}_3 \equiv x_5 \wedge x_6$, and $\mathfrak{v}_4 \equiv x_7 \wedge x_8 \wedge x_9 \wedge x_{10}$. There are various ways to encode simpler policies like $x_1 \wedge x_2$ or $x_1 \wedge (x_2 \vee x_3)$. For example, the former policy can be encoded by mapping, $\mathfrak{v}_1 \equiv \bot$, $\mathfrak{v}_2 \equiv x_1 \wedge x_2$ $\mathfrak{v}_3 \equiv \top$, and $\mathfrak{v}_4$ to a random $\wedge$-gate, or by mapping $\mathfrak{v}_1 \equiv x_1$, $\mathfrak{v}_2 \equiv \bot$, $\mathfrak{v}_3 \equiv x_2$, and $\mathfrak{v}_4 \equiv \bot$. Several other mappings are possible. This small example already shows that the policies encoded in a simple tree can be very complex and diverse.

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

4.1.2. Related Work

In predicate encryption schemes [LOS+10, KSW08, SBC+07, BW07], decryption is possible if a predicate over the user attributes and the ciphertext attributes is fulfilled. Current PE constructions are very powerful and support rather expressive predicates. Currently the most versatile solutions seem to be those that use inner product queries [LOS+10, KSW08]. It has been shown [KSW08] that such a scheme can be used to construct a scheme that supports, for example, DNFs or CNFs of some bounded degree, or a predicate that can be expressed by a polynomial over the attributes. However, this predicate (for example a predicate for DNFs of some degree d) is encoded in the user keys, so it is fixed after the key generation algorithm. The complexity of the system is dependent on the size of that predicate. This means that no single PE scheme is able to express every possible policy in polynomial size and due to the bounded size of the predicate can only support a limited set of policies. Speaking in terms of anonymity, there is a fixed anonymity set that applies to all ciphertexts of an instantiation of a PE system.

In our approach, there is no fixed anonymity set. Instead, each encrypting party decides on the anonymity set while encrypting. All policies are expressed as syntax trees, so every Boolean formula can be expressed in polynomial size. As we will show in Section 4.4.1, the anonymity set is exponential in the size of the tree major that was used to encode the policy.

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

Aside from PE schemes, policy privacy has also been examined in the context of trust negotiation [FLA06]. Here, large scrambled circuits are used to obfuscate the underlying policy, which is similar to our idea of using large tree majors. Trust negotiation is an interactive process whereas in this thesis we are concerned with an off-line access control mechanism. Recently, Seyalioglu and Sahai [SS10] proposed an encryption scheme which also uses garbled circuits and hides the

policy. However, in their scheme, the public key of a recipient must be used for the encryption, making it infeasible in the CP-ABE setting where the identities of the recipients are not known.

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

4.2. Syntax Tree Majors

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

Definition 4.1 (monotonic syntax tree)**.** *A monotonic syntax tree T is a tree where all inner nodes are labeled with either $\wedge$ or $\vee$ and the leaves represent either Boolean variables or the constant values $\bot$ or $\top$. If the root of T is labeled $\wedge$, then every inner node of odd depth is labeled $\vee$, and every inner node of even depth is labeled $\wedge$. We call such a tree* $\wedge$-rooted*. Analogously, a* $\vee$-rooted *tree is a tree whose root is labeled $\vee$ and where every inner node of odd depth is labeled $\wedge$, and every inner node of even depth is labeled $\vee$.*

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

As explained in the introduction, we will use a CP-ABE scheme that encrypts the leaves, which correspond to attributes, but the construction will hide the concrete correspondence between leaves and attributes. Also note that our construction supports only monotonic syntax trees, but as there might be negative attributes (i.e., attributes that attest that the possessor does *not* have a certain property),

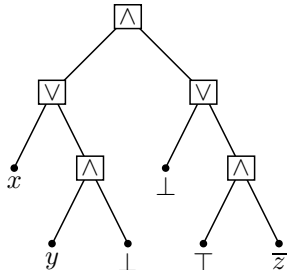


Figure 4.3: Monotonic syntax tree

even non-monotonic policies can be represented by monotonic syntax trees by applying DeMorgan's laws until all negations are atomic.

In order to further obfuscate the policy, we compute a larger policy such that by mapping some of its leaves to the values $\top$ and $\bot$ we are able to encode the original policy. For example, the monotonic syntax tree in Figure 4.3 represents the formula $x \wedge \overline{z}$. As an adversary does not know which leaves (if any) are mapped to $\top$ and $\bot$, there are many possible forms the encoded policy might have. As the configuration of the leaves is hidden, he is not able to access the concrete policy. We say that the larger policy *semantically contains* many smaller policies. More formally:

Definition 4.2 (semantic containment)**.** *Let F and G be Boolean formulas over vectors of Boolean variables $\vec{x} = (x_1, \ldots, x_n)$, resp. $\vec{y} = (y_1, \ldots, y_m)$ where $m \leq n$. We call F semantically contained in G if there exists a function ϕ that maps the variables of $\vec{x}$ to either variables of $\vec{y}$ or to constant values $\top$ or $\bot$, such that $G(\psi(\phi(\vec{x}))) = F(\psi(\vec{x}))$ for all configuration mappings $\psi : \vec{x} \mapsto \{\bot, \top\}^n$.*

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

The type of supertree we examine is closely related to the notion of *tree majors*. Informally, a tree R is a major of a tree Q, if Q can be obtained from R by contracting a number of edges. Equally, a major of Q can be constructed by expanding some nodes into subtrees. A major can be characterized by a mapping

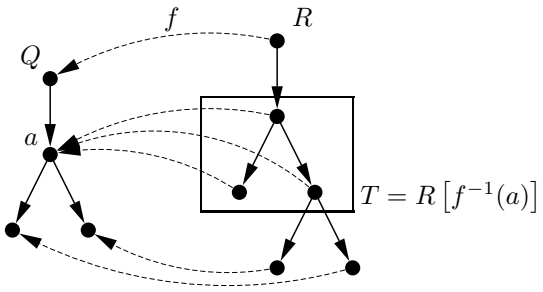


Figure 4.4: A mapping f of a major to a minor (leaves omitted)

$f : V(R) \to V(Q)$ of vertexes of a tree R to Q. For details we refer the reader to the extensive literature about smallest common supertrees and related topics, for example, [RV06, Val05, RB09]. Also, for a good, though somewhat dated, survey see [Bil05]. However, the specific constraints we are dealing with in our scenario have to our knowledge not yet been discussed.

We call R a *syntax tree major* of Q if we can find a mapping f with the following properties: Given a node $a \in Q$, the nodes of $f^{-1}(a)$ form a connected subtree T of R, which we denote $T = R\,[f^{-1}(a)]$. This is illustrated in Figure 4.4. Different subtrees must not overlap and all edges of Q must be preserved in R. This is similar to the definition of a tree major.

However, in our scenario we additionally require the expanded tree to preserve the labeling of all nodes, as it needs to have the same semantics as the original tree. To understand the implications of this, let the label of a in Figure 4.4 be $\vee$. All other labels of Q follow from this by Definition 4.1. Now consider the subtree T of R. As our definition does not allow adjacent $\vee$-nodes, some nodes of T must be labeled $\wedge$. However, as both the direct predecessor of T in R and all direct successors of T in R are labeled with $\wedge$, no node of T can have the label $\wedge$. From this consideration, it follows that all subtrees introduced in a tree major of a syntax tree must have even height.

Both R_1 and R_2 of Figure 4.5 are examples of such majors. Note the placements of the leaves a and b. In both cases, the root of the smallest subgraph that contains both nodes has a root labeled with $\vee$. This node will take on the role that the parent of $f(a)$ and $f(b)$ in Q has (which is also an $\vee$-node).

Generally, all syntax tree majors must follow the rule that if two nodes a and b have a common parent in Q, then their unique common ancestor in R must have the same label as that parent. As a counter example consider the tree major R_3 in

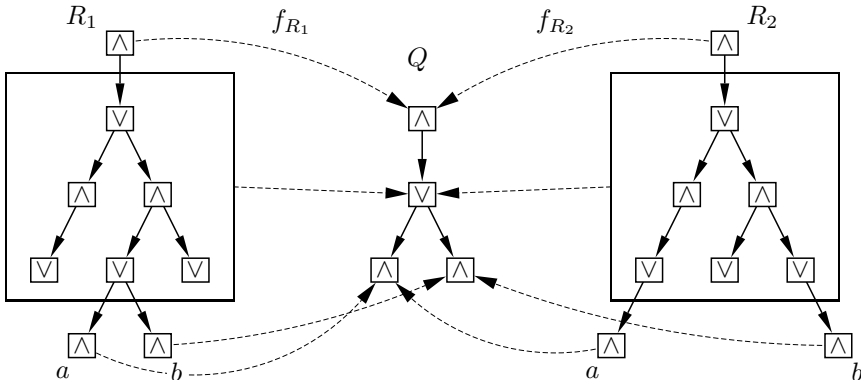


Figure 4.5: Two valid syntax tree majors

Figure 4.6, where the root of the smallest subtree containing a and b is an $\wedge$-node, thus not qualifying as a syntax tree major. The original graph contained a formula $f(a) \vee f(b)$, but this cannot be encoded in the given major, as a and b are only connected by an $\wedge$-node. It is now easy to see that in these cases the smallest subtrees containing a and b must have odd height.

More formally, we adapt the definition of tree minors from [NRT99] to implicitly labeled monotonic syntax trees and define *syntax tree majors* as follows:

Definition 4.3 (syntax tree major)**.** *A tree R is a syntax tree major of a tree Q if there exists a surjection $f : V(R) \to V(Q)$ such that*

1. *for each $a \in V(Q), T = R[f^{-1}(a)]$ is a connected subtree of R, and every path from the root of T to a leaf of T consists of an even number of edges;*

2. *for each pair $a, b \in V(Q)$, $f^{-1}(a) \cap f^{-1}(b) = \emptyset$;*

3. *for $S = \{(u, v) \in E(R) \mid f(u) \neq f(v)\}$, there exists a bijection $\xi : S \to E(Q)$ such that for each $e(s, t) \in S$, $\xi(e) = (f(s), f(t))$.*

4. *For each pair of edges $(x, a) \in E(Q)$ and $(x, b) \in E(Q)$, let U be the smallest subtree of R that has both a and b as leaves. Then the paths from the root of U to the roots of the subtrees $f^{-1}(a)$ and $f^{-1}(b)$ have odd length.*

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

We will now construct a mapping $\tilde{M} : V(R) \to \{\mathcal{F}, \top, \bot, \text{rand}\}$ that allows us to embed Q in a syntax tree major R such that R computes the same function as

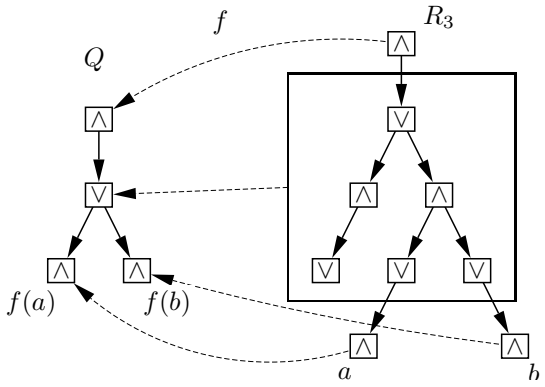


Figure 4.6: An invalid syntax tree major

Algorithm 1: `MarkRelevantNodes`(R, Q, f)

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

forall $T \in V(R)$ **do** $M(T) \longleftarrow \emptyset$;
$M(\text{root}(R)) \longleftarrow \mathcal{F}$;
foreach *leaf* $\mathfrak{u}$ *of* Q **do**
  $T \xleftarrow{\text{R}}$ leaf of $f^{-1}(\mathfrak{u})$;
  while $M(T) \neq \mathcal{F}$ **do**
    $M(T) \longleftarrow \mathcal{F}$;
    $T \longleftarrow \text{parent}(T)$;
  end
end

Q. This is achieved by mapping some of the leaves of R to the constant values $\bot$ or $\top$. The effect of this is that some inner nodes of R that are connected to these leaves will map to trivial formulas like $x \wedge \top$ or $x \wedge \bot$, which are equal to x or $\bot$. For an example, we refer to Figure 4.3. The remaining leaves (that are not set to $\top$ or $\bot$) are then mapped to leaves of Q such that R evaluates the same function as Q.

Finding such a mapping can be done in a two step process. In the first step, we will mark all inner nodes of R that are needed to encode nodes of Q with a special symbol that we call $\mathcal{F}$. For convenience, all other nodes will be marked with $\emptyset$. Call this mapping $M : V(R) \to \{\emptyset, \mathcal{F}\}$. As R is a syntax tree major of Q, this marking can be found by the following iterative algorithm: For each leaf of Q, a corresponding leaf in R is selected and the path from the root of R to this node is marked by traversing upwards (see Algorithm 1).

In the second step, the other inner nodes (not marked with $\mathcal{F}$ by Algorithm 1) will be marked with $\top$ or $\bot$, such that they do not affect the computation of the encoded policy. Furthermore, we will mark all leaves that have no impact on the computation with the value rand. This value shall indicate the leaf can be chosen randomly from $\{\bot, \top\}$ or any randomly chosen policy that can be encoded as a single $\wedge$-gate in the system. This will be useful in the cryptographic construction presented in Section 4.3. Algorithm 2 (on page 76) marks all nodes that were marked with $\emptyset$ to $\bot$, $\top$, or rand. Call this mapping $\tilde{M} : V(R) \to \{\mathcal{F}, \top, \bot, \text{rand}\}$. The algorithm marks the nodes as follows: If a node is an $\wedge$-node marked $\mathcal{F}$, all children marked $\emptyset$ will recursively be set to $\top$. Similarly, if a node is an $\vee$-node marked $\mathcal{F}$, all children marked $\emptyset$ will recursively be set to $\bot$. For nodes that are not marked $\mathcal{F}$, an appropriate mapping of the children to $\bot$, $\top$, or rand is chosen, such that the nodes do not influence the computation.

Theorem 4.1. *Let R be a syntax tree major of a monotonic syntax tree Q and let f be the characteristic function of this major, as described in Definition 4.3. Applying the function $\phi : \mathbb{L}(R) \to \mathbb{L}(Q) \cup \{\bot, \top\}$, defined as*

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

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

For every node $q \in Q$, let $T(q) := R[f^{-1}(q)]$ be the subtree of R that q is expanded into. Consider Definition 4.3. As f is a surjection, all nodes of Q are reached by applying f to the nodes of R. By (1), for each node $q \in Q$, $f^{-1}(q)$ is a connected subtree and by (2), all expanded subtrees $T(q)$ of nodes q are disjoint.

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

We need the following lemmas:

Lemma 4.1. *For the root r_R of R and the root r_Q of Q, it holds that $f(r_R) = r_Q$.*

Proof. Suppose not. Then $f(r_R)$ has a predecessor x and is connected to this predecessor by an edge (x, r_R). By (3) of Definition 4.3, this edge is preserved in R, so r_R also has a predecessor. This contradicts the assumption that r_R is a root. □

Lemma 4.2. *Let R be a syntax tree major of Q, and let Q and R be equally rooted (i.e., both are $\wedge$-rooted or both are $\vee$-rooted). Then for each node $q \in Q$, the root of the subtree $T(q)$ has the same (implicit) label as q.*

Proof. We prove this lemma by induction over the structure of the tree. Lemma 4.1 implies that the root r_Q of Q has the same label as the root of $T(r_Q)$ (because the root of $T(r_Q)$ is the root of R and both trees are equally rooted). Now let $q \in Q$ be a node for which Lemma 4.2 holds, i.e., the root of $T(q)$ has the same label as q. By (1) of Definition 4.3, $T(q)$ has even height, so the leaves of $T(q)$ also have the same label as q. Suppose that the label of q is $\wedge$ and let r be a child of q in Q. Then the label of r must be $\vee$ (by Definition 4.1). By the induction hypothesis, the root of $T(q)$ is labeled $\wedge$. Then the leaves of $T(q)$ are also labeled $\wedge$. Again, by (3), the edge $(r, q) \in E(Q)$ is preserved in R, so there is an edge from a leaf of $T(q)$ to the root of $T(r)$. This root must be labeled $\vee$. If the label of q is $\vee$, a similar argument holds. □

We are now in a position to prove Theorem 4.1 by structural induction.

Proof. Let $\mathfrak{u}$ be a leaf of Q. Algorithm 1 randomly marked a leaf $\mathfrak{v} \in T(\mathfrak{u})$ of R with $\mathcal{F}$, as well as all nodes of the single path from $\mathfrak{v}$ to the root of $T(\mathfrak{u})$. Algorithm 2 configured all nodes on this path such that they return the value of their single marked child (this means that they compute the identify function). By this, the value of the marked leaf is propagated along the path to the root. It follows that for all configurations $\psi : \mathbb{L}(Q) \to \{\bot, \top\}$, $\psi(T(\mathfrak{u})) = \psi(\phi(\mathfrak{v}))) = \psi(f(\mathfrak{v}))) = \psi(\mathfrak{u})$.

