Zeutro LLC

Encryption & Data Security

The OpenABE Design Document

Version 1.0

Contents

Abstract

This document is a software design description for the OpenABE library. It is produced by and is the property of Zeutro, LLC.

Attribute-Based Encryption (ABE) is a next-generation type of public key encryption that allows for flexible policy-based access controls that are cryptographically (that is, mathematically) enforced.

OpenABE is a C/C++ software library that implements several attribute-based encryption schemes together with other core cryptographic functionalities such as authenticated symmetric-key encryption, public key encryption, digital signatures, X.509 certificate handling, key derivation functions, pseudorandom generators and more.

The purpose of the document is two fold. First, the document introduces Attribute-Based Encryption, describes the type of problems ABE addresses (such as those common to today's cloud environments), and details its benefits over prior encryption approaches. Second, the document describes the OpenABE library, which includes support for ABE and other cryptographic tools. The intended use of this document is for understanding ABE and the cryptographic design of the OpenABE library.

Chapter 1

Introduction

In this chapter, we introduce attribute-based encryption (ABE) and place it in the context of prior encryption systems. The following chapters cover how this technology is now available for wide-spread use.

1.1 Background on Prior Encryption Systems

The Need for Encryption. Today sensitive data is often protected by storage servers whose job it is to mediate access. These servers are entrusted to grant access to data to authorized users and withhold it from those not authorized.

While the simplicity of this model is appealing, sole reliance on trusted systems for storage protection is not sensible in today's environment for many reasons:

- First, there is increasing economic pressure to store some data on third party commercial sites. However, it is clearly undesirable to simply store information in the clear on third party servers.
- Second, sensitive data is often stored on devices or machines that are subject to physical compromise. For instance, laptops can be easily lost or stolen. In contexts where network bandwidth is constrained or simply unavailable these devices will need to store sensitive data until there is an opportune time to offload it. If a device is captured, sensitive information could be compromised.
- Finally even if data is stored in a physically secure location, it is still possible that it will be compromised. The number of targeted attacks on government networks and businesses has increased dramatically in recent years. A general assumption of information technology experts is that it is reasonable to assume that an organization's network will be penetrated at some point by unauthorized parties.

Given the above problems, it is important that data access be protected by encryption. Roughly, encryption provides a method of encoding data such that it can only be understood with access to a proper secret key that allows a user (or device) to decrypt the data. Without this secret key an attacker will not be able to learn the original contents of the encrypted data. In this manner, sensitive information can be kept private *even if the underlying storage or storage service is compromised.*

The Limitations of Prior Encryption Systems. Private-key encryption (also called symmetric-key encryption) allows two users with a pre-shared secret to securely encrypt and decrypt data. A prominent example of a private-key encryption scheme in use today is the Advanced Encryption Standard (AES) [10]. While private-key schemes often offer very good efficiency, they are insufficient for many applications. The primary drawback with these systems is that both users must have this shared secret before they can securely communicate, which can be impractical in many online settings today. Exchanging information securely without *a prior* shared secret is the main challenge addressed by public-key encryption.

In public-key encryption (also called asymmetric-key encryption) systems, encrypted data is targeted for decryption by a single *known* user. The user wishing to receive encrypted data generates a public and private key pair. She publishes the public key, which can be used by anyone to encrypt to her, and keeps secret the private key, which she can use for decryption. A prominent example of a public-key encryption scheme in use today is RSA [17] (which is named after its inventors: Rivest, Shamir and Adleman). While this functionality is useful for applications such as encrypted email and establishing secure web sessions, it lacks the expressiveness needed for more advanced data sharing.

For instance, suppose one wants to share data based on the credentials or roles of users in the system. Consider a user that wants to encrypt data to the policy of ("Engineer" **AND** "Level 5 Access") **OR** "Auditing Division". Using traditional encryption techniques, that user would then need to lookup, enumerate, and then encrypt to every single user that matches this policy. (In practice today, we might use RSA to encrypt an AES key to each user and then encrypt the data once under this AES key.) For many polices this is a daunting task considering that this list may be long or not available to the data owner. In addition, personnel change roles and responsibilities frequently. When trying to share information according to a policy using traditional public key encryption, we run up against multiple obstacles.

- Supplying a database that enumerates all users that match a policy is a complex task. The database must be continually updated and a user that wishes to encrypt data must maintain a connection.
- If several users match a policy, then we must encrypt the data to each one of them individually. Both the encryption time and ciphertext size (and resulting storage or bandwidth costs) will grow linearly in the number of users that match the policy even if the policy itself is small.
- This method does not work well under circumstances where different people step into new roles. Suppose that data is encrypted to all users that match a certain policy and then stored. What happens when a user later gains a credential that should allow him to access the data?
- Finally, the information about which users have certain credentials itself is often restricted. A soldier that is tasked with encrypting information for a Special Forces unit in Afghanistan should not necessarily be able to enumerate all other soldiers that match this policy.

ABE is a more powerful form of public-key encryption that can overcome these obstacles.

1.2 An Introduction to Attribute-Based Encryption

Over the last decade, a new vision for encrypting data emerged, called Attribute-Based Encryption (ABE). Instead of encrypting to individual users, in an attribute-based encryption system, one can embed *any* access control policy into the ciphertext itself. Thus, the access controls are now enforced by the hard math of cryptography and no longer by reliance on a trusted server.

In an ABE system, an attribute can be a string or number, such as "female", "senior", "CS256", "2014". An access control policy can be any boolean formula over these attributes, such as "2014" **AND** "female" **AND** ("CS128" **OR** "CS256") **AND** ("freshman" **OR** "sophomore"). (We discuss policies of complexity beyond boolean formulas at the end of this subsection.)