Now suppose that for all children $w_i, 1 \leq i \leq n_q$ of a node q, $w_i \equiv T(w_i)$ (where n_q is the number of children of q). We need to show that this implies that $q \equiv T(q)$. By (3) of Definition 4.3, the unexpanded edges of Q are preserved in R. This means that for each edge $(q, w_i) \in E(Q)$ there is an edge in R that that connects a leaf of $T(q)$ to the root of $T(w_i)$. From Lemma 4.2 we know that the root of $T(q)$ and all leaves of $T(q)$ have the same label as q. From (4) of Definition 4.3 we conclude that marked paths meet only at nodes that have the same label as the

Algorithm 2: `SetNodeValue`(T, m)

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

```
if T is not a leaf then
    M̃(T) ⟵ m ;
    if T is an ∧-node then t ⟵⊥ else t ⟵⊤;
    switch m do
        case F
            foreach child c, M(c) = F do  SetNodeValue(c, F);
            foreach child c, M(c) ≠ F do  SetNodeValue(c, not t);
        end
        case t
            Select a random child c;
            SetNodeValue(c, t);
            forall other children c do  SetNodeValue(c, rand);
        end
        otherwise
            forall children c do  SetNodeValue(c, m);
        end
    end
end
```

root of $T(q)$. All other nodes on the marked paths are set by Algorithm 2 such that they compute the identity function, and all other nodes that are unmarked are configured to have no influence on the computation.

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

A direct consequence of this Theorem is:

Corollary 4.1. *Every syntax tree major R of a monotonic syntax tree Q semantically contains Q.*

4.3. Building the System

We now describe a CP-ABE system with hidden policies, where policies are represented as syntax trees. It is based on [NYO09], but extended to support any Boolean formula by utilizing syntax tree majors. The leaves of the syntax tree are expressions of the form $A = x$, where A identifies an attribute and x the value that this attribute must have. See Figure 4.2 for some examples. Our construction supports n attributes, denoted $L_1, \ldots, L_n$. Each attribute can take on one of a number of symbolic values. Note that this is different from the other constructions in this thesis, were attributes are binary – a user either has a specific attribute or not. Binary attributes can easily be emulated by allowing each attribute to have one of two possible values, one representing that a user has the corresponding property and one representing that he does not have it. We denote the number of possible values of an attribute L_i by n_i and the symbolic values of the attribute by $v_{i,1}, \ldots, v_{i,n_i}$. Thus, each leaf of the tree encodes an expression $L_i = v_{i,t}$. Using this approach, we are able to support every policy that can be expressed as a Boolean formula.

Note that we can also emulate numeric attributes using a bag of bits representation [BSW07], where each number is represented by a bit string and there are two attributes for each bit. To use this, the policy would first be formulated in a more abstract form, using comparisons with numbers in the leaves like $A = x$, $A \leq x$, or $A \geq x$. These leaves would then be expanded into subtrees that evaluate the expressions using the bit representations, as outlined in [BSW07].

4.3.1. Setup and Key Generation

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

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

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

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

4.3.2. Encryption

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

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

Another, more interesting approach is to "mix" Q with other trees, constructing a common major that from an adversary's point of view could encode all of the input trees as well as numerous combinations of them. Ideally, we would like combinations of trees to be as small as possible. This problem is known as the smallest common supertree problem and is well-studied for tree majors [RV06, Val05, RB09]. Generally, this problem is NP-hard, but by adding some reasonable constraints on the input trees, it becomes tractable. We can adopt the algorithm for finding the smallest common major of two trees described by

Nishimura et al. [NRT99] to our definition of syntax tree majors. The only constraint of Nishimura's algorithm is that the input trees must have a bounded degree. This is reasonable for our settings as a sufficient upper bound for all realistic syntax trees of a particular scenario can easily be estimated.

As our notion of syntax tree majors is based on the definition of tree majors used in [NRT99], we can modify Nishimura's algorithm to work for our definition of syntax tree majors. We briefly describe the necessary modifications: The most important restriction in our definition, compared to the original one, is that nodes of tree R can only be combined with nodes of Q that have equal labels. As the algorithm of [NRT99] always combines the roots of Q and R, both trees must be either $\wedge$-rooted or $\vee$-rooted. If they are not equally rooted, we construct a new node N and attach either Q or R as a subtree. The main loop of the algorithm tries all possible combinations of nodes of Q with nodes of R. For syntax tree majors, we must restrict these combinations to combinations that map nodes of equal labels, i.e., the distances of the combined nodes to their respective root must both be even or odd. Finally, the remainder of the algorithm must be restricted such that only mappings that preserve the labels and furthermore adhere to (4) are tried.

With this modification the algorithm can be utilized by the encryptor to systematically construct a tree R that is a syntax tree major of a set of trees: In addition to the monotonic syntax tree Q, select some monotonic syntax trees $P_1, \ldots, P_n$. Find the smallest common tree major of all Q and P_i as follows: Set $Q_0 := Q$ and for all $i \in \{1, \ldots, n\}$ let Q_i be the smallest common major of Q_{i-1} and P_i. The resulting tree Q_n is a syntax tree major of $P_1, \ldots, P_n$ and Q and can be used to encode any of the formulas encoded by Q, any P_i, and numerous combinations. Figure 4.7 shows an example.

A third approach to construct suitable syntax tree majors could be to initially decide on a large generic tree R_0 that semantically contains all possible policies that are used in a given setting. For example, an encryptor may find that all policies that he normally uses are syntax tree minors of a 3-ary tree of height 4. Then he could always set R to that tree and use it as a syntax tree major for all encryptions. Using such an approach for a policy represented by a syntax tree Q, a mapping $f : V(R) \to V(Q)$ must be found that adheres to Definition 4.3. If Q is indeed a minor of R, such a mapping can be found in $O(|V(R)|)$ by a brute

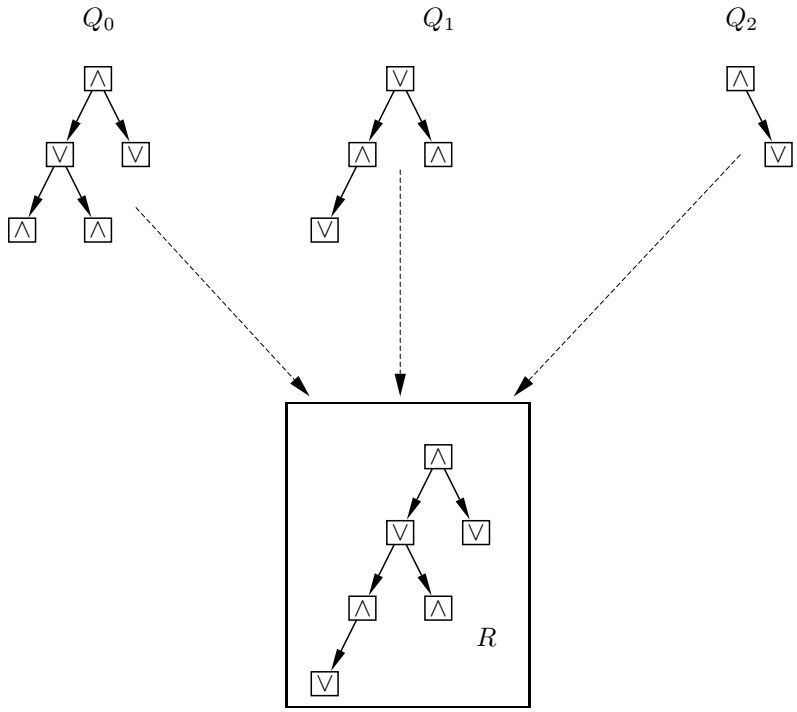


Figure 4.7: Constructing R as a comon major

force algorithm that tries to match vertices of Q to vertices of R starting from the leaves of both trees. In this case, after initially selecting the generic tree R_0, the process of constructing a tree major is omitted for all further encryptions, and instead the encryptor simply sets $R := R_0$.

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

Encoding the formula. After constructing a syntax tree major R with root T and mapping f, the encryptor randomly chooses an $r \in \mathbb{Z}_p^*$ and executes `EncodeSecret`(T, r) (see Algorithm 3). This algorithm encodes the secret value $r \in \mathbb{Z}_p^*$ into the tree in the following way: Beginning from the root of the tree, the algorithm recursively traverses downwards to the leaves. If a node is an $\wedge$-node, the secret r is split into partial secrets, one for each child of the node (the decryptor must satisfy all children to recover the secret), so that the sum of all partial secrets equals r. If a node is an $\vee$-node, the secret is propagated to all child nodes. The

Algorithm 3: `EncodeSecret`(T, r)

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

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

output is a labeling $m(c)$ of all nodes of $c \in R$ to partial secrets in $\mathbb{Z}_p^*$. The idea is that a decryptor needs to be able decrypt a sufficient set of partial secrets to recover the main secret r. The encryptor then applies the Algorithms 1 and 2 of Section 4.2 to find the mapping $\tilde{M}$. After this process, each leaf $\mathfrak{v}$ is marked with $\tilde{M}(\mathfrak{v})$, a partial secret $m(\mathfrak{v})$, and there is a mapping $f(\mathfrak{v})$ that maps $\mathfrak{v}$ to a leaf of Q that represents an expression of the form $L_i = v_{i,t}$. (Note that the algorithms mark all nodes of R, but from now on we will only need the marks of the leaves.)

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

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

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

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

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

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

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

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

The final ciphertext is

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

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

4.3.3. Decryption

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

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

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

satisfy the leaf, $\tau^{(\mathfrak{v})}$ might also be equal to 1. However, the probability for this occurring is $1/p$ for p the order of the bilinear group, which is negligible.

Note that while the decryptor now knows which parts of the tree he satisfies, he does not know the policies of the respective leaves since their configuration is hidden by the construction. However, with all $\tau^{(\mathfrak{v})}$, he is able to decrypt as follows: First he removes some of the leaves that he does not satisfy (i.e., where $\tau^{(\mathfrak{v})} \neq 1$) as they do not contain any information that he can use. For all $\tau^{(\mathfrak{v})} \neq 1$, replace $\mathfrak{v}$ with the constant value $\perp$ and simplify the tree by substituting subtrees with their obvious results using the formulas $A \wedge \perp = \perp$ and $A \vee \perp = A$. The remaining tree either contains only leaves that can be satisfied or is a single node $\perp$. In the latter case, return $\perp$ (as the attribute set does not satisfy the policy). For each remaining $\vee$-node N, randomly choose a subtree of N and substitute N with it. (This works because Algorithm 3 encoded the same value in all subtrees of an $\vee$ node, so we can use any of them to retrieve it.)

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

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

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

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

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

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

so the exponents do not cancel out and the result will be different from M (with overwhelming probability).

4.4. Discussion

4.4.1. Anonymity of the Policy

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

In some scenarios, it might suffice if the attacker knows only the general form of the policy, i.e., he wants to know which nodes of the tree belong to the actual policy and which ones are dummy gates introduced in order to obfuscate the policy. In our construction, the form of the policy is determined by the leaves. Some of these are set to a constant value ($\top$ or $\bot$) to render unused inner nodes inoperative, some are genuine $\wedge$-gates encoding parts of the policy. Thus, to find out which form the original policy has, an attacker must know which $\wedge$-gates are constant values and which ones are not, which for a tree R with leaves $\mathbb{L}(R)$ gives $O(2^{|\mathbb{L}(R)|})$ combinations. However, for reasons of symmetry, some of these forms may be topologically identical, so the number of forms might be smaller than that. More concrete, the most symmetries are found in a complete n-ary tree. However, in [Mat70] it is shown that even in such a tree, the number of subtrees is exponential in the number of nodes, so it is at least exponential in the number of leaves. Thus, even taking into account symmetries, the number of possible forms of a policy

encoded in a syntax tree major is exponential in the number of leaves.

4.4.2. Security and Policy Anonymity

CPA-Security. As the system uses the same structures as Nishide's construction, its CPA-security can directly be derived from it. We can model the security using a standard security game similar to the ones we used in Sections 2.4.1 and 3.3.2, modified to take into account the properties of our construction. As we only claim security in the generic group model, the more powerful non-selective version of the models can be used:

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

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

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

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

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

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

Theorem 4.2. *The construction given in Section 4.3 is secure in the generic group model.*

Proof. Similar to our proof of DABE in Section 3.4.2, we show how any adversary who plays the CP-ABE game (denoted $\texttt{Adv}_1$) can be used to construct an adversary in a slightly modified game ($\texttt{Adv}_2$). Then we prove that no such $\texttt{Adv}_2$ can exists,

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

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

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

Lemma 4.3. *If there exists an adversary $\texttt{Adv}_1$ who has advantage of ϵ to win the original CP-ABE game, then there exists an adversary $\texttt{Adv}_2$ who wins the modfied game with advantage $\epsilon/2$.*

Proof. Given an adversary $\texttt{Adv}_1$ that has advantage ϵ in the CP-ABE game, we can construct an adversary $\texttt{Adv}_2$ as follows: $\texttt{Adv}_2$ simulates $\texttt{Adv}_1$. In the phases **Setup**, **Phase 1**, and **Phase 2**, $\texttt{Adv}_2$ forwards all messages he receives from $\texttt{Adv}_1$ to the challenger, and all messages from the challenger to $\texttt{Adv}_1$. In the **Challenge** phase, $\texttt{Adv}_2$ receives to messages M_0 and M_1 from $\texttt{Adv}_1$ and the challenge CT (which contains $\tilde{C}$ that is either $e(g_1, g_2)^{\omega r}$ or $e(g_1, g_2)^\theta$ for a random θ) from the challenger. He flips a coin b and multiplies $\tilde{C}$ by M_b and sends the resulting ciphertext CT' to $\texttt{Adv}_1$. When $\texttt{Adv}_1$ outputs a guess b', $\texttt{Adv}_2$ outputs 1 if $b' = b$ and 0 if $b' \neq b$. If for the original $\tilde{C} = e(g_1, g_2)^\omega$, then $\texttt{Adv}_2$'s challenge given to $\texttt{Adv}_1$ is a well-formed CP-ABE ciphertext and $\texttt{Adv}_1$ has advantage ϵ of guessing the correct $b' = b$. Otherwise, the challenge is independent of the message M_0 and M_1, so the advantage of $\texttt{Adv}_2$ is 0. Thus, we have

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

□

To prove Theorem 4.2, we use the asymmetric case of the Generic Diffie-Hellman Exponent problem as introduced in Section 2.4.2. We need to show that the adversary's target term $f = \omega r$ is independent of (P, Q, R) according to Definition 2.5, where (P, Q, R) are the polynomials from $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ that he knows. If this is the case, the security follows from Theorem 2.2. After the challenge phase, the adversary has the following terms:

- The public system key PK: $\left\{a_{i,t} \in \mathbb{Z}_p^*\right\}_{1 \leq t \leq n_i, 1 \leq i \leq n}, \beta, \overline{\beta} \in P.\ \omega, \overline{\omega} \in R$,
- Private keys SK_L for several queries L. Let the number of these sets be N and denode the jth query $L^{(j)}$. For each query $1 \leq j \leq N$, the challenger created a secret value $s^{(j)}$ and random values $\lambda_i^{(j)}, 1 \leq i \leq n$ (n being the number of attributes): $\beta^{-1}(\omega - s^{(j)}), \overline{\beta}^{-1}(\overline{\omega} - s^{(j)}), s^{(j)} + a_{i,t_i}\lambda_i^{(j)}, \lambda_i^{(j)} \in Q$. No $L^{(j)}$ satisfies the policy per the definition of the security game.
- The ciphertext components
  - C_0: $\beta r \in P$ for a randomly chosen r.
  - For all leaves that contain genuine $\wedge$-gates and are satisfied for an $L^{(j)}$ $C_{i,1}^{(\mathfrak{v})}, 1 \leq i \leq n$ and $C_{i,t,2}^{(\mathfrak{v})}, 1 \leq t \leq n_i, 1 \leq i \leq n$: $r_i^{(\mathfrak{v})}$, $a_{i,t} r_i^{(\mathfrak{v})} \in P$.
  - For all leaves the unblinding values of the secondary cryptosystem $H^{(\mathfrak{v})}$ and $\overline{C_0}^{(\mathfrak{v})}$: $\overline{\omega} m^{(\mathfrak{v})} \in R$, $\overline{\beta} m^{(\mathfrak{v})} \in P$.

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

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

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

and $f = \omega r$.

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

Table 4.1: Pairing a term containing r with a term containing $s^{(j)}$

The adversary needs to build the target polynomial ωr from P, Q, R using only the combinations implied by Definition 2.5. The product ωr is not contained in any term, so he needs to multiply a polynomial of Q with a polynomial of P or Q such that both factors ω and r are contained in either term. First observe which terms contain ω. There is only $\omega \in R$ which cannot be multiplied with any other polynomial as it is in R and $\beta^{-1}(\omega - s^{(j)}) \in Q$ for any $1 \leq j \leq N$. As this has β^{-1} as additional factor, the adversary must eliminate β^{-1} from any of these polynomials by multiplying it with a polynomial from P or Q that contains β. There are two possibilities to achieve this: Multiplying with $\beta \in P$ yields $(\omega - s^{(j)}) \in R$, and multiplying with $\beta r \in P$ yields $(\omega r - r s^{(j)}) \in R$. These are the only two methods to create a polynomial that contains ω, but not β^{-1}. We now examine which of these two combinations can be used to build the target polynomial ωr.

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

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

To construct a polynomial that contains $r s^{(j)}$, the adversary now needs to multiply a polynomial that contains $s^{(j)}$ with a polynomial that contains r or at least $r_i^{(\mathfrak{v})}$ (as r can be constructed using a sum over some $r_i^{(\mathfrak{v})}$). Table 4.1 lists all results.

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

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

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

However, note that none of the adversary's keyrings satisfies the access policy. This means that for each j there is at least one $\mathfrak{v}$, for which he does not have a sufficient set $s^{(j)} + a_{i,t_i} \lambda_i^{(j)}$ to build the partial secret $m^{(\mathfrak{v})}$. If he uses such a wrong attribute set, he will not be able to cancel out the term $\sum_{\mathfrak{v}} \sum_i a_{i,t_i} \lambda_i^{(j)} r_i^{(\mathfrak{v})}$, as the corresponding ciphertext components are not $a_{i,t}^{(\mathfrak{v})} r_i^{(j)} \in P$ but were set by the encryptor to random values. Without a sufficient set of partial secrets, he cannot build r and thus not $rs^{(j)} + \sum_{\mathfrak{v}} \sum_i a_{i,t_i} \lambda_i^{(j)} r_i^{(\mathfrak{v})}$ for any j.

As this is the only remaining possibility to remove the unneeded term $rs^{(j)}$ from the polynomial, it can now be seen that $f = \omega r$ is independent of (P, Q, R), and from Theorem 2.2 it follows that the adversary cannot build f and thus cannot break the system. □

Anonymity. We also need to show that an attacker cannot distinguish encryptions performed by two policies that are in the same anonymity set. As shown in Section 4.4.1, from an adversary's view the anonymity set is defined by the tree major and the set of nodes that he can decrypt. The latter set is in turn defined by his attribute set.

To formally define anonymity, we utilize a game as follows: The adversary decides on two policies W_0 and W_1 as well as a common syntax tree major R of both W_0 and W_1. Note that there is only a fixed number of ways that each policy can be embedded in R, as the process is defined by our algorithms given in Section 4.2. Given an attribute list and a syntax tree major R, an adversary can determine if exactly one of W_0 and W_1 is an element of the anonymity set. This can only be the case if one of the leaves of R can be decrypted for one of the two policies W_0 and W_1 and cannot be decrypted for the other one. If this is the case for any of the leaves, the attacker can determine which of the policies was used, and the anonymity is broken. This also means that the anonymity set for this list

of attributes does not contain both W_0 and W_1, but only one of them. However, as we only claim policy anonymity for the case that either both or neither of W_0 and W_1 are in the anonymity set, we restrict the adversary to only query attribute lists where the adversary is not able to make such a distinction.

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

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

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

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

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

Theorem 4.3. *If Nishide's construction is secure, the adversary cannot win the game.*

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

4.4.3. Reducing the Size of the Ciphertext

For each leaf's encryption every attribute of the system is used. This is the only way to maximize the anonymity set, because when some attribute A is not used

for the decryption of a leaf $\mathfrak{v}$, then the decryptor can obviously conclude that the partial policy of $\mathfrak{v}$ does not contain A. However, if the universe of attributes of a given system is very large, it might be feasible to use only a subset of all attributes for the encryption of each leaf while still using enough attributes to get a sufficiently large anonymity set. Similarly to [NYO09], each leaf $\mathfrak{v}$ may be encrypted with its own set of attributes $\mathcal{A}_\mathfrak{v}$. $\mathcal{A}_\mathfrak{v}$ can be a random superset of the set attributes actually used in the leaf. However, in order to hide as much of the semantics of each partial policy, some care must be taken, as it should be understood which information an attacker gains by the knowledge of $\mathcal{A}_\mathfrak{v}$. It must also be considered that $\mathcal{A}_\mathfrak{v}$ must be sent along with each leaf, which slightly increases the size of the ciphertext.

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

4.5. Conclusion

We introduced the notion of policy anonymity in cryptographic access control. To this end, we proposed the idea to obfuscate the policy used in an encryption by constructing a syntax tree major of the syntax tree that encodes the desired policy. The leaves are then hidden from an adversary using a cryptographic primitive. We discussed how these majors can be characterized and how the leaves need to be configured to encode a specific, given policy in one of its majors. The majors can be chosen arbitrarily large, and the larger a major is the larger becomes the anonymity set. From the anonymity set, an adversary gains only very general informations about the encoded policy; for example he knows an upper bound

of its complexity and that some of his attribute sets satisfy certain parts of the major. We then used these primitives to modify a CP-ABE scheme with partially hidden policies to support every policy that can be expressed as a Boolean formula and enable an encryptor to obfuscate that policy.

Our construction compares favorably to [NYO09], as it is able to efficiently encode any policy that can be expressed as a Boolean formula and is not limited to policies with small DNF represenations, and to the various Predicate Encryption schemes. However, it may be possible to construct a scheme that hides even more properties of the encoded policy by using a different encoding of the it, like garbled circuits which presently have been utilized to solve different problems [FLA06, SS10]. We leave this as future work. Also, our approach may be applicable to other CP-ABE system that like Nishide's support only $\wedge$-conjunctions.

Part III.

Applications

Chapter 5

Cryptographic Enforcement of DRM Licenses

5.1. Introduction

Digital Rights Management (DRM), as its variants *Enterprise Rights Management (ERM)* and *Information Rights Management (IRM)* [Tho09], provide mechanisms for fine-grained access and usage control for documents. To this end, documents are encrypted, and decryption is controlled by a DRM viewer that runs on the clients' system. This viewer enforces the access rules associated with each document. The basic idea of enforcing policies on the client side using encrypted documents is very similar to Attribute-Based Encryption, so we will now take a close look at Digital Rights Management (DRM), as it seems to be an especially interesting application scenario for cryptographic access control.

In DRM, usage rules of a document are represented in the form of a *license*, a digital object that contains both the usage policy and a decryption key. A user needs to undertake two steps in order to use content protected with DRM: First he must obtain a license, then he downloads the content itself which is encrypted by the *content provider* and can be decrypted with the decryption key specified in the usage license. These two data objects may be acquired independently from two different parties. There are a number of different DRM standards that describe different architectures.

For simplicity, in this chapter we examine only the Open Mobile Alliance DRM

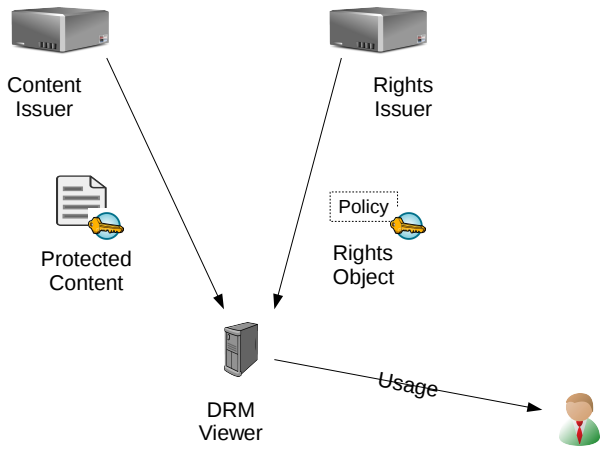


Figure 5.1: OMA architecture (simplified)

architecture (OMA DRM) [Mot07, All08]. In OMA DRM, a protected content object is called a *media object*. It is offered for download by a *content issuer*. The usage license is contained in a *rights object (RO)* which is purchased from a *rights issuer*. This license contains *usage rules* and a key that is personalized for the device that is (by the rules of the license) allowed to render the content. The license may also differ depending on the payment model (i.e., what usage rights the user has payed for specifically) and other aspects that are up to the rights issuer. Figure 5.1 visualizes this concept.

An advantage of such an architecture is that the content can be stored on untrusted devices and distributed without restrictions because it is encrypted and cannot be used without obtaining a rights object. As the content is often very large (e.g., video and audio data), this is an especially useful property because cheap, insecure data storage can be used for the content and a trusted party is only needed to distribute the small rights objects.

From a more abstract point of view, we can consider that the owner of a content has specified a policy made up of rules for every possible usage scenario for the content (*publishing license*). The issuing process of the license done by the rights

issuer can be seen as an enforcement of some parts of that policy (for example parts describing who is eligible to purchase what kind of licenses), while other parts that are only applicable to specific usages and specify restrictions that apply to a specific user extracted into the license. The task of the player (DRM agent) is to enforce such remaining parts. Our approach is to embed the publishing license directly into the ciphertext using Attribute-Based Encryption. This way, the first enforcement process is done implicitly by cryptographic means, and the personalization of the license is no longer necessary. Another advantage of this approach is that the rights issuer does not have to be queried (or contacted at all) on each transaction in the process of obtaining a usage license. This reduces the number of network transmissions and lowers the demands on the rights issuer's servers (especially in regard to reachability).

In this approach, a rights object that is purchased from a right issuer contains a policy over attributes that was created using public attribute keys. Each user has secret attribute keys representing information about the devices he owns and other properties that may be interesting for a DRM provider. Using these attribute keys, the user can decrypt the media object if his attributes satisfy the policy.

However, in practice not all parts of such a DRM policy can be represented by static attributes. For example, a rights issuer may allow customers to purchase rights to play a particular video on a certain device at most 3 times. While it is easy to grant the user an attribute key that allows him to access the video on a device, it is not obvious how he can be forced to give up this ability if there is no way for the rights issuer to revoke the respective attribute. Generally, revocation of attribute keys is an open problem of all ABE constructions and – more generally – of all variants of cryptographic access control. We will discuss this issue later in Section 6.2.2.

Using ABE for such a scenario leads to a new architecture for DRM systems that enforces parts of the policies cryptographically, while at the same time reduces the trust required in DRM viewers. To do this, we partition the set of rules into *static* and *dynamic* rules. Static rules are enforced by cryptographic means before an access to the media takes place, while dynamic rules are enforced at runtime by a trusted DRM viewer. To give an intuition of this idea, consider a typical license of a DRM protected system where a media can only be accessed by a certain type of device. This is an example of a static rule: It must checked before any kind of

access to the media, no matter what specific usage (e.g., playing or copying) is desired. To statically enforce this rule, we can encrypt the media with a key that is only given to valid devices. In fact, simple static rules like this are already in use: In most DRM systems, each media is encrypted with a *title key* K_T, which is sent encrypted along the license and can only be decrypted by using a private *player key* K_P that is known only to authenticated DRM viewers (and the rights issuer). This construction can be seen as a static rule stipulating that access must only be granted to an authenticated DRM viewer. If one manages to restrict usage of K_P to authorized viewers, trust relies upon the issuer of the key K_P, who – unlike the viewing platform – can be controlled by the media distributor. This observation is crucial to our approach.

Opposed to traditional DRM architectures, DRM content in our framework is not directly encrypted by a single title key, but with an arbitrary Boolean formula over a set of keys. In the most general case, these keys may even be maintained by different, independent authorities. A DRM viewer has to retrieve the relevant keys from the respective authorities in order to decrypt the media. Each of these keys is associated with a certain property of the user or the platform that is executing the viewer and is only granted to the viewer if certain conditions and/or obligations (that are part of the license) are fulfilled.

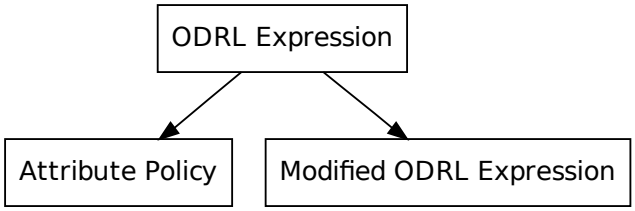


Figure 5.2: Extraction of static rules

More formally, let m be a media and K_P the player specific key that is needed to access the media. We denote the title key that is needed to access m by K_T. In traditional DRM schemes, the encrypted media is distributed along with $E(K_P, K_T)$. Generalizing this approach, we can use cryptography to enforce properties statically. For example, let K_D be another key that is only granted by a

key-issuer after a certain date has passed. Using this approach, we can additionally encrypt the media with K_D to enforce that the media can only be accessed by an authorized viewer *and* after the given date:

$$E(K_D, E(K_P, K_T)) .$$

We can further generalize this idea so that any Boolean formula can be used to represent conditions that have to be met before decryption. To achieve this, we use the notion of *Attribute-Based Encryption (ABE)* as introduced in Section 2.2 and discussed in the preceding chapters of this thesis. In our example, where both K_D and K_P are required to decrypt K_T, the encryption policy could be written as $p(\mathbb{K}) := K_D \wedge K_P$, where $K_D, K_P \in \mathbb{K}$. The title key would then be encrypted as

$$E(p(\mathbb{K}), K_T) .$$

In order to decrypt, a user has to acquire a set of attributes such that he is able to satisfy $p(\mathbb{K})$. We call the set of obligations that are enforced by such a predicate *static rules*.

Static rules add an additional layer to the dynamic DRM enforcement framework supplementing the enforcement of the dynamic rules with cryptographic primitives. Figure 5.3 visualizes this relationship. Even if an attacker is able to overcome the enforcement of the dynamic rules, there are static rules that must be fulfilled before he is able to access the media. However, the player lacks cryptographic keys for this step. This is favourable as it allows a reduction of trust in the viewers: While classic DRM schemes are completely broken if an attacker manages to extract a secret player key, our scheme at least ensures that – even in case of a compromised viewer – some parts of the license, i.e., the static rules, are always enforced, since crucial keys are not available to the attacker. Furthermore, using cryptography to enforce static rules can replace the license personalization process, as outlined before, as the publisher license itself can be used directly instead.

It is, however, not very practical to require two policies (one containing the static rules and one containing the dynamic rules), and in fact, the distinction between static and dynamic rules might not be clear to license authors. Thus, we propose a *license conversion tool* as a central component of our framework. This conversion tool takes as input a DRM license and gives as output the static enforceable sub-

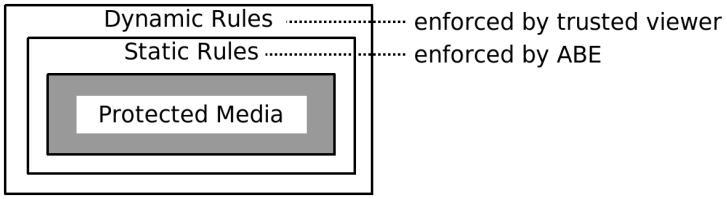


Figure 5.3: Structure of DRM rules

policy, containing only static rules which are suitable for use with a cryptographic algorithm (called attribute policy), and a modified DRM license that contains the remainder of the rules. The attribute policy is enforced through a cryptographic construction, while the remainder of the policy (i.e., the dynamic rules) needs to be enforced at runtime by the DRM viewer. This strategy effectively allows to reduce the trust in the viewer, as some important rules like group membership, payed fees, or a temporal usage range can be enforced cryptographically even on compromised DRM viewers.

Outline. The rest of this chapter is structured as follows: We describe a DRM framework that utilizes DABE in Section 5.2. To illustrate the practicability of our framework, we describe a conversion process for DRM licenses in a scenario where ODRL policies are used in Section 5.3. We describe an implementation of this conversion process in Section 5.4. Finally, Section 5.5 concludes the chapter.

5.2. Framework

We propose the DRM framework as depicted in Figure 5.4. As usual, the media distributor formulates a license that he attaches to the media. This license includes both dynamic rules and static rules. In fact, from the point-of-view of the distributor there may not be any obvious differences between the two types of rules. Subsequently, he runs a conversion tool that takes as input the license and a set of *conversion rules*. The tool analyses the policy to find the components that contain static rules, and extracts an *attribute policy* $\mathbb{A}$ (i.e., a sub-policy containing only the static rules) from the license. This attribute policy $\mathbb{A}$ contains all cryptographically enforceable parts of the license and is used in the DABE encryption step. The media distributor furthermore obtains *public attribute keys*

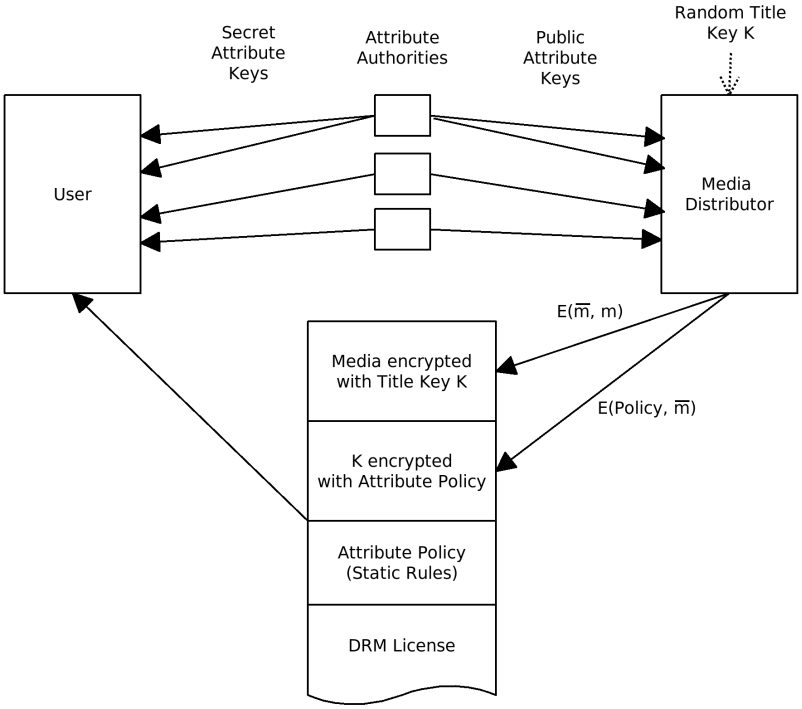


Figure 5.4: Involved parties, relationships, and data structures

for all attributes referenced in $\mathbb{A}$ from the attribute authorities. In a multi-authority scenario he might have a choice between different, independent authorities, and the media distributor can modify the conversion rules used for extracting static rules from ODRL policies to constitute his decision of the authorities he trusts most. This allows for much greater flexibility. For example, the distributor may entrust different parties with the control of different aspects of a policy, or he may even give the decryptor alternatives that allow him to request some attributes from one a set of different authorities in case one of them is unreachable or the decryptor does not trust it. However, it must be clear from the names of the attributes in the attribute policy which attribute authority was chosen, so that the users can request secret attribute keys from the same authorities.