In a basic ABE system, there is one authority that publishes some public parameters (used for encryption) and distributes private keys to users (for decryption). Anyone with the public parameters can encrypt their data in such a way that only users with the proper permissions can recover the data.

There are three main types of ABE.

1. **Role-Based Access Controls: Ciphertext-Policy ABE.** In a CP-ABE system, attributes are associated with users and policies are associated with ciphertexts. A user can decrypt a ciphertext if and only if her attributes satisfy the policy.

   Example: Alice has attributes "female", "nurse", "floor 3", "respiratory specialist". Bob encrypts a patient's medical file for staff members matching the policy ("doctor" **OR** "nurse") **AND** ("floor 3" **OR** "floor 4"). Alice is able to open this patient's file.

This system works well when the encryption policies are known beforehand, e.g., technical reports, HR documents, medical records.

2. **Content-Based Access Controls: Key-Policy ABE.** In a KP-ABE system, policies are associated with users (that is, their private keys) and attributes are associated with ciphertexts. A user can decrypt a ciphertext if and only if its attributes satisfy her (private key's) policy.

   Example: Company Y maintains a record of its emails, where each email is encrypted with a set of attributes associated with the meta-data of that email, e.g., to, from, subject, date, IP address, etc. Later, it is discovered that sensitive information was leaked by employee Edward during the first two weeks of March 2014. Company Y can generate a key that allows an auditor to open any emails matching the policy (From: Edward AND (Dated: March 1-14, 2014)). This grants an auditor access to the slice of information he needs without allowing him to view the entire email database.

   This system is well-suited for granting access to slices of data, e.g., releasing information to an analyst or auditor, responding to a subpoena, searching in email, cloud or other big data applications.

3. **Multi-Organization Role-Based Access Controls: Multiauthority ABE.** An MA-ABE system is a CP-ABE system that can support multiple independent authorities. That is, multiple different authorities can grant users private keys based on their own set of attributes and anyone can encrypt data using policies over attributes from multiple authorities.

   Example: Bob has attributes "male", "birthdate: 10/14/1960" certified by his US passport, attributes "Senior Scientist", "Product Z Group" certified by his employer Capstone, etc. Carol can generate an encryption policy such as ("DoJ: Secret Clearance" **OR** "DoD: Secret Clearance") **AND** ("Capstone: Product Z Group" **AND** "US Passport: over the age of 35").

   The ability to support multiple authorities allows for localized control within a large organization (e.g., each division of an organization can certify its own attributes) and makes it easier to share information across trust boundaries. The MA-ABE functionality is patented by Zeutro (US Patent No. 8880875). The MA-ABE functionality is not included in OpenABE, but is available as part of a more comprehensive library called Zeutro's Toolkit (ZTK) which is available for commercial license.

Please see Chapter 2 for a precise description of how we implement these ABE systems in OpenABE.

Collusion Resistance. A cornerstone of all these systems is the idea of security against *collusion attacks* or against attackers that gain access to multiple keys. The guarantee wanted is that a group of colluding users cannot gain access to more data than the union of what they could each access individually. That is, they cannot escalate their privileges to recover data that none of them were authorized to see.

Consider an example of sharing information to anyone that matches the policy ("Navy Officer" **AND** "PACOM") **OR** "TRANSCOM". Suppose an attacker compromises two different users: Naval Officer Smith with a key that has the credential for "Navy Officer" and Army Private Jones with a different key that has the credential for "PACOM". Then a secure system should deny the attacker the ability to decrypt the encrypted data. Building systems secure against this attack is the core technical challenge in building secure ABE systems. This is typically what distinguishes ABE, as a technology, from systems "engineered" from traditional public or symmetric key encryption that are claiming similar features.

Background on Attribute-Based and Functional Encryption. In 2005, Dr. Brent Waters (Zeutro's Chief Scientist) and Prof. Amit Sahai laid the conceptual foundations for attribute-based encryption and presented the first (limited) realization of an ABE system [18]. Since then, over 100 peer-reviewed publications have appeared on ABE representing significant progress on the underlying cryptographic capabilities including: functionality, efficiency, security analysis, and decentralization of trust.

In the initial release of OpenABE, we focus on ABE systems where the access control policies are boolean formulas, because we believe this offers the best balance of efficiency and practicality for most applications.

However, the access control policy could be a regular expression (Zeutro US Patent No. 8566601). Or it could be any policy that can be represented by a circuit, at the cost of some efficiency.

Over time, the concept of ABE broadened into the notion of *Functional Encryption*, which we now define to clarify its relation to ABE. In an ABE system, the entire message in a ciphertext can either be recovered or not based on the access control policy. However, in a functional encryption system, a different function of the message may be returned based on the user's access permissions. For instance, if the encrypted message is a photograph, a user with a high clearance level might be able to view the entire image, while a user with a lower clearance level might only be able to view a partially redacted image. While there exist candidate constructions allowing any type of function for a functional encryption scheme, these systems are not yet efficient enough for practice. The reader is referred to [9] for an overview of functional encryption.

To summarize the state of modern public key encryption, we include Figure 1.2.