The media is then encrypted using a title key, which is in turn encrypted with the *Encrypt* algorithm of the DABE scheme. Thus, only users with secret attribute key rings that satisfy the attribute policy $\mathbb{A}$ can decrypt the title key and the media. The media is distributed along with the encrypted title key, the attribute policy $\mathbb{A}$ and the modified ODRL expression.

Whenever a user downloads protected content, he inspects the attribute policy

and obtains a set of secret attribute keys that fulfill the policy, unless he is already in possession of the required keys. He is able to access the title key by executing the *Decrypt* algorithm, which in turn can be used to access the media.

Not shown in Figure 5.4 is the central authority that initializes the global system parameters (algorithm *Setup*) and grants every user u a key pair (PK_u, SK_u) that is used by the attribute authorities to personalize the secret attribute keys. The global system parameters and public user keys created by this trusted authority are made available securely to all participants.

The task of the attribute authorities is the provision of attribute keys. At creation, each authority executes *CreateAuthority*, taking as input the global system parameters. Whenever queried for public attribute keys – usually by media distributors – the authority executes *RequestAttributePK*. All attribute keys are identified by strings, and each attribute authority is able to maintain an arbitrary number of attributes. Users may query the authorities for secret attribute keys. When this happens, the queried attribute authority has to verify if the user is eligible of the attribute, and if so, execute *RequestAttributeSK*, giving as input the public user key. Depending on the implementation and the semantics of the attributes, a user may have to perform some actions (e.g. make a payment) before he becomes eligible for some of the keys. Note that this incremental claim of attributes is not supported by most conventional CP-ABE constructions, as there is usually a single *KeyGen* operation that gives a user all of his attribute keys at once. DABE, however, is specifically designed to support settings where users may request new attribute keys at any time.

Also note that the attribute policy limits general access to the media by enforcing certain prerequisites for decryption, but it cannot control how the media is used once the attribute policy is fulfilled. Thus, only those rules are enforceable that specify conditions that must be fulfilled by the viewers *before* access is granted for the first time. Static rules are thus a subset of all rules of a policy. Dynamic rules must still be enforced at runtime by the DRM viewer.

5.3. Processing ODRL Expressions

To illustrate the practicability of our framework, we consider a DRM scenario in which the Open Digital Rights Language (ODRL) [Ini02] is used to express licenses.

An ODRL expression allows to describe a set of actions a user is allowed to perform and a set of rules associated with each action. The rules describe properties that the user or the executing device has to fulfil before the associated actions can be taken. They are classified into constraints, requirements, and conditions. Constraints are limits to exercising the actions, requirements are obligations that must be fulfilled in order to be allowed to perform the action, and conditions specify exceptions that, if they become true, expire the permissions. We will subsume ODRL constraints and requirements as *rules* and further partition them into *static rules* and *dynamic rules*. For example, *static rules* could mandate that an action can only be performed by members of a certain group, on certain hardware, after a certain time has passed, or that the user must pay some fee before taking the action.

5.3.1. Content Protection

In detail, the encryption process, executed by media distributors, contains the following phases (see also Figure 5.5 for a visualization of the process):

1. **XML Extraction.** This phase takes as input an ODRL expression Pol and a set of conversion rules that map cryptographically enforceable ODRL rules within Pol to a single attribute policy. It outputs a Boolean expression in the form of a circuit containing a predicate $p(\mathbb{K})$ that describes the enforceable sub-policy of Pol. It also outputs a modified ODRL expression Pol'. In Pol', all static attribute nodes are marked with a special XML attribute, so that the DRM viewer knows that they do not need to be enforced at runtime.

2. **Attribute Expansion.** The Boolean expression created by the XML extraction phase may contain numerical and date/time attributes and numerical comparisons. As ABE constructions usually support only Boolean attributes and comparisons, these numerical attributes need to be suitably encoded, see Section 5.3.4. The result of the first two steps is a logical expression that can be represented as a syntax tree.

3. **Policy Finalization.** The final steps are dependent on the DABE construction that is used. The tree obtained in the last phase is optimized (e.g. by combining adjacent AND/OR nodes) and converted to the required

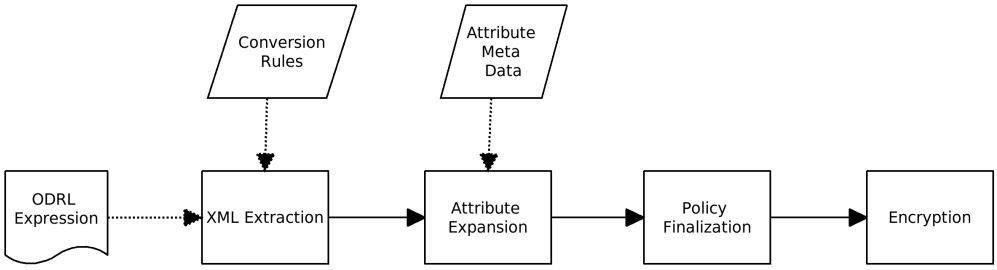


Figure 5.5: ODRL expression conversion process

access structure format (i.e., DNF [KTS07, CN07, MKE08], access trees [BSW07, BKP09, GJPS08, LCLX09], or share-generating matrices of linear secret-sharing schemes [Wat11]). Call the result $\mathbb{A}$.

4. **Encryption.** The media distributor retrieves public keys for all attributes used in $\mathbb{A}$ from the respective attribute authorities. The distributor executes the algorithm $\mathit{Encrypt}(PK, K_T, \mathbb{A}, PK_{\mathcal{A}_1}, \ldots, PK_{\mathcal{A}_N})$ with the random title key K_T and the public attribute keys as input and receives as output a ciphertext CT. Subsequently, he encrypts the plaintext media m with the title key K_T and publishes the encrypted media together with K_T, the license and access policy:

$$\langle E(K_T, m), CT, \mathbb{A}, Pol' \rangle .$$

In our implementation we decided to also include statically enforced rules in the modified license, but mark them with a special XML tag. This gives the DRM viewer additional information about what is already enforced by the cryptographic components, (i.e., what attributes the decryptor is proven to have) so that the entire license is accessible by the user.

5.3.2. Parsing Agreements

An ODRL expression consists of offers and agreements; agreements are used as DRM licenses and are thus the only components that have to be enforced at runtime. Agreements are XML blocks enclosed in `<agreement>` ... `</agreement>` tags. An agreement consists of a set of permissions, where a set of rules can be associated with each permission. Figure 5.6 shows an example permission that

```
<permission>
  <print>
    <requirement>
      <prepay>
        <payment>
          <amount currency="USD">20</amount>
        </payment>
      </prepay>
    </requirement>
  </print>
</permission>
```

Figure 5.6: Sample permission with rule

allows printing of a document once an amount US$ 20 has been payed in advance. This rule is an example of a static rule, as an attribute rule

```
media1234.hasPayedFor.USD >= 20
```

can be added to the attribute policy $\mathbb{A}$, where `1234` is the identifier of the media. For permissions with multiple rules, all rules must be honored, so the enforable rules in the attribute policy must be combined with the `AND` operator.

ODRL supports the declaration of various permissions inside a single permissions block, corresponding to different rule sets associated with different actions that can be performed alternatively. We can not control the type of action that a user performs once the decryption has taken place, but obviously he has to fulfil at least one rule set to be able to decrypt. Thus we allow decryption of the title key if *any* of the rule sets is satisfied (all other parts of the license must be enforced at runtime). To achieve this, the rule sets of different permissions are connected by an `OR` operator to produce the policy $\mathbb{A}$. Formally, given a set of permissions P_i, where each permission contains enforceable rules $R_{i,j}$, the generated attribute policy has the form

$$\bigvee_i \bigwedge_j R_{i,j}.$$

Note, however, that some $R_{i,j}$ could themselves become Boolean expressions after the Attribute Expansion phase (see Section 5.3.4), so the final access policy might not yet be in DNF.

```
constraint/group/context/uid/(\w+)
 =>    user.groupname.uid{str[0]}

requirement/prepay/payment/amount\[@currency=
"(\w+\?)"\]/(\d+)
 =>    media{id}.hasPayedFor.{str[0]} >=
  {int[1]}}

constraint/spatial/constraint/datetime/
start/\(\d\d\d\d-\d\d-\d\dT\d\d:\d\d:\d\d)
 => datepassed.{year[0]}.{month[0]}.{day[0]} OR
   datepassed.numerical >= {numericaldate[0]}
```

Figure 5.7: Sample Conversion rules

5.3.3. Path Conversion

We automate the extraction of rules from ODRL permissions by conversion rules that map an ODRL rule to an attribute.

Consider, for example, the rule shown in Figure 5.6. The ODRL rule can be written in its XPath form as `permission/print/requirement/prepay/payment/amount[@currency="USD"]/20`. To find all rules of this form within an ODRL expression, we can use a regular expression: `requirement/prepay/payment/amount[@currency="(.+)"]/(.+)`. This expression will match all ODRL rules of the type `prepay`, and output two *match groups*: One that represents the currency as a string ("USD" in our example), and one that represents the amount (20 in our example). We can subsequently use these match groups to automatically create the above rule in the attribute policy.

Our conversion tool takes as input a set of conversion rules that define mappings of XPath-like regular expressions to attribute names. Figure 5.7 shows two example conversion rules as understood by our framework. Each rule conists of two parts, separated by the character sequence `=>`. The first part of each rule is a regular expression over an XPath, describing a node in a permission subtree. The second part of each rule describes the destination attribute for any rule matching the regular expression. This destination attribute can contain patterns enclosed in `{}` brackets that will be replaced by values of the match groups. Table 5.1 summarizes these patterns. Note that we are using Perl regular expressions, which offer some additional features over conventional regular expressions. For example, in a Perl

Pattern	Replaced by
id	Media ID
uid	UID of current context block
str[i]	String representation of i^{th} group
int[i]	Integer representation of i^{th} group
year[i]	Year component of i^{th} group
month[i]	Month component of i^{th} group
day[i]	Day component of i^{th} group
numericaldate[i]	Numeric date representation of i^{th} group

Table 5.1: Substituting match groups in destination attributes

regular expression, the substring `\d` can be used to represent a single decimal digit. In a conventional regular expression, this would have been written as `[0-9]`.

For example, the first rule of Figure 5.7 maps an ODRL element of type `group`, described further by a context field `UID`, to an attribute that represents the viewer's desired group membership. The term `{str[0]}` of the second part of the rule will be replaced by the string representation of the first match group, which is matched by the term `(\w+)` of the regular expression. Consider again the ODRL expression from Figure 5.6. The XML subtree inside `permission/modify` matches the second conversion rule of Figure 5.7.

5.3.4. Representing ODRL Rules by Attributes

In this section we show how enforceable ODRL rules, as encoded by the extraction process described above, are represented as attributes. Enforceable attributes deal with properties of the user who tries to access the media, as his identity, affiliation, age, role, and group membership. Alternatively they could be connected to the document itself, like payments made for purchase. Table 5.2 lists all ODRL rules that can be expressed as static rules.

In our sample implementation we use a systematic naming scheme where attribute names incorporate URLs of attribute authorities, components describing ODRL context blocks, and unique identifiers.

The names also support attributes that can represent integers. For many scenarios it might be enough to use Boolean attributes to represent numerical values. For example, if a license requires the payment of US$ 20, inclusion of an attribute `media{id}.hayPayedFor.USD20` is enough to enforce the rule. However,

ODRL Name	Attribute
Individual	`user.<context>`
Group	`user.groupmember.<context>`
CPU	`device.cpu.<context>`
Network	`device.network.<context>`
	`device.network.ip`
Screen	`device.screen.<context>`
Storage	`device.storage.<context>`
Memory	`device.memory.<context>`
Printer	`device.printer.<context>`
Hardware	`device.hardware.<context>`
Software	`device.software.<context>`
Date Time	`datepassed.<year>`
	`datepassed.<year>-<month>`
	`datepassed.<year>-<month>-<day>`
	`datepassed.numerical`
Prepay	`media<id>.hasPayedFor.<currency>`
Accept	`media<id>.accepted`
Register	`media<id>.registered`

Table 5.2: Static rules of the ODRL standard with standardized attribute mappings

ODRL allows very flexible rules that may, for example, restrict the number of pages that a user can read depending on the concrete amount of money that he has payed. Thus, the attributes should be represented in a way to allow numerical comparisons within the attribute policy $\mathbb{A}$. We thus encode integers as "bags of bits", as described in [BSW07]: In a bag of bits representation a number is represented as a binary string of fixed length. For each of these bits there exist two attributes. One of these represents a binary 1 in the respective position, one represents a 0. The syntax that we use is

$$attributeName\texttt{[}bitPosition\texttt{]}.bitValue\ ,$$

where *bitPosition* is the index of the bit in the bit string and *bitValue* is either 1 or 0, depending on the value of the respective bit. For example, if someone possesses a secret key for the attribute `x[2].0`, then the binary representation of his integer attribute x has a binary 0 at position 2. If x has a bit length of 3, a complete set of attributes for x could be, for example,

`x[2].0`, `x[1].1`, `x[0].1`,

meaning that the attribute x has the value $011_2 = 3_{10}$ for the respective user. To encode an ODRL expression that represents a numerical comparison, for example,

the expression $x \geq 3$, we can use a Boolean expression over these bit attributes:

```
x[2].1 OR (x[1].1 AND x[0].1) .
```

To fulfill this expression, a decryptor could either have the bit number 2 set (then x would be at least 4), or both bits 1 and 0 (then x would be 3).

Attributes that deal with date or time can be encoded in various ways. There could be Boolean attributes that are only issued after some point in time has passed (this approach is similar to using a key K_D as mentioned in the introductory example), or the time could be encoded numerically in the way described above.

5.3.5. Example

Figure 5.8 shows a complete ODRL expression, and Figure 5.9 shows the resulting attribute policy after XML Extraction (reformatted for better readability). Note that the second permission (the `<o-dd:print>` block) is the same as the example of Figure 5.6, which was already discussed. Here, the media ID was taken from the `<asset>` block near the beginning of the ODRL expression. In the Attribute Expansion phase, the two numeric comparisons (`datepassed.numerical >= 20011231` and `mediasamplemedia.hasPayedFor.USD >= 20`) will be expanded to their bags of bits representation.

5.4. Implementation

The conversion process was implemented during a bachelor's thesis project [Bra10]. Our tool takes as input an ODRL policy and a set of conversion rules as described in the preceding sections. It converts the XML file to its DOM representation and parses it, listing all rules contained in its agreement blocks, and identifying which of them are enforceable according to the given conversion rules. In the next step it applies the conversion rules and outputs all resulting attribute rules. The tool is also able to convert rules containing a comparison of numerical attributes with constants to comparisons with "bag of bits" representations. For example, Figure 5.10 shows a subtree of a set of static rules before the bags of bits conversion, and Figure 5.11 shows the same tree after a bags of bits conversion. Briefly, in this example, the numerical attribute `device.network.uidnet1.version` is represented as a three bit value (encoded by six Boolean attributes), and the left side of the tree evaluates

```
<?xml version="1.0" encoding="UTF-8"?>
<o-ex:rights xmlns:o-ex="http://odrl.net/1.1/ODRL-EX
    xmlns:o-dd="http://odrl.net/1.1/ODRL-DD">
<o-ex:agreement>
  <o-ex:asset>
    <o-ex:context>
      <o-dd:uid>samplemedia</o-dd:uid>
      <o-dd:name>A Sample Media</o-dd:name>
    </o-ex:context>
  </o-ex:asset>
  <o-ex:permission>
    <o-dd:play>
      <o-ex:constraint>
        <o-dd:group>
          <o-ex:context>
            <o-dd:uid>samplegroup</o-dd:uid>
          </o-ex:context>
        </o-dd:group>
        <o-dd:datetime>
          <o-dd:start>2001-12-31T00:00:00</o-dd:start>
        </o-dd:datetime>
      </o-ex:constraint>
    </o-dd:play>
    <o-dd:print>
      <o-ex:requirement>
        <o-dd:prepay>
          <o-dd:payment>
            <o-dd:amount o-dd:currency="USD">
              20
            </o-dd:amount>
          </o-dd:payment>
        </o-dd:prepay>
      </o-ex:requirement>
    </o-dd:print>
  </o-ex:permission>
</o-ex:agreement>
</o-ex:rights>
```

Figure 5.8: Example ODRL expression

```
(
 (
  user.groupmember.samplegroup AND
   (
    datepassed.2001-12-31 OR
    datepassed.numerical >= 20011231
   )
 ) OR
 mediasamplemedia.hasPayedFor.USD >= 20
)
```

Figure 5.9: Attribute policy after XML Extraction phase

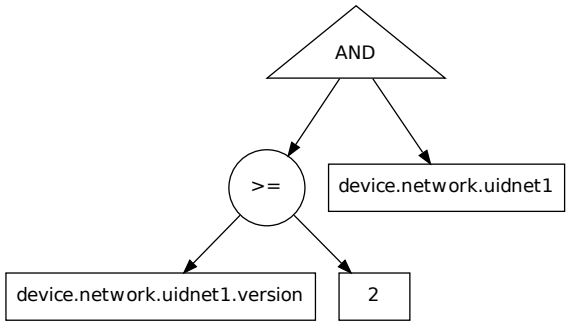


Figure 5.10: Static rule as tree

to true, if any of the upper two bits (with index 1 and 2) is set to true.

In the next step, several optimizations of the tree are performed, like joining adjacent AND/OR nodes or removing unneeded nodes. Finally, in order to be usable with an existing implementation of the DABE construction of Section 3.4, the tree is converted to its DNF representation. Due to the simple structure of ODRL policies this final conversion can be done efficiently, and the resulting DNF trees are usually quite small.

The implementation was written in Java, using JDOM to access the DOM representation of ODRL policies and SableCC [Éti98] as a parser generator, which was needed to process the large number of more than 100 conversion rules that were created to support all relevant fields of ODRL. For details, we refer the reader to Appendix B of the bachelor thesis [Bra10].

Data type	Attribute type	Example
Numerical	Numerical	The user has paid 10 €
Date	Numerical	The content is usable only after January 1st, 2011
Boolean	Boolean	The user is a member of group “Samplegroup”

Table 5.3: Data types of implemented tool

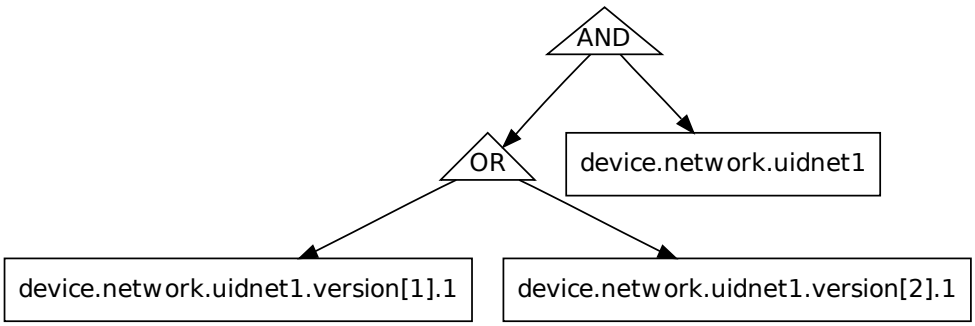


Figure 5.11: Static rule of Figure 5.10 after bags of bits conversion

5.5. Conclusion

In this chapter, we proposed a new DRM architecture that uses Distributed Attribute-Based Encryption to enforce policies by cryptographic means. We argued that this not only allows stronger security claims, as we do not rely on a trusted reference monitor anymore, but may in many cases also remove the necessity of a user license creation process by the rights issuer by instead using the publishing license. This greatly simplifies the architecture of a DRM system and reduces network load.

Taking OMA DRM as an example architecture, we have then identified those ODRL expressions that can be cryptographically enforced through ABE and proposed a framework that automatically extracts the enforceable policies from ODRL expressions as Boolean attribute policies and encodes them. This allows to implement a DRM scheme where an enforceable attribute policy is automatically extracted from an ODRL expression. This attribute policy can in turn be used as input to a DABE scheme which encrypts the title key with the policy.

We also implemented an extraction utility that can be used as part of a complete

DRM framework and described implementation details, showing specifically how such a conversion process can be done.

Chapter 6

Attribute-Based Encryption in SOA

6.1. Introduction

The application of attribute-based encryption is particularly attractive in distributed settings. DABE shows its strengths when a large number of parties are present and central trusted authorities are undesired. Such settings are commonly found in Service-Oriented Architectures (SOA). In SOA, functionality is provided through standardized *services*. A service is initiated by a *service request* message that is sent to a *service provider* and contains all information about the request. This request triggers the functionality on the provider's site. The provider then replies with the result of the service with another standardized message, the *service result*. (This chapter will only briefly introduce SOA concepts. For a more thorough introduction, we refer to [Erl08].) Obviously, the results of a service request must adhere to certain privacy requirements. In this chapter we use the SOA setting as an example to take a more detailed look on how data-centric security with DABE can be implemented. (Note that many of the results here are also applicable to other variants of ABE.) We describe technical details like attribute revocation and the creation of LSSS matrices that we have mentioned only briefly in the preceding chapters.

The core object of SOA is the SOAP message [W3C07]. Such messages usually contain data, i.e., documents, but may also have RPC-like information (i.e., active objects). In a typical SOA setting, all messages exchanged between parties follow the SOAP standard, and from a security perspective it should be ensured that

SOAP messages are only accessible to eligible users. For example, if a service requester is a company that uses a service to acquire some sensitive information that is needed for a certain project, then the service results should be accessible by some, but not all employees of the company. These eligible employees can be described by a policy. However, due to the highly distributed nature of SOA scenarios and the potentially very high number of requests, a central policy enforcement point would quickly become a bottleneck.

It appears natural to utilize ABE to improve both efficiency and security in SOA, but several obstacles make this difficult: In this chapter, we show that DABE, as opposed to conventional CP-ABE fits the highly distributed setting of SOA very well.

For the SOA setting, we also take a look at the of problem of attribute revocation: When a property of a user, represented by one of his attributes, changes, he must lose the ability to access any encrypted message for which the revoked attribute would be needed. While there are some constructions that support variants of this idea [YWRL10b], they are limited in several ways, and it is unclear what kinds of revocation are actually possible in the specific settings of Attribute-Based Encryption. We show that although attribute revocation is problematic under the flexible scenario of the ABE schemes, DABE allows for a deterministic and well-defined revocation mechanism, and we describe the infrastructure needed for it.

WS-Security is commonly used to describe access control policies [JSGI09]. However, the WS-Security standard [OAS06] does not take into account the special features and implications of cryptographic types of access control, such as ABE. Nevertheless, we show that WS-Security is well-suited for ABE policies and can easily be extended to allow DABE policies to be embedded in WS-Security.

Finally, we show that nearly all expensive calculations of common ABE schemes can be outsourced to third parties which can help the decryption without gaining any information about the encrypted messages. These third parties will be denoted *access gates* as they are able to enforce a policy (i.e., do access control) but need not be completely trusted. Even if these gates are corrupted or under full control of an attacker, they are not able to violate the policies encoded in the ciphertexts, and no information about the encrypted messages is leaked. Of course, an attacker could shut down corrupted access gates completely, so they do not offer any functionality

anymore. In that case, a decryptor can choose a different access gate. In regard to that property, we model access gates as honest-but-curious for our analysis, so we consider them willing to do the functionality but interested to gain as much information as possible. We show that no part of the encrypted data is revealed to access gates.

6.1.1. Related Work

In his dissertation, Shucheng Yu [Yu10] discussed the applicability of ABE for cloud computing (also published as [YWRL10a]), and his work was later extended by Jeong-Min Do et. al. [DSP11]. This setting bears some similarities to SOA. However, only KP-ABE was considered in these papers. A cloud computing setting was also considered in [LWG11]. In the short paper [LbHC10], it is briefly shown how conventional single-authority CP-ABE can be included in WS-Security. We expand on these ideas.

When using ABE for any setting, one must choose a specific construction that implements all required features. In our SOA setting we use DABE, and more specifically the construction of Section 3.5.

We note that in [LW11], the authors propose a CP-ABE scheme similar to DABE that does not need any central authority as there are no secret user keys. We do not consider this a useful addition in most practical scenarios for the following reasons:

1. Everybody who gets a suitable set of attribute keys is able to decrypt messages, so all attribute keys must be delivered through a secure (encrypted) channel. In DABE systems with a secret user key, the secret attribute keys can be delivered unencrypted, as they are of no use to anybody who has no access to the secret user key. We will use this feature later in this chapter when we introduce *Access Gates*. This means that while [LW11] offers the advantage of requiring one key less (the secret user key) than DABE, it suffers from the disadvantage of having all other keys delivered in a much more expensive way.

2. For most practical scenarios, a user's eligibility of an attribute requires authentication, so each user needs to have an identity that can be verified

by using a trusted authority. Also, to build a secure channel, the decryptor needs a unique key which cannot be maintained outside of the system, so the proposed advantage seems to be only of theoretical nature.

3. The ciphertexts are 50% larger as three group elements are required for each row of the LSSS matrix, whereas the construction we use requires only two group elements per row. Also, for the reduction proof to work, the construction requires very large groups of order $p_1p_2p_3$ for p_1, p_2, p_3 being large primes. This makes all group elements much larger.

Outline. The rest of this chapter is structured as follows: First we discuss attributes in a global setting in Section 6.2. We take a detailed look on how to implement the crucial functionalities of encryption (Section 6.3) and decryption (Section 6.4). Finally, we describe the complete SOA system in Section 6.5. We conclude the chapter in Section 6.6.

6.2. Attributes

6.2.1. Unified Naming Scheme

In a multi-authority setting as described by DABE, each attribute authority is responsible for its own universe of attributes and the encryptors decide which authorities they trust (see our description of DABE in Section 3.2). From a technical point of view, each authority may have its own way of naming attributes.

Nonetheless, from a usage point of view it may be desirable to have a unified naming scheme that all authorities follow. Consider the naming scheme we used in the preceding chapter, for example

```
device.network.ip
```

Here, the naming follows a hierarchical pattern, where the name starts with a specifier of the domain (i.e., `device`, meaning that the attribute is concerned with a device), followed by a specifier of a more specific sub-domain (i.e., `network`), followed by the name (i.e., `ip`).

We propose a similar naming scheme for the DABE setting. Attribute names are prefixed by the authority URI to make them unique. The delimiters shall be `/`

characters instead of dots. Similar to web services we also prepend the `http://` prefix to all attribute names, making each attribute name a valid URL. Such a URL could be used to point to resources that provide information about the attribute, for example explaining what property it represents, its semantics and how a user can prove his eligibility. We discuss this process later in Section 6.4.1.

For example, an attribute that specifies the memory size in gigabytes and is managed by an attribute authority identified by `http://www.someauthority.com` might look like this:

`http://www.someauthority.com/device/memory/size/gb/`

Note that we also added an additional slash character at the end. This last `/` marks the end of the attribute name. We will later use this feature to append additional information about the attribute (that is not part of the name) after that character.

It is noteworthy that we consider the name of the attribute authority part of the attribute name. For example, an attribute

`http://www.authority1.com/user/role/admin/`

while having a similar meaning is different from the attribute

`http://www.authority2.com/user/role/admin/`

In both cases, the user is supposed to be an administrator, but probably for different domains. He may request the `admin` attribute from the respective authority, but it only certifies his role within the specified domain.

Numerical attributes. Inspired by the Hungarian notation [Klu88] we propose to append the information on the bit length to the attribute name, followed by the bags of bits qualifiers that we introduced in Section 5.3.4. A complete attribute name might then look like this:

`http://www.someauthority.com/device/memory/size/gb/i8[2].1`

This specific attribute specifies that the bit with index 2 of the 8 bit value is set.

6.2.2. Revocation

Revocation is frequently used in the context of Digital Rights Management infrastructures. Here, certain user keys can be revoked, making the respective users unable to access (i.e., decrypt) documents. This is very useful because over time user properties change and user may lose some of their access rights. We now discuss how this can be applied to ABE.

In DABE, access is restricted not by users but by their properties, so the meaning of revocation changes: If a user loses a certain property, this property is called *revoked*, and it the future, he must be unable to access objects that require this property, even if he was able to access them before the revocation took place. In settings with a single trusted authority that controls all accesses, this is easy to accomplish as the authority can simply deny users access.

For ABE, where an access attempt is equal to a decryption attempt, things get a bit more complicated. In [YWRL10b], the authors try to solve the revocation problem in the ABE case by proposing a CP-ABE construction that supports efficient proxy re-encryption. The idea is that whenever an attribute gets revoked, the master key components of that attribute are changed and all stored ciphertexts are re-encrypted, forcing future users to request an updated secret attribute key. This new attribute key will only be given to users that are still eligible. In effect, a new attribute is introduced that takes on the role of the old one but is only given to non-revoked users. However, this solution is negligent of the fact that in the ABE setting, ciphertexts are stored on untrusted media. (Note that if the storage is on trusted media, there is usually no need for a cryptographically expensive technology as ABE, as trusted storage entities can also be trusted to do conventional access control.) Untrusted storage can not be trusted to delete the old versions of the ciphertexts that can still be decrypted with the old attribute keys. This observation about revocation of data stored on untrusted media is true for most revocation mechanisms. Even in broadcast encryption schemes [NNL01] commonly used in DRM architectures[1], revocation is only effective for data created *after* the revocation event took place.

While we cannot circumvent this limitation, we can use the idea of changing attribute keys in order to revoke attributes, but note that this only has an effect

[1]Such as AACS, `http://www.aacsla.com/specifications/`

on newly encrypted content as we cannot be sure that old data is always re-encrypted. Our revocation mechanism works as follows: Whenever an attribute is revoked for a set of users, the respective attribute authority creates a new attribute key pair. This should be reflected in the attribute name, for example, by appending a sequential number. For example, an attribute that represents a user's eligibility to access movies of a site could be initially called `moviedownload/1/`. The respective keys are given to all eligible users. After a revocation, a new attribute called `moviedownload/2/` is issued. From that point in time, the old attribute `moviedownload/1/` is not used anymore, and all newly created movies are encrypted using the new keys. The corresponding secret attribute key is given to all users that are still eligible, but not to revoked users. This way, revoked users cannot access the newly encoded documents anymore. Note that to implement this efficiently, the underlying ABE construction is required to support incremental attribute claims (DABE explicitly is).

Instead of sequential numbers, it can be more useful to periodically update attributes and reflect the validity period in the attribute names. For example, an attribute name could look like this:

```
http://www.someauthority.com/subscription/moviedownload/2011-10/
```

A user who has an attribute key corresponding to this attribute is eligible to download movies encrypted in October 2011. If the user's subscription ends in October, he is not able to decode movies encrypted after that date anymore, as he will not get the corresponding key for November. The effect of this is that his subscription property is revoked.

6.3. Encryption

6.3.1. Incorporating CP-ABE into WS-Security

In [LbHC10], the authors propose a SOAP format that supports encoding data directly encrypted with CP-ABE. Unfortunately, the approach is limited: ABE is computationally very expensive and usually described in terms of encrypting a single group element of a group $\mathbb{G}_T$. For all practical purposes, one will use a hybrid encryption where only a symmetric key is encrypted with ABE, and

the actual data is encrypted symmetrically with that key. Also, instead of using established XML and SOA standards, the authors introduce their own headers.

Our approach is as follows: The core standard that allows for security claims in SOAP messages is WS-Security [OAS06]. (For a good introduction to WS-Security we refer to [KC08], which we implicitly use as source for most of the technical claims in this section.) To support encrypted documents, WS-Security commonly uses the W3C recommendation *XML Encryption Syntax and Processing* [IDS02] that supports several powerful ways to encrypt XML data, including hybrid encryption properties by using an element called `EncryptedKey`. This element is used to transport (usually symmetric) encryption keys. The information on how to decrypt such an `EncryptedKey` can be stored in an element called `EncryptionProperties`.

Note that there are standards that deal with general policies (i.e., *WS-Policy* [W3C06]) and security policies (*WS-SecurityPolicy* [LK07]), but we do not use these, as we are only concerned with *access policies written as Boolean formulas* and these standards were tailored for a much more general case.

Figure 6.1 shows an example of a SOAP message encrypted with DABE using the policy discussed earlier in Section 3.1. Briefly, the SOAP header (lines 8–29) contains a symmetric key encrypted with DABE using the policy depicted in lines 9–14. For simplicity we omit the public keys of the system and instead consider a system were the public key (including the URI of the Central Authority) is publicy known. The `EncryptedKey` element references both the policy (line 25) and the actual encrypted message (line 22). After decrypting the symmetric key, it can be used to decrypt the actual message that is contained in the SOAP message body (lines 31–41). In this example, AES was used for symmetric encryption (see line 33).

In our architecture, the decryptor processes the policy and examines if he can satisfy it. If he needs to have more attributes, he invokes a web service to query an attribute authority for a specific secret attribute key, for example for the attribute `http://www.openid.org/user/is18OrOlder`. We get into more detail on this in the following section. If he satisfies the policy, he decrypts the symmetric key using the DABE construction and then uses the symmetric key to decrypt the actual message. How such an encrypted message is encoded is specified by the core standards and can be done with conventional web service tools.

```
1  <?xml version="1.0" encoding="UTF8"?>
2  <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap
       /envelope/">
3  <SOAP-ENV:Header>
4   <wsse:Security
5     xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-...
6     xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
7
8    <wsse:BinarySecurityToken Id="abe-policy">
9     http://db.mycompany.org/user/role/admin OR
10    http://db.mycompany.org/user/permissions/fullAccess OR
11    ( http://www.openid.org/user/is18OrOlder AND
12     ( http://www.contprov1.com/article/byid/1234/hasPaidFor OR
13       http://www.contprov2.com/article/byid/4325/hasPaidFor OR
14       http://www.contprov3.com/article/byid/ABC/hasPurchased ) )
15   </wsse:BinarySecurityToken>
16   <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#
         ">
17    <xenc:EncryptionMethod Algorithm="wsdabe" />
18    <xenc:CipherData>
19     <xenc:CipherValue>
          sMqPjXpGrM24SsmP8NVXJ1WhWuudpim9TjtsNzeWvrShNgytPoYx6GH0v27xvZu
          </xenc:CipherValue>
20    </xenc:CipherData>
21    <xenc:ReferenceList>
22     <xenc:DataReference URI="#Enc1"/>
23    </xenc:ReferenceList>
24    <wsse:securityTokenReference>
25     <wsse:Reference URI="#abe-policy" />
26    </wsse:securityTokenReference>
27   </xenc:EncryptedKey>
28  </xsse:Securiy>
29 </SOAP-Env:Header>
30
31 <SOAP-Env:Body>
32  <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#
       " Id="Enc1" Type="http://www.w3.org/2001/04/xmlenc#Content">
33   <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/
        xmlenc#aes-192-cbc"/>
34   <xenc:CipherData>
35    <xenc:CipherValue>
36     cU5vDITPk78thObr7AxoEsHTkdp2VOsTDSJEnkNMMj4jbhkdRUpNK6QbcEAoM
37     RvYVEZDPKbCPrBfvvyMkCuXNkUJ6T5YN1DSKKoooG8zNXNsUHcTm9ghX8Dr67
38    </xenc:CipherValue>
39   </xenc:CipherData>
40  </xenc:EncryptedData>
41 </SOAP-ENV:Body>
42 </SOAP-ENV:Envelope>
```

Figure 6.1: Sample DABE-encrypted SOAP object

6.3.2. Preparing an Encryption

In order to protect a document, the encryptor must decide on the policy, and for each attribute he must choose one attribute authority he trusts that is able to specify the attribute. As described in Chapter 3, he needs public attribute keys for all attributes that he uses in the policy. Note that if a single-authority scenario is deployed and no incremental attribute claims are required, a conventional CP-ABE construction like [BSW07] may be used. In such single-authority CP-ABE constructions, public attribute keys are generated by computing a publicly know hash function $H : \{0,1\}^* \to \mathbb{G}$ (with $\mathbb{G}$ a bilinear group) that maps an attribute name to a group element.