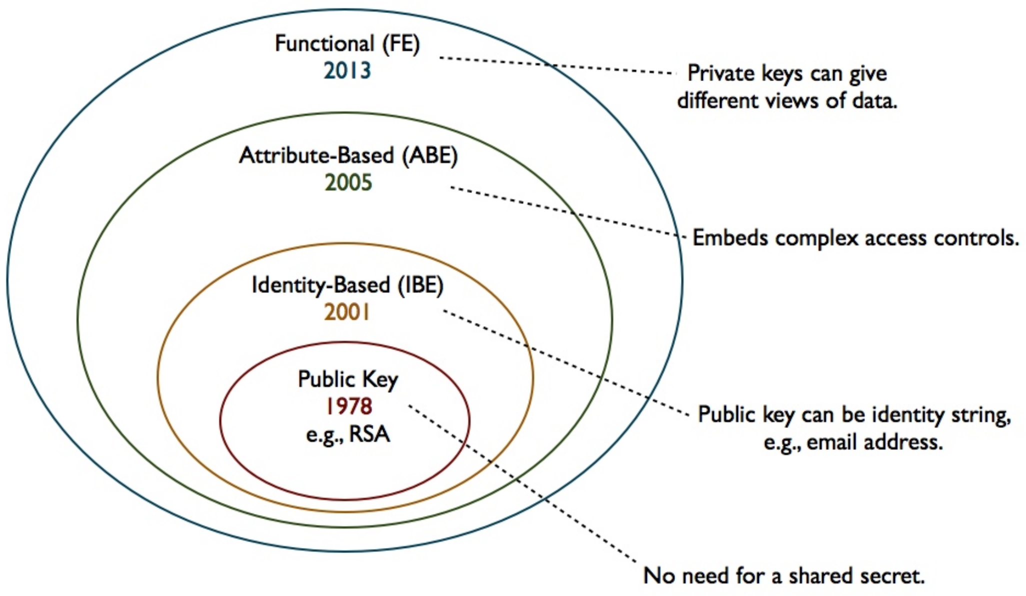
Functional (FE)
2013
Private keys can give different views of data.
Attribute-Based (ABE)
2005
Embeds complex access controls.
Identity-Based (IBE)
2001
Public key can be identity string, e.g., email address.
Public Key
1978
e.g., RSA
No need for a shared secret.

Figure 1.1: A summary of the evolution of public key encryption.

1.3 The Purpose of OpenABE

OpenABE library. OpenABE is a cryptographic library that incorporates state-of-the-art cryptographic algorithms, industry standard cryptographic functions and tools, and an intuitive application programming interface (API). OpenABE is intended to allow developers to seamlessly incorporate ABE technology into applications that would benefit from ABE to protect and control access to sensitive data. OpenABE is designed to be easy to use and does not require developers to be encryption experts.

The purpose of OpenABE is to make the power of ABE widely available in an easy-to-use cryptographic library. We believe this technology can revolutionize the data security industry.

There were a number of notable obstacles in making ABE ready for practical deployment. The underlying cryptography of ABE needed to be advanced to simultaneously provide security against tampering attacks, sharing across trust domains, efficiency, revocation and more. Zeutro's team of cryptographers put together a suite of flagship ABE systems and solutions, resulting in multiple US patents. Some of these features are included in OpenABE and others are available with a commercial license.

Chapter 2

OpenABE library

2.1 Overview

The OpenABE library is a C++ library that implements several attribute-based encryption (ABE) schemes[1] that can be used across a variety of applications. The goal of this library is to develop an efficient implementation that embeds security guarantees at the architectural level. OpenABE is modular in the sense that we can swap one cryptographic scheme for another without updating application logic, comprehensive in that it supports routines necessary to perform common cryptographic tasks, and extensible in that we can support several additional advanced encryption scheme types with relatively little effort. Furthermore, the OpenABE incorporates best practices in encryption scheme design which includes security against chosen ciphertext attacks, a simple interface for transporting symmetric keys, and support for performing encryption of large data objects.

The rest of this chapter is organized as follows: we first describe high-level features of the library in Section 2.2, then the remaining sections will elaborate on the components of the OpenABE architecture shown in Figure 2.1.

2.2 High-Level Features

Generalized Application Programming Interface (API) for public-key cryptography. A challenge in developing an encryption library that supports both traditional public-key and advanced cryptography such as Attribute-Based Encryption (ABE) and Functional Encryption (FE) is the differences in their interfaces and the preponderance of schemes in the literature. For example, attribute-based encryption schemes come in many different flavors, including Ciphertext-policy and Key-policy. Within each flavor of ABE, there may be several alternative schemes, each of which offers different tradeoffs of features, performance and security.

In previous academic efforts, this has resulted in multiple distinct (and incompatible) implementations. Since this is an area that is advancing quickly, we chose to implement a single comprehensive library capable of supporting a number of schemes (including non-FE types). See Section 2.3 for details.

To accomplish this, OpenABE provides a generalized API that simplifies the encryption process by identifying common elements that are similar across many different schemes. This results in a single interface for an arbitrary number of schemes, and makes it possible to transition an application from one scheme to an alternative scheme with almost no code changes. The OpenABE API provides different options only where underlying scheme types differ − *e.g.*, the input to certain algorithms − in a way that is intuitive.

[1]An encryption scheme consists of a set of algorithms that enable two parties to communicate secretly in the presence of an eavesdropper who can monitor all communication between them.

Modular Underlying Math Library. A limitation of previous advanced encryption prototypes is that they are highly dependent on a single underlying mathematics library, which is required to perform critical operations such as elliptic curve operations and bilinear maps (or pairings). This makes it quite challenging to update the code to take advantage of advances in the technology, such as the performance improvements offered by more recent math libraries such as RELIC.

Our goal in designing the library was to provide a neutral mathematics API to support base elliptic curve and bilinear operations. As a result, our API can be connected to and implemented via one or more specific math libraries. The advantage of this approach is that the centralized math API makes it relatively easy to update the library to take advantage of new libraries, or even to deploy multiple versions of OpenABE that take advantage of specific strengths in different implementations.

We refer to this underlying mathematics API as the Zeutro Math Library (ZML) (shown at the bottom of Figure 2.1) and we use this abstraction throughout OpenABE. The current implementation uses one of the fastest pairing implementations available in the RELIC library [1], giving us dramatic performance improvements over previous efforts.