In this chapter, we realize attribute claims through web services. Figure 6.2 shows an example for an attribute claim in SOAP for the attribute `http://www.authority1.com/user/role/admin/` (see line 10). The web services are defined in a file `wsdabe.wsdl` that is referenced in line 7 of the code. The result of such a claim is shown in Figure 6.3. It consists of a structure of type `ws:publicAttributeKey` that has the fields `name` (the name of the attribute) and `pka` (the actual public key encoded as base 64 of a string representation of the key[2]). The encryptor can decode this to a binary string and use this binary string to create a new `element_t` value that represents the public attribute key $PK_{\mathcal{A}} = g^{H_{SK_a}(a)}$ for attribute $PK_{\mathcal{A}}$. This value can be used as input for the encryption function.

6.4. Decryption

6.4.1. Preparing a Decryption

Managing identities and user keys. Everybody can request a user key pair from the central authority. In our framework, there is a single service `getUserKeyPair` that takes no argument and returns a new user key pair. It is the only service offered by the central authority. There must be a connection between the public user key PK_u and the user identity u. For example, the user could sign PK_u using a signature key that is bound to him by means of a certificate that is propagated through an established PKI. It is also possible to bind PK_u to some pseudonyms

[2] In our implementation that is described in Section 6.5.4, we use the return value of `libpbc`'s function `element_to_bytes`.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ws="http://localhost:8080/wsdabe.wsdl">
 <SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org
    /soap/encoding/">
  <ws:getPublicAttributeKey>
   <name>http://www.authority1.com/user/role/admin/</name>
  </ws:getPublicAttributeKey>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 6.2: Sample public attribute key request

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ws="http://localhost:8080/wsdabe.wsdl">
 <SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org
    /soap/encoding/">
  <ws:publicAttributeKey>
   <name>http://www.authority1.com/user/role/admin/</name>
   <pka>RGllIEhvZmZudW5nIGlzdCBkaWUgU+
      R1bGUsIHdlbGNoZSBkaWUgV2VsdCB0cuRndC4</pka>
  </ws:publicAttributeKey>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 6.3: Sample public attribute key result

in order to achieve anonymity when dealing with certain services.

To be able to do this, a slight modification is required. This modification works for both DABE constructions we proposed in Chapter 3: For each ith pseudonym a user u wants to create, he chooses a random $b_i \in \mathbb{Z}_p^*$ and computes his ith public user key $\mathrm{PK}_u^{(i)}$ as $PK_u^{(i)} := PK_u^{b_i}$. For example, a user could create an (anonymous) account on a license provider site and associate it with his second pseudonymous public user key $PK_u^{(2)} = PK_u^{b_2}$ for some random b_2. Whenever he buys something with this account, the license provider site offers an attribute representing this purchase and personalizes it for $PK_u^{(2)}$, returning a secret attribute key $SK_{\mathcal{A},u}^{(2)} = (PK_u^{(2)})^{H_{SK_{\mathcal{A}}}(\mathcal{A})}$ (as explained in the *RequestAttributeSK* algorithms). The user can now raise this value to the power b_2^{-1} to retrieve a valid secret attribute key $SK_{\mathcal{A},u}$ for his real user key PK_u without ever having revealed PK_u to the attribute authority:

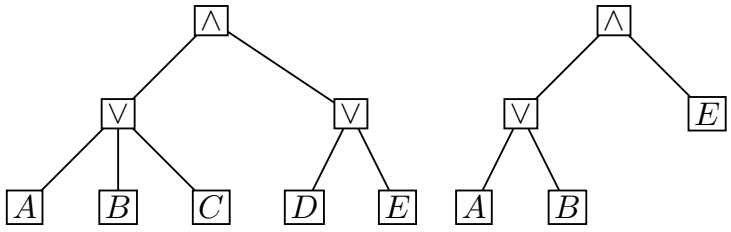
$$(SK_{\mathcal{A},u}^{(2)})^{b_2^{-1}} = (PK_u^{(2)})^{b_2^{-1} H_{SK_{\mathcal{A}}}(\mathcal{A})} = PK_u^{H_{SK_{\mathcal{A}}}(\mathcal{A})} = SK_{\mathcal{A},u}\,.$$

The effect of this is that although the user u does not reveal his true identity to the site, he is still able to use the attributes together with his other attributes. Getting attributes using different identities is desirable in many scenarios. For example, an attribute like `http://www.openid.org/user/is18OrOlder` probably requires the respective attribute authority to know the user's real identity, while he might prefer to use only a pseudonym when acquiring other attributes. Due to the DABE construction, the user can freely mix attributes gained under his true identity with attributes gained under pseudonyms.

Requesting secret attribute keys. Using his public user key the user can now request secret attribute keys. To issue these secret user keys, an attribute authority must have a way to decide if user represented by his PK_u is actually eligible of the attribute. As we have described, this decision can be based on the user identity or on a pseudonym. How this works in particular may vary greatly between the many different possible types and domains of attributes, so the details are beyond the scope of this work. However, in general we can describe this process as a communication protocol where the authority request a set of information from the user that proves his eligibility. This protocol may contain, for example,

```
<message name="getSecretAttributeKey">
 <part name="name" type="xsd:string" />
 <part name="usertoken" type="xsd:string" />
 <part name="pku" type="xsd:string" />
</message>
```

Figure 6.4: WSDL definition of `getSecretAttributeKey` service



(a) Original policy syntax tree (b) Missing attributes

Figure 6.5: Determining missing attributes

challenge-response steps that let the user prove his possession of certain keys or other means of convincing the authority of a property. The last step is the sending of the actual secret attribute key. Figure 6.4 is a WSDL description of such a final message, denoted `getSecretAttributeKey`. It contains the `name` of the attribute as well as some `usertoken` that identifies the result of the communication protocol and the public user key `pku` of the user. The `usertoken` may be as simple as a user identity or the session key of a session opened between the authority and the user, where he proved his eligibility. In order to have the greatest possible versatility, we do not limit the format of this token.

Determining how to satisfy the policy. In a typical system, a user may be eligible to a large number of attributes, but he only needs to request secret attribute keys required for the ciphertexts he wants to decrypt. Whenever a user receives a ciphertext, he analyses the policy to see if his current set of secret attribute keys satisfies it or if he needs to request more keys.

If he does not satisfy the policy, it is useful to give him some representation describing possible steps that he can do to satisfy it. Such a representation can directly be done by removing all parts of the policy that the user already satisfies.

To illustrate this, we give a short example: Consider the syntax tree in Figure 6.5(a) that represents a ciphertext policy and assume that the decryptor only

has secret attribute keys for the attributes C and D. The decryptor automatically removes these nodes (by replacing them with $\top$ and simplifying the resulting tree) and gets the tree shown in Figure 6.5(b). It shows not only that the policy is not (yet) satisfied, but also tells the decryptor that to be able to decrypt, he needs to acquire secret attribute keys for either both A and B or for E. This tree may be displayed to the decryptor to help him decide on his next steps.

6.4.2. Access Gates

When a ciphertext policy is satisfied, the ciphertext can be decrypted. As decryption is the most expensive process of ABE constructions, we now briefly take a look at the computational cost of this process and propose a new concept that allows to improve the efficiency in many cases:

Implementations of pairing-based cryptosystems on smart cards [SCA06], mobile phones [KTO06], and even wireless sensor nodes [SKSC09] have shown that the technology is feasible. However, the actual pairing operation is about 150-500 times slower than even inversion, which is the slowest commonly used operation within public-key cryptography (see especially Table 1 of [KTO06] and Table 5 of [SKSC09]), a single pairing operation being approximately as expensive as a complete RSA exponentiation [SCA06]. While we have already proposed a CP-ABE construction that requires only two pairing operations in Section 3.4, in all other known constructions the number of pairing operations in the decryption function is in $O(n)$, where n is the number of attributes used for the decryption. (Note that in some constructions like [NYO09] *all* attributes are used for decryption, so in these cases n denotes the number of attributes in the system.)

As the parties accessing ABE protected documents may be light-weight, it is important to look for ways to speed up decryption. In this section, we show that in most common CP-ABE constructions decryption can be seen as a two-step process and that the first step, which is the computationally expensive part, can be done by a third party without compromising security. We consider such third parties as "honest, but curious" as introduced in [dVFJ+07] and denote them *Access Gates*. Figure 6.6 gives an overview of the involved parties and their relationships.

We note that these findings apply to many CP-ABE constructions, for example all of [MKE09, LOS+10, NYO09, ZH10, CN07]. (Recently, in [GHW11], the authors

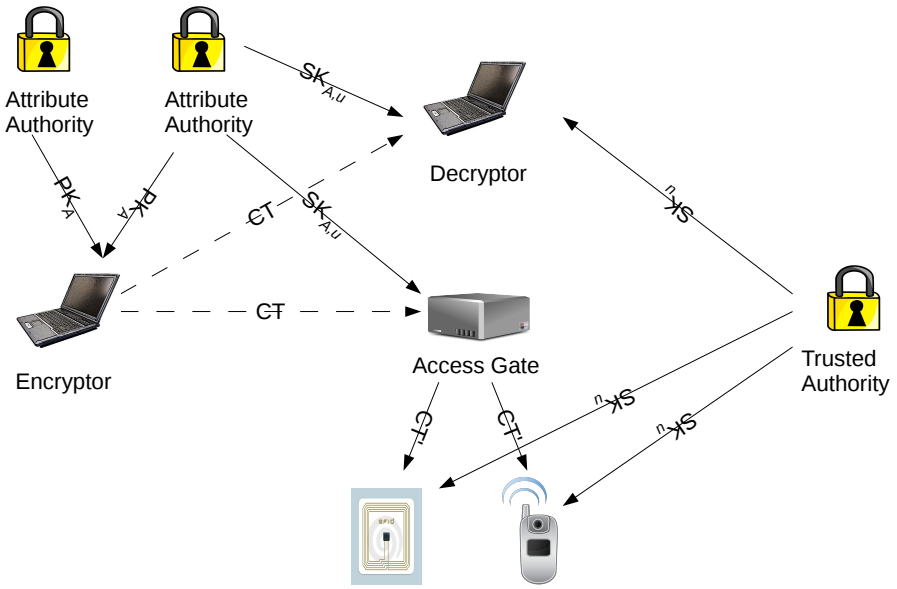


Figure 6.6: Overview of SOA architecture

proposed a modification of [Wat11] that achieves a similar speed up. However, their modification is more expensive, probably in debt of their security proof, while we show how to *directly* apply the speed up without modifying the constructions at all. Also, as explained, unlike [GHW11] our result is more general and applicable to several CP-ABE constructions.)

In most CP-ABE constructions, the message is encrypted with a formula of the following type:

$$C = MY^s,$$

where M is the plaintext message, Y is a publicly known system parameter of the form $Y = e(g_1, g_2)^y$ for generators g_1, g_2 (in the symmetric case, $g_1 = g_2$), some random value y chosen during system setup, and a random value s decided on by the encryptor during the encryption process.

The only way to decrypt the ciphertext and recover M is to find the value of Y^s and compute $M = C/Y^s$. This is done with an elaborate computation process where the decryptor's attribute keys are combined with ciphertext components in

a way specific to the ciphertext policy. The ciphertext components used in the step are not related to the original value M in any way, as they are only used to recover Y^s. To prevent collusions, all attribute keys are blinded with a user specific value, so they are not compatible with attribute keys from a different user, so the actual result is Y^s blinded by this user specific value, too.

This is why the decryptor has another secret key that is used to unblind the result of the computation, which we call the unblinding key. This unblinding step is different for each construction, but usually it requires a single pairing operation which has as input the unblinding key and another ciphertext component. Before this step, the data can be considered to be still encrypted, with the unblinding key being the decryption key. After unblinding, the value of Y^s is obtained, and M can be decrypted.

Note that without the unblinding key, one ends up with encrypted data, so having only the attribute key is not enough to decrypt. On the other hand, the unblinding key is not needed in the first part of the decryption process at all, so anyone who has the attribute keys can compute this part without learning anything about M, while the attribute keys are not needed to compute the second part that requires only a constant number of computation steps using the unblinding key.

This means that the two steps can be done separately by two parties. The first party (the access gate) has access to the user's attribute keys and does the expensive first part of the decryption, ending up with a partially encrypted ciphertext that does not contain information that it can use to recover M. In fact, for some constructions it may not even need to have access to the single ciphertext component C that is dependent on the plaintext message M. The decryptor needs only the unblinding key to be able to complete the decryption.

We will not go into too many cryptographic details here but give an intuition by showing how access gates can be realized for two ABE constructions. For example, in our DABE construction, the complete decryption step is:

$$M = C \cdot \frac{\prod_{i \in I} \left(e(C_i, PK_u) \cdot e(D_i, SK_{\rho(i),u}) \right)^{\omega_i}}{e(C', SK_u)} .$$

Note that only the secret attribute keys and the public user key are used for computing the numerator of the right-hand fraction, while the secret user key is used to compute the denominator. This observation gives us an intuition how to

realize access gates for this construction. If – as explained – the secret attribute keys are given to the access gate, decryption can be split up into two separate steps as follows:

Access Gate: The access gate knows the public user key PK_u as well as the secret attribute keys $SK_{\mathcal{A},u}$ for the same user u, but it has no access to the secret user key SK_u. Using the ciphertext components C_i and $D_i, 1 \leq i \leq \ell$, it computes constants $\{\omega_i | i \in I\}$, such that $\sum_{i \in I} \omega_i \lambda_i = s$ and creates M' as follows:

$$M' = \prod_{i \in I} \left(e(C_i, PK_u) \cdot e(D_i, SK_{\rho(i),u}) \right)^{\omega_i} = e(g, P)^{smk_u} .$$

(For details on this equation we refer to the correctness proof of the respective construction in Section 3.5.1.) Note that the result of this computation is not related to M, as during encryption M was only used in $C = Me(g, Q)^s$, while the ciphertext components C_i and D_i are independent of M. Also note that the only way to compute M from C is by dividing it by $e(g, Q)^s$. However, the access gate cannot recover this value using M' because M' is blinded: Instead of having $e(g, Q)^s$, the access gate has computed $e(g, P)^{smk_u}$. One more value, the secret user key SK_u is needed to complete the decryption.

Decryptor: Using his secret user key $SK_u = \mathrm{MK} \cdot P^{\mathrm{mk}_u}$, the decryptor is able to decrypt the message:

$$M = C \cdot \frac{M'}{e(C', SK_u)}$$

Obviously, the computation time is constant and requires one pairing operation, one inversion in $\mathbb{G}_T$ and two multiplications in $\mathbb{G}_T$.

The same steps can also be applied to the decryption operations of [LOS+10] and [Wat11].

To give another example, [ZH10] proposes a CP-ABE construction with a very interesting property: While the ciphertext policies are restricted to a single AND gate (similar to [CN07] and [NYO09]), the ciphertext size is constant, consisting of only two group elements. In this construction, access gates are especially interesting as each user has one key component for each attribute in the system and all these components are required for each decryption. By storing these key components

remotely on an access gate and only giving the user key component (denoted D_0 in the paper) to the user, both space and time efficiency can be improved. The decryption step again can be split up into to separate steps.

Access Gate: For each attribute of the system, each user has a secret key component D_i. These values are given to the access gate, but the value D (that we can view as the secret user key) is kept secret by the user. With public keys g_i for all i and the ciphertext component C_1, the step performed by the access gate is:

$$M' = \prod_i e(g_i, C_1)/e(C_0, D_i \cdot \prod_{j \in S, j \neq i} g_{K+1-j+i}) .$$

As before, the result M' of this computation is an intermediate result that contains the decryption key but is blinded with a value that the access gate does not know and can not remove using the values it knows. One more value, the secret user key D is needed to complete the decryption.

Decryptor: The decryptor needs the intermediate value M' as well as the secret key component D and the ciphertext component C_0 to recover M by computing

$$M = M' \cdot e(D, C_0) .$$

Again, this computation is very inexpensive, requiring only one pairing operation and one multiplication, so it can be done efficiently even on a resource constrained device in reasonable time.

6.5. Description of System

We now describe each of the parties involved in our system as depicted in Figure 6.6, the services they offer, and briefly discuss our proof-of-concept implementation.