Protection against Ciphertext Tampering Attacks. There are different, well-established categories of attacks that encryption systems should withstand. Generally, the most basic type of security offered is called *security against chosen plaintext attacks* (CPA-secure), where an adversary is allowed to see encrypted messages of his choice. Real world implementations demand a higher level of security called *security against chosen ciphertext attacks* (CCA-secure), where an adversary is allowed to tamper with ciphertexts and observe the behavior of the system afterwards. In many cryptosystems, one can learn the plaintext by just observing whether a client throws an error or not. A famous example of this is Bleichenbacher's [8] demonstration of such a weakness in the RSA PKCS#1 encryption standard. Therefore, CCA security is generally regarded as necessary for practical deployments.

OpenABE uses CCA-secure implementations of all ABE (and non-ABE) encryption systems, and this security level is turned on by default. To do this for ABE, we developed a general approach for transforming any CPA-secure scheme into a CCA-secure scheme. The details of this are described in Section 2.3.8.

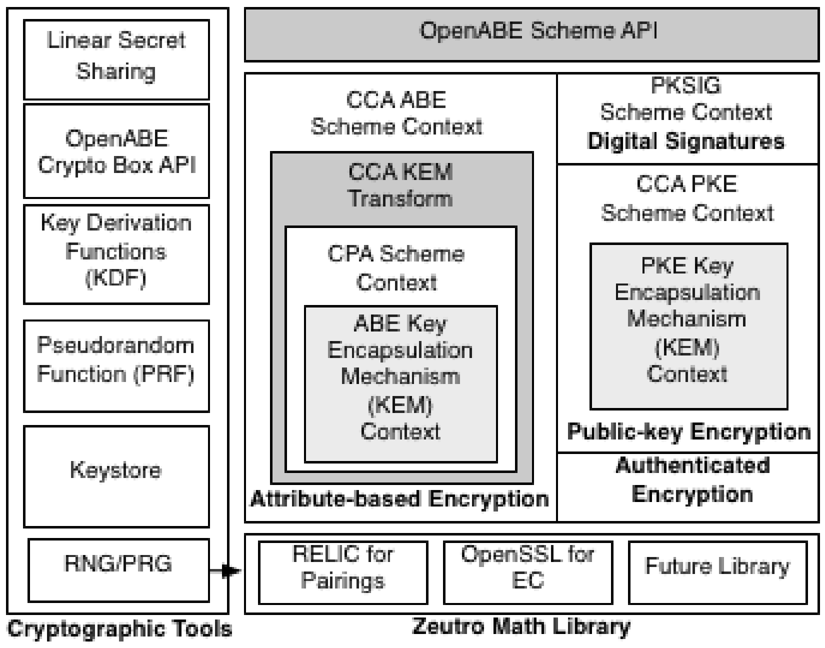

Figure 2.1: OpenABE library architecture.

2.3 Core Cryptographic Algorithms

This section describes the core cryptographic algorithms we have implemented. Specifically, OpenABE provides the following:

- support for multiple types of attribute-based encryption (ABE) Key Encapsulation (KEM) schemes including for Key-Policy ABE and Ciphertext-Policy ABE. We provide chosen-ciphertext security for each ABE KEM scheme type as well.
- support for public-key encryption with chosen-ciphertext security, digital signatures, and authenticated symmetric-key encryption.
- support for several common cryptographic tools including a linear secret sharing scheme (LSSS), key derivation functions (KDF), and pseudo-random functions (PRF). We also include support for a random number generator (RNG) that can be modularly swapped with a pseudo-random generator (PRG).

We first provide background on some terms we will use throughout the rest of this section, then we will describe each cryptographic algorithm and the implementation choices.

2.3.1 Algebraic Setting

Bilinear Groups. Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be groups of prime order p. A map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is an admissible bilinear map (or pairing) if it satisfies the following three properties:

1. *Bilinearity*: for all $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_p$, it holds that $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.
2. *Non-degeneracy*: if g_1 and g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, then $e(g_1, g_2)$ is a generator of $\mathbb{G}_T$.
3. *Efficiency*: there exists an efficiently computable function that given any $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, computes $e(g_1, g_2)$.

An admissible bilinear map generator BSetup is an algorithm that on input a security parameter 1^τ, outputs the parameters for a bilinear group $(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ such that p is a prime in $\Theta(2^\tau)$, $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are groups of order p where g_1 generates $\mathbb{G}_1$, g_2 generates $\mathbb{G}_2$ and $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is an admissible bilinear map. The above bilinear map is called *asymmetric* and the implementations use this highly efficient setting. We describe the curve choices for OpenABE later on in Section 2.5.

2.3.2 Attributes

Attributes are arbitrary binary strings and are not restricted or fixed at system initialization time. We stress that your set of attributes can be anything you want and are likely specific to your application.

We illustrate here our specific approach for representing attributes. In the OpenABE, attributes are in two parts: "$\{type\} : \{value\}$." A *type* part that indicates the type of attribute being represented and a *value* part that represents the actual attribute string. This *type* field serves as a way of capturing namespaces or domains for attributes. For example, "$system : attribute1$" could indicate that a specific $attribute1$ is a system-only type attribute.

For certain features, it is desirable to represent numbers as attributes. In OpenABE, we follow a similar approach to Bethencourt *et al.* [7] for extending attributes to support numerical values. Specifically, to represent the numerical attribute "$a = k$" for some n-bit integer k, we convert the comparison into a *binary* representation which produces n (non-numerical) attributes which specify the value of each bit in k (up to 32-bits). As an optimization, we support specifying how many bits are actually used to represent the value k with a simple encoding such as "$a = k\#b$" where b is the number of bits to use for representing k.[2]

[2]Note that we do a sanity check to make sure that all of k can be represented with b bits.

2.3.3 Access Control Structures

This section defines the notion of an access control structure (or access structure for short) that we will use throughout the description of ABE.

Definition 2.3.1 Access Structure. *Let $\{P_1, P_2, \ldots, P_n\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^{\{P_1,P_2,\ldots,P_n\}}$ is monotone if $\forall B, C$: if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$. An access structure (respectively, monotone access structure) is a collection (respectively, monotone collection) $\mathbb{A}$ of non-empty subsets of $\{P_1, P_2, \ldots, P_n\}$,* i.e., *$\mathbb{A} \subseteq 2^{\{P_1,P_2,\ldots,P_n\}} \backslash \{\emptyset\}$. The sets in $\mathbb{A}$ are called authorized sets, and the sets* **not** *in $\mathbb{A}$ are called the unauthorized sets.*

In the OpenABE, access structures are boolean formulas represented as trees.[3] Each non-leaf node of the tree represents a threshold gate, described by its children and a threshold value. If num_x is the number of children of a node x and k_x is its threshold value, then $0 < k_x \leq num_x$. When $k_x = 1$, the threshold gate is an OR gate and when $k_x = num_x$, it is an AND gate. Each leaf node x of the tree is described by an attribute and a threshold value $k_x = 1$.

We now define some terms over trees that we use in the ABE scheme descriptions. We denote the parent of the node x in the tree by $\mathbf{parent}(x)$. The function $\mathbf{att}(x)$ is defined only if x is a leaf node and denotes the attribute associated with the leaf node x in the tree. The access tree $\mathcal{T}$ also defines an ordering between the children of every node. That is, the children of a node are numbered from 1 to num. The $\mathbf{index}(x)$ returns such a number associated with the node x. Note that the index values are uniquely assigned to nodes in the access structure as the tree is being constructed.

As described in Section 2.3.2, we represent numerical attributes as a "bag of bits" representation [7]. To support integer comparisons (*e.g.*, $<, \leq, >, \geq, =$), we then use AND and OR gates over the non-numerical attributes representation. For comparisons such as "$a < k$", there is a direct relation between the bits of the constant k and the choice of gates. The same applies to other operators $\leq, >, \geq, =$ with at most n gates or possibly fewer gates depending on the constant k. Note that this technique enables implementing revocation of ABE decryption keys by expressing dates in the access structure. See Section 2.3.10.

Satisfying an Access Structure. Let $\mathcal{T}$ be a tree with root r. We denote by $\mathcal{T}_x$ that the subtree of $\mathcal{T}$ is rooted at the node x. Thus, $\mathcal{T}$ and $\mathcal{T}_r$ are equivalent. If a set of attributes γ satisfies the access structure $\mathcal{T}_x$, this will be denoted as $\mathcal{T}_x(\gamma) = 1$. We implement a recursive algorithm called **ScanTree** in OpenABE that computes $\mathcal{T}_x(\gamma)$ as follows:

- If x is a non-leaf node, evaluate $\mathcal{T}_{x'}(\gamma)$ for all children x' of node x. $\mathcal{T}_x(\gamma)$ returns 1 if and only if at least k_x children also return 1.
- If x is a leaf node, then $\mathcal{T}_x(\gamma)$ returns 1 if and only if $\mathbf{att}(x) \in \gamma$.
- In the implementation, the algorithm also returns a minimum subset of attributes in γ needed to satisfy $\mathcal{T}$.

2.3.4 Secret Sharing

ABE constructions typically makes use of secret sharing schemes as a core building block, so we describe it in this section. Because OpenABE represents access structures in terms of a tree representation, we slightly adapt the usual secret sharing definition [6] to this specialized data structure. The secret sharing scheme consists of two algorithms:

$\mathbf{ComputeSecretShare}(s, \mathcal{T}) \rightarrow \lambda$. This algorithm takes as input a secret $s \in \mathbb{Z}_p$, an access structure $\mathcal{T}$ and computes the secret shares of s according to the structure of $\mathcal{T}_r$. First, choose a polynomial q_x for each non-leaf node x in the access structure. This is done recursively starting from root node r.

[3] In the academic literature, the secret sharing definition is usually described over access structures represented as a matrix [6]. Similarly, access structures could be extended to represent regular languages (*e.g.*, regular expressions).

For each node x in the tree, set the degree d_x of the polynomial q_x to be one less than the threshold value k_x of that node (*i.e.*, $d_x = k_x - 1$). For root node, set $q_r(0) = s$ and d_r other points of the polynomial q_r randomly to define it completely. For any other node x, set $q_x(0) = q_{parent(x)}(\textbf{index}(x))$ and choose d_x other points randomly to completely define q_x.

With the polynomial q defined as above, we set $\lambda_x = q_x(0)$ for reach leaf node x in $\mathcal{T}$. Thus, let $\{\lambda_x\}_\ell$ be the set of secret shares of s where ℓ is the number of leaf nodes in $\mathcal{T}$. The algorithm outputs λ.

It has been proven in the literature [6] that every secret sharing scheme has a reconstruction property. We define a reconstruction algorithm that follows the structure of access structure $\mathcal{T}$.

RecoverCoefficient$(\mathcal{T}, \gamma) \rightarrow (\omega, \mathrm{T})$. This algorithm takes as input an access structure $\mathcal{T}$ and a set of attributes γ. If the set of attributes γ satisfies the access structure $\mathcal{T}$ (*i.e.*, $\mathcal{T}(\gamma) = 1$), then we proceed as follows. We define the Lagrange coefficients $\triangle_{i,S}(x)$ for $i \in \mathbb{Z}_p$ and a set S of elements in $\mathbb{Z}_p$:

$$\triangle_{i,S}(x) = \prod_{j \in S, j \neq i} \frac{x - j}{i - j}$$

We first run the **ScanTree** algorithm to pick a subset $\mathrm{T} \subseteq \gamma$ that satisfies the access structure $\mathcal{T}$. Then, for each node x, we set $\omega_x = \omega_{parent(x)} \cdot \triangle_{i,S_x}(0)$. We need only return the coefficients ω_x associated with each leaf node $x \in \mathrm{T}$. The algorithm outputs the Lagrange coefficients ω and the set of attributes T.