6.5.1. Central Authority

The central authority is the only party that knows the system's master key. As explained before, the public system key PK is supposed to publicly available to all

parties, and it is not explicitly distributed by the central authority. So the only service offered by the central authority is the creation of new user key pairs as discussed in Section 6.4.1:

Service `getUserKeyPair`

Input: (Nothing)
Output: Public and secret user key (PK_u, SK_u)

Create a new user key pair and return it. Both public and secret user keys are single elements of the group $\mathbb{G}$, and can be represented by a string of characters as explained above. The user will later send the encoded public user key to attribute authorities. However, for the decryption functionality he needs to convert the string representation of the secret user key into a cryptographic value. To do this, he needs to acquire the public system parameters that describe the cryptographic groups used in the system.

Note: The response of this service has to be secure against eavesdroppers.

The authenticity of the central authority can easily be verified by testing if the user key pair is valid in the context of the public system key. This is done as follows: Given the public key components P and $e(g, Q)$ and the public user key PK_u compute: $e(g, Q) \cdot e(PK_u, P)$. If the result is equal to $e(g, SK_u)$ for the secret user key SK_u, then (PK_u, SK_u) is a valid key pair.

The test is correct as for any well-formed user key pair it holds that with $PK_u = g^{mk_u}$,

$$e(g, Q) \cdot e(g^{mk_u}, P) = e(g, Q) \cdot e(g, P^{mk_u}) = e(g, Q \cdot P^{mk_u}) = e(g, SK_u)\,.$$

6.5.2. Attribute Authority

Service `getPublicAttributeKey`

Input: Attribute name $\mathcal{A}$
Output: Public attribute key $PK_{\mathcal{A}}$

This service returns a public attribute key for the given attribute. The key is a single element of the group $\mathbb{G}$, and can be represented by a single string of characters as explained above.

Service `getSecretAttributeKey`

Input: Attribute name $\mathcal{A}$, user token u, and public user key PK_u
Output: Secret attribute key $SK_{\mathcal{A},u}$

This service returns a secret attribute key for the given attribute, personalized for the user with the given public user key. The service only returns a valid key if the user (identified by the user token) is eligible of the attribute key. This is discussed in Section 6.4.1.

6.5.3. Access Gates

Service `storeAttribute`

Input: User id u, secret attribute key $SK_{\mathcal{A},u}$
Output: (Nothing)

Stores the secret attribute key, associated with the given user id in the access gate's attribute database. The access gate does not need to verify the secret attribute key. Interestingly, the gate can verify if two secret attribute keys belong to the same user (even if the secret attribute keys are from different authorities): Note that given two secret attribute keys for the same user $SK_{\mathcal{A}_1,u}$ from an authority a_1 and $SK_{\mathcal{A}_2,u}$ from an authority a_2, as well as the respective public attribute keys $PK_{\mathcal{A}_1}$ and $PK_{\mathcal{A}_2}$, it holds that

$$e(PK_{\mathcal{A}_1}, SK_{\mathcal{A}_2,u}) = e(g,g)^{mk_u H_{SKa_1}(\mathcal{A}_1) H_{SKa_2}(\mathcal{A}_2)} = (PK_{\mathcal{A}_2}, SK_{\mathcal{A}_1,u})\,.$$

This equation is not true if the secret attribute keys are associated with different users. This feature can be used as a simple sanity check that may be used to avert the storage of invalid keys.

Service `decryptPartial`

Input: ABE ciphertext $(CT, \mathbb{A})$, user id u
Output: Modified ciphertext M'

Returns a modified ciphertext that can be decrypted by using only a secret user key.

To this end, the access gate applies all secret user keys that it has stored for the given user id to the partial decryption formula (see Section 6.4.2). The result of this is a ciphertext that can be decrypted by the secret user key corresponding to the secret attribute keys.

6.5.4. Implementation

To implement pairing-based algorithms, we use the libpbc [Lyn], which in turn is based on the GNU MP Bignum library[3]. As we also require symmetric cryptography, we use the cryptographic libraries supplied by the OpenSSL Project[4]. The result of all the cryptographic functionality is a library `libdabe`. This library is able to convert between cryptographic keys stored in libpbc format and binary strings, such that the interface does not rely on cryptographic libraries and uses only `char*` elements and the dependencies of `libdabe` are invisible to the parts of the code that implement the actual web services.

All web services were written using the gsoap toolkit[5], where the services were described in a C `.h` header file and then converted into WSDL descriptions by the gsoap compiler as well as C interfaces to use. See Figure 6.7 for an excerpt from the generated WSDL definitions. Specifically, the excerpt exemplarily shows the parts that are relevant for the definition of the service `getPublicAttributeKey` that we already described above. Using these definitions, we implemented server programs for central and attribute authorities as well as a test client that interacted with them.

All services could be realized directly using gsoap without any modifications, using interfaces that conform to the libraries and the standards. Building a complete library from this proof-of-concept implementation is straightforward. This shows that DABE for SOA as developed in this chapter is feasible and can be incorporated smoothly into existing projects using modular software components. The current reliance on three cryptographic libraries in addition to the DABE functions may be mitigated in the future as pairings-based cryptography matures and more complete libraries are developed.

[3] `http://gmplib.org/`
[4] `http://www.openssl.org/`
[5] `http://www.cs.fsu.edu/~engelen/soap.html`