2.3.5 Key-Policy Attribute-based Encryption

This section describes a Key-Policy ABE Key Encapsulation scheme for a variant of the Goyal-Pandey-Sahai-Waters Large Universe system [11, Section 5], [15, 16]. The KEM variant also incorporates optimizations in [15, Section 2.2.3]. The construction consists of four algorithms. The setup algorithm generates the public parameters and the master secret key for a single authority. The authority can run keygen to generate a user's private key with an access structure that grants them fine-grained access. The encryption algorithm takes as input a descriptive set of attributes and outputs a symmetric key and ciphertext. The decryption algorithm is used by an authorized user to decrypt a ciphertext.

Setup$(\tau, n) \rightarrow \mathrm{PK}, \mathrm{MSK}$. The setup algorithm takes as input a security parameter τ and runs $\mathsf{BSetup}(1^\tau) \rightarrow (p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, chooses a random exponent $y \in \mathbb{Z}_p$, and computes $Y = e(g_1, g_2)^y$. In addition, we will use as a collision-resistant hash function $H_1 : \mathbb{G}_T \rightarrow \{0,1\}^n$ that we model as a random oracle.[4] The algorithm outputs the public parameters PK and the master secret key MSK as follows:

$$\mathrm{PK} = \{g_1, g_2, e(g_1, g_2)^y\}, \qquad \mathrm{MSK} = y$$

Keygen$(\mathrm{PK}, \mathrm{MSK}, \mathcal{T}) \rightarrow \mathrm{SK}$. Let $T : \{0,1\}^* \rightarrow \mathbb{G}_1$ be a function that we will model as a random oracle.[5]

The key generation algorithm outputs a private key which enables the user to decrypt a message encrypted under a set of attributes γ, if and only if $\mathcal{T}(\gamma) = 1$. The algorithm calculates the randomized shares of y according to the access structure $\mathcal{T}$ (*e.g.*, $\mathsf{computeSecretShares}(y, \mathcal{T}) \rightarrow \lambda$). The following secret values are handed to the user for each leaf node x in the tree:

$$D_x = g_1^{\lambda_x} \cdot T(i)^{r_x} \text{ where } i = \textbf{att}(x) \qquad d_x = {g_2}^{r_x}$$

Encrypt$_{\mathbf{KEM}}(\mathrm{PK}, \gamma; u) \rightarrow (\mathsf{Key}, \mathrm{CT})$. The encryption algorithm takes as input the public parameters PK, a set of attributes γ, and an optional input seed $u \in \{0,1\}^k$ to a pseudo-random generator G (see Section 2.4.2 for implementation details). The algorithm first chooses a random $s \in \mathbb{Z}_p$ (if seed u is specified, then use G as source of randomness) and then returns the following:

$$\mathsf{Key} \leftarrow H_1(e(g_1, g_2)^{ys}), \mathrm{CT} = (\gamma, E'' = g_2^s, \{E_i = T(i)^s\}_{i \in \gamma})$$

[4] H_1 is a keyed-hash function instantiated using SHA-256.

[5] Random oracle T is a keyed-hash function instantiated using SHA-256. See Section 2.2.2 in [15]

Decrypt$_{\mathbf{KEM}}$ $(\mathrm{CT}, \mathrm{SK}) = \mathsf{Key}$. The decryption algorithm takes as input the ciphertext CT and the user's private key SK which embeds the access structure $\mathcal{T}$. The algorithm first determines whether the access structure $\mathcal{T}$ satisfies the attributes γ on the ciphertext (*e.g.*, if $\mathcal{T}(\gamma) = 1$). If so, then we proceed to recover the Lagrange coefficients ω for the minimum set of attributes S necessary to satisfy $\mathcal{T}$ (*e.g.*, $\mathsf{recoverCoefficients}(\mathcal{T}, \gamma) \rightarrow (\omega, S)$). Therefore, for each such attribute $i \in S$, the corresponding coefficient ω_i and the corresponding SK components in $\mathcal{T}$ (*e.g.*, defined by $D_{\rho(i)}$ and $d_{\rho(i)}$),[6] the algorithm first computes:

$$\prod_{i \in S} \left(\frac{e(D_{\rho(i)}, E'')}{e(E_i, d_{\rho(i)})} \right)^{\omega_i} =$$

$$\prod_{i \in S} \frac{e(g_1, g_2)^{\lambda_x \omega_i s} \cdot e(T(i), g_2)^{\omega_i r_x s}}{e(T(i), g_2)^{\omega_i r_x s}} = \prod_{i \in S} e(g_1, g_2)^{\lambda_x \omega_i s} = e(g_1, g_2)^{ys}$$

The algorithm can then compute $\mathsf{Key} = H_1(e(g_1, g_2)^{ys})$.

Implementation Details

- *Optimizations.* Let $k = |S|$. We can reduce the number of pairings from $2k$ to $k + 1$ by bringing the Lagrange coefficients inside the pairing at the expense of increasing the number of exponentiations from k in $\mathbb{G}_T$ to $2k$ in $\mathbb{G}_1$. Because pairings operations are more computationally intensive than exponentiation, this optimization increases the overall speed of decryption. The resulting decryption in OpenABE:

$$\frac{e(\prod_{i \in S} D_{\rho(i)}{}^{\omega_i}, E'')}{\prod_{i \in S} e(E_i{}^{\omega_i}, d_{\rho(i)})}$$

- *CCA-Security.* Recall that the above scheme is only CPA-secure, so the actual implementation uses the CCA-secure transformation via Section 2.3.8.