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <definitions name="wsdabe"
3   targetNamespace="http://localhost:8080/wsdabe.wsdl"
4   xmlns:tns="http://localhost:8080/wsdabe.wsdl"
5  [...]
6   xmlns="http://schemas.xmlsoap.org/wsdl/">
7  <types>
8   <schema targetNamespace="http://localhost:8080/wsdabe.wsdl"
9  [...]
10   elementFormDefault="unqualified"
11   attributeFormDefault="unqualified">
12   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
13   <complexType name="getPublicAttributeKeyResponse">
14    <sequence>
15      <element name="name" type="xsd:string" minOccurs="0"
           maxOccurs="1" nillable="true"/>
16      <element name="pka" type="xsd:string" minOccurs="0" maxOccurs
           ="1" nillable="true"/>
17    </sequence>
18   </complexType>
19 [...]
20  </schema>
21 </types>
22 <message name="getPublicAttributeKey">
23  <part name="name" type="xsd:string"/>
24 </message>
25 [...]
26 <portType name="wsdabePortType">
27  <operation name="getPublicAttributeKey">
28   <documentation>Returns public attribute key for the given
        attribute</documentation>
29   <input message="tns:getPublicAttributeKey"/>
30   <output message="tns:getPublicAttributeKeyResponse"/>
31  </operation>
32 [...]
33 </portType>
34 <binding name="wsdabe" type="tns:wsdabePortType">
35  <SOAP:binding style="rpc" transport="http://schemas.xmlsoap.
       org/soap/http"/>
36  <operation name="getPublicAttributeKey">
37   <SOAP:operation style="rpc" soapAction=""/>
38   <input>
39 [...]
40   </input>
41   <output>
42 [...]
43   </output>
44  </operation>
45 </definitions>
```

Figure 6.7: WSDABE WSDL definition (excerpt)

6.6. Conclusion

In this chapter, we examined how DABE can be incorporated into Service-Oriented Architectures. To achieve good performance, we used hybrid encryption, where the documents are encrypted symmetrically with a randomly generated key, and only the key is encrypted with DABE.

While the relevant standards were not designed with data-centric security in mind, we showed how we still can use them to support DABE encrypted documents with only minor modifications.

To manage the large number of cryptographic keys required for DABE, we implemented a number of Web Services. These services could be realized using only standard tools, in addition to several libraries that implement the cryptographic functionality. We showed how the user can even maintain a number of pseudonymous identities, retrieve attributes for each of them and use all of them together for his real identity.

We also extended the DABE framework with a new functionality called *access gates*. Computationally weak decryptors can use such gates to outsource the most expensive parts of the DABE decryption. The gates are secure in an honest-but-curious model.

Finally, we described a proof-of-concept implementation that demonstrates that the ideas proposed in the chapter are applicable to SOA settings and can be implemented with reasonable effort.

Part IV.

Summary

Chapter 7

Conclusion

Data-centric security is a powerful and versatile concept with many possible applications, and it has the potential to vastly improve trust in many scenarios by removing the necessity of costly fine-grained access control and of a trusted party to enforce accesses. With Attribute-Based Encryption, there is now a technology that allows to realize data-centric security in an efficient way.

However, as diversified as the possible applications are, as broad are the features that are expected from an ABE construction. Among the possible ABE features that we discussed in thesis are different formats for the ciphertext policies, single- vs. multi-authority, policy anonymity, and attribute revocation mechanisms. There are also some more exotic features that we have not covered here, such as different kinds of attribute hierarchies [LWG11, LCbHC11], user accountability [LRZW09], conditional proxy re-encryption [ZFZ10] as well as a large number of concepts that use attribute policies to improve other primitives like in ABBE [JK10, ZH10, AHS05] or Worry-free encryption [SS10]. ABE constructions can further be categorized by the underlying security assumptions and models. Thus, when planning to use ABE in any particular setting, there is a large selection of possible constructions, each with its own benefits and drawbacks. To date, the Swiss Army knife of ABE has not yet been discovered, and no known construction supports a majority of wanted features.

In this thesis we approached a number of these challenges, focussing on what we consider relevant in practical scenarios, and we successfully tackled some crucial challenges. We found that when considering practical applications, more

interesting constructions are possible when allowing security proofs in the Generic Group Model instead of restricting oneself to reduction proofs. One of the main contributions of this work was our approach to support the multi-authority case where an arbitrary number of independent attribute authorities is allowed to issue secret attribute keys. In Chapter 3 we described this scenario in detail, defined a scheme named *Distributed Attribute-Based Encryption* and proposed two constructions that implement it. These constructions differ in the format of the input policy as well as the security proof. We found a proof for the second construction that is based on a security reduction, but had to rely on a weaker security model to make it work, while security in the Generic Group Model of both constructions could be proven in a model that is the strongest one used for ABE. This gives further evidence that accepting the Generic Group Model for security proofs allows more versatility and more powerful and efficient constructions.

In Chapter 4 we took a look at privacy issues caused by sending the ciphertext policy in clear. Our new concept of *policy anonymity* allows to approach this problem by obfuscating the policy so that an attacker cannot identify the used policy in a candidate policy set. To this end, we introduced the notion of *Syntax Tree Majors* that allow to construct suitable *policy anonymity sets* for any policy expressible as a monotonic syntax tree. A modification of a know CP-ABE scheme was then used to demonstrate the applicability of policy anonymity using syntax tree majors.

Our examination of ABE in practical settings showed that data-centric security is feasible not only in theory, but also from the technical point of view. While data-centric security has its limits and cannot be used for all real-world access policies (exemplified in a Digital Rights Management scenario), we showed that the classification of access rules into static, i.e., cryptographically enforceable, and dynamic ones is intuitive and that the process of separating the former from the latter ones can be automatized easily. As an example for this we analyzed the Open Digital Rights Language in Chapter 5 and described such an automation process in detail.

Finally, in Chapter 6 we took the leap from theory to practice by extending the standards used for Service Oriented Architectures with ABE technology. We concluded that while not all standards are directly ready to express concepts of data-centric security, modifications that add the relevant parts are possible

without breaking compatibility with conventional access control settings. Our implementation of relevant core services could be done using a well developed and mature SOA framework that is widely used in the web services community.

Throughout this work we have gained an understanding about data-centric security using ABE in practical settings: While the restriction of data-centric security to static access rules prevents it from being a *complete* replacement for conventional access control mechanisms, it can in practically all cases be used to *improve* the enforcement step of access control, allowing to base security assurances on cryptographic methods instead of trust. Using constructions as proposed in this thesis, we have shown ABE to be a useful cryptographic primitive for IT security and ready for implementation in real-world settings.

List of Figures

Bibliography

[AHS05] André Adelsbach, Ulrich Huber, and Ahmad-Reza Sadeghi, *Property-based broadcast encryption for multi-level security policies*, 8th International Conference on Information Security and Cryptology, ICISC 2005 (Dongho Won and Seungjoo Kim, eds.), Lecture Notes in Computer Science, vol. 3935, Springer, 2005, pp. 15–31.

[AI09] Nuttapong Attrapadung and Hideki Imai, *Conjunctive broadcast and attribute-based encryption*, Pairing (Hovav Shacham and Brent Waters, eds.), Lecture Notes in Computer Science, vol. 5671, Springer, 2009, pp. 248–265.

[All08] Open Mobile Alliance, *OMA DRM architecture*, 2008, Version 2.1, `http://www.openmobilealliance.org/Technical/release_program/docs/DRM/V2_1-20081106-A/OMA-AD-DRM-V2_1-20081014-A.pdf`.

[ASKS10] Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov (eds.), *Proceedings of the 17th acm conference on computer and communications security, ccs 2010, chicago, illinois, usa, october 4-8, 2010*, ACM, 2010.

[BB08] Dan Boneh and Xavier Boyen, *Short signatures without random oracles and the SDH assumption in bilinear groups*, J. Cryptology **21** (2008), no. 2, 149–177.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh, *Hierarchical identity*

based encryption with constant size ciphertext, in Cramer [Cra05], pp. 440–456.

[BCHK07] Dan Boneh, Ran Canetti, Shai Halevi, and Jonathan Katz, *Chosen-ciphertext security from identity-based encryption*, SIAM J. Comput. **36** (2007), no. 5, 1301–1328.

[BDF+10] Jean-Luc Beuchat, Hiroshi Doi, Kaoru Fujita, Atsuo Inomata, Piseth Ith, Akira Kanaoka, Masayoshi Katouno, Masahiro Mambo, Eiji Okamoto, Takeshi Okamoto, Takaaki Shiga, Masaaki Shirase, Ryuji Soga, Tsuyoshi Takagi, Ananda Vithanage, and Hiroyasu Yamamoto, *FPGA and ASIC implementations of the η_T pairing in characteristic three*, Computers & Electrical Engineering **36** (2010), no. 1, 73–87.

[Bei96] Amos Beimel, *Secure schemes for secret sharing and key distribution*, Ph.D. thesis, Dept. of Computer Science, Technion, 1996.

[BF03] Dan Boneh and Matthew K. Franklin, *Identity-based encryption from the weil pairing*, SIAM J. Comput. **32** (2003), no. 3, 586–615.

[BGDM+10] Jean-Luc Beuchat, Jorge E. González-Díaz, Shigeo Mitsumari, Eiji Okamoto, Francisco Rodríguez-Henríquez, and Tadanori Teruya, *High-speed software implementation of the optimal ate pairing over barreto-naehrig curves*, 4th International Conference on Pairing-Based Cryptography, Pairing 2010 (Marc Joye, Atsuko Miyaji, and Akira Otsuka, eds.), Lecture Notes in Computer Science, vol. 6487, Springer, 2010, pp. 21–39.

[Bil05] Philip Bille, *A survey on tree edit distance and related problems*, Theor. Comput. Sci. **337** (2005), no. 1-3, 217–239.

[BKP09] Rakeshbabu Bobba, Himanshu Khurana, and Manoj Prabhakaran, *Attribute-sets: A practically motivated enhancement to attribute-based encryption*, 14th European Symposium on Research in Computer Security, ESORICS 2009 (Michael Backes and Peng Ning, eds.), Lecture Notes in Computer Science, vol. 5789, Springer, 2009, pp. 587–604.

[BM05] Walid Bagga and Refik Molva, *Policy-based cryptography and applications*, 9th International Conference on Financial Cryptography and Data Security, FC 2005 (Andrew S. Patrick and Moti Yung, eds.), Lecture Notes in Computer Science, vol. 3570, Springer, 2005, pp. 72–87.

[BM06] Walid Bagga and Refik Molva, *Collusion-free policy-based encryption*, 9th International Conference on Information Security, ISC 2006 (Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel, eds.), Lecture Notes in Computer Science, vol. 4176, Springer, 2006, pp. 233–245.

[BMC06] Walid Bagga, Refik Molva, and Stefano Crosta, *Policy-based encryption schemes from bilinear pairings*, ACM Symposium on InformAtion, Computer, and Communications Security, ASIACCS (Ferng-Ching Lin, Der-Tsai Lee, Bao-Shuh Lin, Shiuhpyng Shieh, and Sushil Jajodia, eds.), ACM, 2006, p. 368.

[Bon07] Dan Boneh, *A brief look at pairings based cryptography*, 48th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2007, IEEE Computer Society, 2007, pp. 19–26.

[Bra10] Erik Brangs, *ODRL-Policies für kryptographische Zugriffskontrolle mit verteilter attributbasierter Verschlüsselung*, Bachelorarbeit, April 2010.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters, *Ciphertext-policy attribute-based encryption*, IEEE Symposium on Security and Privacy, 2007, pp. 321–334.

[BW07] Dan Boneh and Brent Waters, *Conjunctive, subset, and range queries on encrypted data*, 4th IACR Theory of Cryptography Conference, TCC 2007 (Salil P. Vadhan, ed.), Lecture Notes in Computer Science, vol. 4392, Springer, 2007, pp. 535–554.

[CdVFS07] Valentina Ciriani, Sabrina De Capitani di Vimercati, Sara Foresti, and Pierangela Samarati, *k-anonymity*, Secure Data Management in

Decentralized Systems (Ting Yu and Sushil Jajodia, eds.), Advances in Information Security, vol. 33, Springer, 2007, pp. 323–353.

[Cha07] Melissa Chase, *Multi-authority attribute based encryption*, 4th IACR Theory of Cryptography Conference, TCC 2007 (Salil P. Vadhan, ed.), Lecture Notes in Computer Science, vol. 4392, Springer, February 2007, pp. 515–534.

[CN07] Ling Cheung and Calvin C. Newport, *Provably secure ciphertext policy ABE*, ACM Conference on Computer and Communications Security, CCS 2007 (Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, eds.), ACM, 2007, pp. 456–465.

[Cra05] Ronald Cramer (ed.), *Advances in cryptology - eurocrypt 2005, 24th annual international conference on the theory and applications of cryptographic techniques, aarhus, denmark, may 22-26, 2005, proceedings*, Lecture Notes in Computer Science, vol. 3494, Springer, 2005.

[DSP11] Jeong-Min Do, You-Jin Song, and Namje Park, *Attribute based proxy re-encryption for data confidentiality in cloud computing environments*, Computers, Networks, Systems and Industrial Engineering, CNSI (Yung-Cheol Byun, Kiumki Akingbehin, Petr Hnetynka, and Roger Lee, eds.), IEEE, 2011, pp. 248–251.

[dVFJ+07] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati, *Over-encryption: Management of access control evolution on outsourced data*, 33rd International Conference on Very Large Data Bases, VLDB 2007 (Christoph Koch, Johannes Gehrke, Minos N. Garofalakis, Divesh Srivastava, Karl Aberer, Anand Deshpande, Daniela Florescu, Chee Yong Chan, Venkatesh Ganti, Carl-Christian Kanne, Wolfgang Klas, and Erich J. Neuhold, eds.), ACM, 2007, pp. 123–134.

[Erl08] Thomas Erl, *Service-oriented architecture: Concepts, technology, and design*, Prentice Hall, 2008.

[Éti98] Gagnon Étienne, *Sablecc, an object-oriented compiler-framework*, Master's thesis, 1998.

[FLA06] Keith B. Frikken, Jiangtao Li, and Mikhail J. Atallah, *Trust negotiation with hidden credentials, hidden policies, and policy cycles*, Network and Distributed System Security Symposium, NDSS 2006, The Internet Society, 2006.

[FN93] Amos Fiat and Moni Naor, *Broadcast encryption*, 13th Annual International Cryptology Conference, CRYPTO 1993 (Douglas R. Stinson, ed.), Lecture Notes in Computer Science, vol. 773, Springer, 1993, pp. 480–491.

[GGKL89] Morrie Gasser, Andy Goldstein, Charlie Kaufman, and Butler Lampson, *The digital distributed system security architecture*, 12th National Computer Security Conference, vol. 95, 1989.

[GHW11] Matthew Green, Susan Hohenberger, and Brent Waters, *Outsourcing the decryption of ABE ciphertexts*, 20th USENIX Security Symposium, 2011.

[GJPS08] Vipul Goyal, Abhishek Jain, Omkant Pandey, and Amit Sahai, *Bounded ciphertext policy attribute based encryption*, 35th International Colloquium on Automata, Languages and Programming, ICALP 2008 (Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, eds.), Lecture Notes in Computer Science, vol. 5126, Springer, 2008, pp. 579–591.

[Gol06] Dieter Gollmann, *Computer security*, 2nd ed., John Wiley & Sons Ltd, 2006, pp 52ff.

[GPS08] Steven D. Galbraith, Kennth G. Paterson, and Nigel P. Smart, *Pairings for cryptographers*, Discrete Applied Mathematics **156** (2008), no. 16, 3113–3121.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent 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.), ACM, 2006, pp. 89–98.

[IDS02] Takeshi Imamura, Blair Dillaway, and Ed Simons, *XML encryption syntax and processing*, Tech. report, W3C Recommendation, `http://www.w3.org/TR/xmlenc-core/`, 2002.

[Ini02] ODRL Initiative, *Open Digital Rights Language (ODRL) Specification*, August 2002, Version 1.1, `http://odrl.net/1.1/ODRL-11.pdf`.

[IOIO07] Piseth Ith, Yoshihito Oyama, Atsuo Inomata, and Eiji Okamoto, *Implementation of ID-based signature in RFID system*, Asian Pacific Conference on Communications 2007, APCC, IEEE, 2007, pp. 233–236.

[JK10] Pascal Junod and Alexandre Karlov, *An efficient public-key attribute-based broadcast encryption scheme allowing arbitrary access policies*, 10th ACM DRM Workshop, 2010.

[Jou00] Antoine Joux, *A one round protocol for tripartite Diffie-Hellman*, 4th International Symposium on Algorithmic Number Theory, ANTS-IV (Wieb Bosma, ed.), Lecture Notes in Computer Science, vol. 1838, Springer, 2000, pp. 385–394.

[JSGI09] Meiko Jensen, Jörg Schwenk, Nils Gruschka, and Luigi Lo Iacono, *On technical security issues in cloud computing*, IEEE International Conference on Cloud Computing, CLOUD '09, IEEE, 2009, pp. 109–116.

[KC08] Ramaro Kanneganti and Prasad Chodavarapu, *SOA security*, Manning Publications Co., 2008.

[Klu88] Doug Klunder, *Hungarian naming conventions*, Tech. report, Microsoft, `http://www.byteshift.de/msg/hungarian-notation-doug-klunder`, January 1988.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters, *Predicate encryption supporting disjunctions, polynomial equations, and inner products*, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 2008 (Nigel P. Smart, ed.), Lecture Notes in Computer Science, vol. 4965, Springer, 2008, pp. 146–162.

[KTO06] Yuto Kawahara, Tsuyoshi Takagi, and Eiji Okamoto, *Efficient implementation of tate pairing on a mobile phone using java*, International Conference on Computational Intelligence and Security, CIS 2006 (Yuping Wang, Yiu ming Cheung, and Hailin Liu, eds.), Lecture Notes in Computer Science, vol. 4456, Springer, 2006, pp. 396–405.

[KTS07] Apu Kapadia, Patrick P. Tsang, and Sean W. Smith, *Attribute-based publishing with hidden credentials and hidden policies*, 14th Annual Network and Distributed System Security Symposium, NDSS 2007, 2007, pp. 179–192.

[LbHC10] Song Luo, Jian bin Hu, and Zhong Chen, *Implementing attribute-based encryption in web services*, IEEE International Conference on Web Services, ICWS 2010, IEEE Computer Society, 2010, pp. 658–659.

[LC10] Zhen Liu and Zhenfu Cao, *On efficiently transferring the linear secret-sharing scheme matrix in ciphertext-policy attribute-based encryption*, Tech. report, `http://eprint.iacr.org/2010/374.pdf`, 2010.

[LCbHC11] Song Luo, Yu Chen, Jian bin Hu, and Zhong Chen, *New fully secure hierarchical identity-based encryption with constant size ciphertexts*, 7th Information Security Practice and Experience Conference, ISPEC 2011 (Feng Bao and Jian Weng, eds.), Lecture Notes in Computer Science, vol. 6672, Springer, 2011, pp. 55–70.

[LCLS08] Huang Lin, Zhenfu Cao, Xiaohui Liang, and Jun Shao, *Secure threshold multi authority attribute based encryption without a central authority*, 9th International Conference on Cryptology in India, INDOCRYPT 2008 (Dipanwita Roy Chowdhury, Vincent Rijmen, and

Abhijit Das, eds.), Lecture Notes in Computer Science, vol. 5365, Springer, 2008, pp. 426–436.

[LCLX09] Xiaohui Liang, Zhenfu Cao, Huang Lin, and Dongsheng Xing, *Provably secure and efficient bounded ciphertext policy attribute based encryption*, 4th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2009 (Wanqing Li, Willy Susilo, Udaya Kiran Tupakula, Reihaneh Safavi-Naini, and Vijay Varadharajan, eds.), ACM, 2009, pp. 343–352.

[LK07] Kelvin Lawrence and Chris Kaler, *WS SecurityPolicy 1.2*, 2007, Version 1.2, `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html`.

[LOS+10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters, *Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption*, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 2010 (Henri Gilbert, ed.), Lecture Notes in Computer Science, vol. 6110, Springer, 2010, pp. 62–91.

[LRZW09] Jin Li, Kui Ren, Bo Zhu, and Zhiguo Wan, *Privacy-aware attribute-based encryption with user accountability*, 12th International Conference on Information Security, ISC 2009 (Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Agostino Ardagna, eds.), Lecture Notes in Computer Science, vol. 5735, Springer, 2009, pp. 347–362.

[LW11] Allison Lewko and Brent Waters, *Decentralizing attribute-based encryption*, 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 2011 (Kenneth G. Paterson, ed.), Lecture Notes in Computer Science, vol. 6632, Springer, 2011, pp. 568–588.

[LWG11] Jun'e Liu, Zhiguo Wan, and Ming Gu, *Hierarchical attribute-set based encryption for scalable, flexible and fine-grained access control*

in cloud computing, 7th Information Security Practice and Experience Conference, ISPEC 2011 (Feng Bao and Jian Weng, eds.), Lecture Notes in Computer Science, vol. 6672, Springer, 2011, pp. 98–107.

[Lyn] Ben Lynn, *Pbc library*, `http://crypto.stanford.edu/pbc/`.

[Lyn07] Ben Lynn, *On the implementation of pairing-based cryptosystems*, Ph.D. thesis, Stanford University, 2007.

[Mat70] David W. Matula, *On the number of subtrees of a symmetric n-ary tree*, SIAM Journal on Applied Mathematics **18** (1970), no. 3, 668–703.

[MK10] Sascha Müller and Stefan Katzenbeisser, *A new DRM architecture with strong enforcement*, Fifth International Conference on Availability, Reliability and Security, ARES 2010, IEEE Computer Society, 2010, pp. 397–403.

[MK11] Sascha Müller and Stefan Katzenbeisser, *Hiding the policy in cryptographic access control*, Tech. report, `http://eprint.iacr.org/2011/255.pdf`, 2011.

[MKE08] Sascha Müller, Stefan Katzenbeisser, and Claudia Eckert, *Distributed attribute-based encryption*, 11th International Conference on Information Security and Cryptography, ICISC 2008 (Pil Joong Lee and Jung Hee Cheon, eds.), Lecture Notes in Computer Science, vol. 5461, Springer, 2008, pp. 20–36.

[MKE09] Sascha Müller, Stefan Katzenbeisser, and Claudia Eckert, *On multi-authority ciphertext-policy attribute-based encryption*, Bulletin of the Korean Mathematical Society (B-KMS) **46** (2009), no. 4, 803–819.

[Mot07] Motodev, *Introduction of basic concepts in OMA DRM v1.0*, Technical article, Motorola, Inc., July 2007, `http://developer.motorola.com/docstools/articles/OMA_DRM.pdf/`, Retrieved 14/02/11.

[MOV93] Alfred Menezes, Tatsuaki Okamoto, and Scott A. Vanstone, *Reducing elliptic curve logarithms to logarithms in a finite field*, IEEE Transactions on Information Theory **39** (1993), no. 5, 1639–1646.

[NNL01] Dalit Naor, Moni Naor, and Jeffery Lotspiech, *Revocation and tracing schemes for stateless receivers*, 21st Annual International Cryptology, CRYPTO 2001 (Joe Kilian, ed.), Lecture Notes in Computer Science, vol. 2139, Springer, 2001, pp. 41–62.

[NRT99] Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos, *Finding smallest supertrees under minor containment*, 25th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 1999 (Peter Widmayer, Gabriele Neyer, and Stephan Eidenbenz, eds.), Lecture Notes in Computer Science, vol. 1665, Springer, 1999, pp. 303–312.

[NYO09] Takashi Nishide, Kazuki Yoneyama, and Kazuo Ohta, *Attribute-based encryption with partially hidden ciphertext policies*, IEICE Transactions **92-A** (2009), no. 1, 22–32.

[OAS06] OASIS, *WS-security core specification 1.1*, Tech. report, `http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf`, 2006.

[Pas11] Rafael Pass, *Limits of provable security from standard assumptions*, 43rd ACM Symposium on Theory of Computing, STOC 2011 (Lance Fortnow and Salil P. Vadhan, eds.), ACM, 2011, pp. 109–118.

[RB09] Kaspar Riesen and Horst Bunke, *Approximate graph edit distance computation by means of bipartite graph matching*, Image Vision Comput. **27** (2009), no. 7, 950–959.

[RSA11] RSA, *Information-centric security*, `http://www.rsa.com/node.aspx?id=3151`, 2011, retrieved 09/02/11.

[RV06] Francesc Rosselló and Gabriel Valiente, *An algebraic view of the relation between largest common subtrees and smallest common supertrees*, Theoretical Computer Science **362** (2006), 33–53.

[SBC+07] Elaine Shi, John Bethencourt, Hubert T.-H. Chan, Dawn Xiaodong Song, and Adrian Perrig, *Multi-dimensional range query over en-*

crypted data, IEEE Symposium on Security and Privacy, S&P 2007, IEEE, 2007, pp. 350–364.

[SCA06] Michael Scott, Neil Costigan, and Wesam Abdulwahab, *Implementing cryptographic pairings on smartcards*, 8th International Workshop on Cryptographic Hardware and Embedded Systems, CHES 2006 (Louis Goubin and Mitsuru Matsui, eds.), Lecture Notes in Computer Science, vol. 4249, Springer, 2006, pp. 134–147.

[SD02] Andrei Serjantov and George Danezis, *Towards an information theoretic metric for anonymity*, Privacy Enhancing Technologies (Roger Dingledine and Paul F. Syverson, eds.), Lecture Notes in Computer Science, vol. 2482, Springer, 2002, pp. 41–53.

[Sho97] Victor Shoup, *Lower bounds for discrete logarithms and related problems*, 14th Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 1997 (Walter Fumy, ed.), Lecture Notes in Computer Science, vol. 1233, Springer, 1997, pp. 256–266.

[SKSC09] Piotr Szczechowiak, Anton Kargl, Michael Scott, and Martin Collier, *On the application of pairing based cryptography to wireless sensor networks*, Second ACM Conference on Wireless Network Security, WISEC 2009 (David A. Basin, Srdjan Capkun, and Wenke Lee, eds.), ACM, 2009, pp. 1–12.

[SS10] Amit Sahai and Hakan Seyalioglu, *Worry-free encryption: functional encryption with public keys*, 17th ACM Conference on Computer and Communications Security, CCS 2010 (Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, eds.), ACM, 2010, pp. 463–472.

[Sti92] Douglas R. Stinson, *An explication of secret sharing schemes*, Des. Codes Cryptography **2** (1992), no. 4, 357–390.

[SW05] Amit Sahai and Brent Waters, *Fuzzy identity-based encryption*, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 2005 (Ronald Cramer,

ed.), Lecture Notes in Computer Science, vol. 3494, Springer, 2005, pp. 457–473.

[Tho09] Simon Thorpe, *IRM, ERM, EDRM, DRM! What does it all mean?*, Tech. report, Oracle, 2009, Oracle IRM blog, `http://blogs.oracle.com/irm/entry/irm_erm_edrm_drm_what_does_it`, retrieved 20. July 2011.

[Val05] Gabriel Valiente, *Constrained tree inclusion*, J. Discrete Algorithms **3** (2005), no. 2-4, 431–447.

[W3C06] W3C, *Web services policy 1.2 - framework (WS-policy)*, April 2006, `http://www.w3.org/Submission/WS-Policy/`.

[W3C07] W3C, *Simple object access protocol (SOAP) 1.2*, 2007, `http://www.w3.org/TR/soap/`.

[Wat11] Brent Waters, *Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization*, 14th International Conference on Practice and Theory of Public Key Cryptography, PKC 2011 (Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi), Lecture Notes in Computer Science, vol. 6571, Springer, 2011, pp. 53–70.

[YAHK11] Shota Yamada, Nuttapong Attrapadung, Goichiro Hanaoka, and Noboru Kunihiro, *Generic constructions for chosen-ciphertext secure attribute based encryption*, 14th International Conference on Practice and Theory in Public Key Cryptography, PKC 2011 (Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, eds.), Lecture Notes in Computer Science, vol. 6571, Springer, 2011, pp. 71–89.

[YRL08] Shuyeng Yu, Kui Ren, and Wnjing Lou, *Attribute-based content distribution with hidden policy*, 4th workshop on Secure Network Protocols, NPSEC 2008, 2008.

[Yu10] Shucheng Yu, *Data sharing on untrusted storage with attribute-based encryption*, Ph.D. thesis, Worcester Polytechnic Institute, July 2010.

[YWRL10a] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou, *Achieving secure, scalable, and fine-grained data access control in cloud computing*, 29th IEEE International Conference on Computer Communications, INFOCOM 2010, IEEE, 2010, pp. 534–542.

[YWRL10b] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou, *Attribute based data sharing with attribute revocation*, 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2010 (Dengguo Feng, David A. Basin, and Peng Liu, eds.), ACM, 2010, pp. 261–270.

[ZFZ10] Jing Zhao, Dengguo Feng, and Zhenfeng Zhang, *Attribute-based conditional proxy re-encryption with chosen-ciphertext security*, Global Communications Conference, GLOBECOM 2010, IEEE, 2010, pp. 1–6.

[ZH10] Zhibin Zhou and Dijiang Huang, *On efficient ciphertext-policy attribute based encryption and broadcast encryption: extended abstract*, in Al-Shaer et al. [ASKS10], pp. 753–755.

Sascha Müller

Wissenschaftlicher Werdegang

1999 – 2006	Studium der Informatik TU Darmstadt, Darmstadt
2005 – 2006	Prakikant/Werkstudent im Bereich *Smart Devices and Embedded Security* Fraunhofer SIT, Darmstadt
September 2006	Diplomarbeit *Bewertung der Qualität von Unterschriften für die biometrische Erkennung* Fraunhofer SIT, Darmstadt
2006 – 2011	Wissenschaftlicher Mitarbeiter / Doktorand in den Fachgebieten *Sicherheit in der Informationstechnik* und *Security Engineering* TU Darmstadt, Darmstadt