2.3.6 Ciphertext-Policy Attribute-based Encryption

This section describes a Ciphertext-Policy ABE KEM scheme variant of Waters Large Universe system [20, Appendix A]. The construction consists of four algorithms. The setup algorithm generates the public parameters and the master secret key for a single authority. An authority can run keygen to generate a private key for a particular user that grants them a set of attributes. The encryption algorithm takes as input an access structure and outputs a symmetric key and ciphertext. The decryption algorithm is used by an authorized user to decrypt a ciphertext.

Setup$(\tau, n) \rightarrow \mathrm{PK}, \mathrm{MSK}$. The setup algorithm takes as input a security parameter τ and runs $\mathsf{BSetup}(1^\tau) \rightarrow (q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, and chooses random exponents $\alpha, a \in \mathbb{Z}_p$. In addition, we will use as a collision-resistant hash function $H_1 : \mathbb{G}_T \rightarrow \{0,1\}^n$ and $H_2 : \{0,1\}^* \rightarrow \mathbb{G}_1$ (both will be modeled as random oracles).[7] The authority publishes the public parameters PK and sets master secret key MSK as follows:

$$\mathrm{PK} = \{g_1, g_2, g_1^a, e(g_1, g_2)^\alpha\} \quad \mathrm{MSK} = \{\alpha, g_2^a\}$$

Keygen$(\mathrm{PK}, \mathrm{MSK}, \gamma) \rightarrow \mathrm{SK}$. The key generation algorithm takes as input the master secret key and a set of attributes γ. The algorithm first chooses a random $t \in \mathbb{Z}_p$. It creates the private key SK as follows:

$$K = g_2^\alpha \cdot g_2^{a \cdot t} \qquad L = g_2^t \qquad \forall x \in S \ K_x = H_2(x)^t$$

Encrypt$_{\mathbf{KEM}}$ $(\mathrm{PK}, \mathcal{T}; u) \rightarrow (\mathsf{Key}, \mathrm{CT})$. The encryption algorithm takes as input the public parameters PK, an access tree structure $\mathcal{T}$, and an optional input seed $u \in \{0,1\}^k$ to a pseudo-random generator G (see Section 2.4.2 for implementation details). The algorithm first chooses random exponent $s \in \mathbb{Z}_p$ (if seed

[6]Let ρ be an injective function that maps leaf nodes to attributes.
[7]H_1 and H_2 are keyed-hash functions instantiated using SHA-256.

$u \in \{0,1\}^k$ is specified, then use G as source of randomness. Note that the output length of G will vary based on the size of $\mathcal{T}$). We then calculate the randomized shares of s according to the access structure $\mathcal{T}$ (*e.g.*, $\mathsf{computeSecretShares}(s, \mathcal{T}) \to \lambda$). Let $\{\lambda_1, \ldots, \lambda_\ell\}$ be the shares of s where ℓ is the number of leaf nodes in $\mathcal{T}$ and ρ is an injective function that maps leaf nodes to attributes. Additionally, the algorithm chooses random $r_1, \ldots, r_\ell \in \mathbb{Z}_p$ and returns the following:

$\mathsf{Key} = H_1(e(g_1, g_2)^{\alpha s})$,
$\mathrm{CT} = \{C' = g_1^s, (C_1 = g_1^{a\lambda_1} H_2(\rho(1))^{-r_1}, D_1 = g_2^{r_1}), \ldots, (C_\ell = g_1^{a\lambda_\ell} H_2(\rho(\ell))^{-r_n}, D_\ell = g_2^{r_\ell})\}$

Decrypt$_{\mathbf{KEM}}$ $(\mathrm{CT}, \mathrm{SK}) = \mathsf{Key}$. The decryption algorithm takes as input a ciphertext CT for access structure $\mathcal{T}$ and a private key for a set of attributes γ. The algorithm first determines whether the attributes γ satisfies the access structure $\mathcal{T}$ on the ciphertext (*e.g.*, $T(\gamma) = 1$). If so, then we proceed to recover the Lagrange coefficients ω for the minimum set of attributes S necessary to satisfy $\mathcal{T}$ (*e.g.*, $\mathsf{recoverCoefficient}(\mathcal{T}, \gamma) \to (\omega, S)$). For each such attribute $i \in S$ and the corresponding coefficient ω_i, the decryption algorithm first computes:

$$\frac{e(C', K)}{\prod_{i \in S}(e(C_i, L) \cdot e(K_{\rho(i)}, D_i))^{\omega_i}} =$$

$$\frac{e(g_1, g_2)^{\alpha s} \cdot e(g_1, g_2)^{ast}}{\prod_{i \in S}(e(g_1, g_2)^{ta\lambda_i \omega_i})} = e(g_1, g_2)^{\alpha s}$$

The algorithm can then compute $\mathsf{Key} = H_1(e(g_1, g_2)^{\alpha s})$.

Implementation Details

- *Optimizations.* Let $k = |S|$. Similar to the KP-ABE KEM implementation, here we can also reduce the number of pairings from $2k+1$ to $k+2$ by bringing the Lagrange coefficients inside the pairings at the expense of increasing exponentiation from k in $\mathbb{G}_T$ to $2k$ in $\mathbb{G}_1$. Thus, this optimization yields a noticeable speed up in decryption time. The resulting decryption in OpenABE:

$$\frac{e(C', K)}{e(\prod_{i \in S} C_i^{\omega_i}, L) \cdot \prod_{i \in I}(e(K_{\rho(i)}{}^{\omega_i}, D_i))}$$

- *CCA-Security.* Recall that the above scheme is only CPA-secure, so the actual implementation uses the CCA-secure transformation via Section 2.3.8.

2.3.7 Transforming a CPA KEM to Standard Encryption

We now describe our general method for transforming any CPA-secure ABE scheme into a CCA-secure one. (Recall the discussion of encryption security categories in Section 2.2.) In the context of the OpenABE , a CPA scheme context combines a CPA KEM scheme variant (*i.e.*, any of the above ABE schemes) with a pseudo-random generator (PRG) (defined in Section 2.4.2) and forms the primitive we use to build CCA-secure ABE schemes. We use subscripts CPA′ or CPA KEM to indicate whether the algorithm is the produced CPA scheme context or the existing CPA KEM scheme respectively. The CPA scheme context leaves the setup and keygen algorithms of the underlying scheme unmodified. The construction modifies encryption and decryption as follows:

Encrypt$_{\mathbf{CPA'}}$ $(\mathrm{PK}, M, \mathbb{A}; u) \to \mathrm{CT}'$. The encrypt algorithm takes as input the public parameters PK, a message $M \in \{0,1\}^\ell$, either an access structure or set of attributes (denoted as $\mathbb{A}$), and a random seed u.

1. Run **Encrypt$_{\mathbf{CPA\_KEM}}$**$(\mathrm{PK}, \mathbb{A}; u) \to (K, \mathrm{CT}_0)$
2. Run $\mathbf{G}(K, \ell) \to y$
3. Let $\mathrm{CT}_1 = y \oplus M$

4. Output ciphertext $\mathrm{CT} = (\mathrm{CT}_0, \mathrm{CT}_1)$

$\mathbf{Decrypt}_{\mathbf{CPA'}}(\mathrm{SK}, \mathrm{CT}) = M'$

1. Run $\mathbf{Decrypt}_{\mathbf{CPA\_KEM}}(\mathrm{SK}, CT_0) = K'$
2. Compute $\ell' = \mathbf{length}(CT_1)$
3. Output $M' = \mathrm{CT}_1 \oplus \mathbf{G}(K', \ell')$

2.3.8 Chosen-Ciphertext Secure Transformation

We build on the primitive defined above and can now show how we build a chosen-ciphertext secure ABE system from any CPA scheme context. The CCA KEM algorithm will use the CPA scheme context in a black box fashion. Moreover, we will use a collision-resistant hash function $H : \{0,1\}^* \to \{0,1\}^k$ that we model as a random oracle.[8] Here we will use subscripts CCA KEM or CPA′ to indicate whether the algorithm is the produced CCA KEM scheme or the existing CPA scheme context respectively. The CCA KEM transform leaves the setup and keygen algorithms of the underlying scheme unmodified.

Finally, we assume that the underlying IND-CPA secure ABE system has a special property in that the access structure used to create a ciphertext can be efficiently extracted from that ciphertext using only the public key. In particular, we assume the existence of a deterministic algorithm $\mathbf{ExtractAccessStructure}(\mathrm{PK}, C) \to \mathbb{A}$ which has the correctness property that

$$\mathbf{ExtractAccessStructure}(\mathrm{PK}, C = \mathbf{Encrypt}_{\mathbf{CPA'}}(\mathrm{PK}, \mathbb{A}, M; u)) = \mathbb{A}$$

for all messages M and random tapes u. All IND-CPA ABE schemes that we use will (trivially) have this property; however, we note there could exist ABE schemes which do not and the below transformation would not apply to these.

The transform modifies encryption and decryption as follows:

$\mathbf{Encrypt}_{\mathbf{CCA\_KEM}}(\mathrm{PK}, \mathbb{A}) \to (K, C)$. The encryption algorithm takes as input the public parameters PK and either an access structure or set of attributes also denoted generically here as $\mathbb{A}$.

1. Choose random $K \in \{0,1\}^n$
2. Choose random $r \in \{0,1\}^n$ and let $u = H(r||K||\mathbb{A})$.
3. Run $\mathbf{Encrypt}_{\mathbf{CPA'}}(\mathrm{PK}, \mathbb{A}, M = (K, r); u) \to C$.
4. Output the key K, and ciphertext C.

$\mathbf{Decrypt}_{\mathbf{CCA\_KEM}}(\mathrm{PK}, \mathrm{SK}, C) = K' \cup \perp$.

1. Run $\mathbf{Decrypt}_{\mathbf{CPA'}}(\mathrm{SK}, C) = M' = (K', r')$
2. $\mathbb{A}' = \mathbf{ExtractAccessStructure}(\mathrm{PK}, C)$.
3. Let $u' = H(r'||K'||\mathbb{A}')$.
4. Run $\mathbf{Encrypt}_{\mathbf{CPA'}}(\mathrm{PK}, \mathbb{A}', M' = (K', r'); u') \to C'$.
5. Check $C' \overset{?}{=} C$ and if equal, output K'. Otherwise, output $\perp$.

Note that the added time to encryption is almost negligible. The overhead added to decryption is that the ciphertext will have to be re-encrypted as part of the validity check.

[8] H is a keyed-hash function instantiated using SHA-256.

2.3.9 Hybrid Encryption (using CCA KEM)

To encrypt data in the system, the OpenABE defines a CCA scheme context that builds on the CCA KEM scheme for the purposes of encrypting the original data. Specifically, the CPA scheme context combines a CCA KEM with an authenticated encryption scheme (see Section 2.3.12 for details). The CCA scheme context leaves the setup and keygen algorithms of the underlying scheme unmodified. The construction modifies encryption and decryption as follows:

$\mathbf{Encrypt_{CCA}}(\mathrm{PK}, M, \mathbb{A}) \rightarrow \mathrm{CT}$

1. Run $\mathbf{Encrypt_{CCA\_KEM}}(\mathrm{PK}, \mathbb{A}) \rightarrow (K, C_0)$.
2. AAD = $\mathbf{ExtractHeader}(C_0)$
3. Run $\mathbf{AuthEncrypt}(K, M; AAD) \rightarrow C_1$.
4. Output $\mathrm{CT} = (C_0, C_1)$.

$\mathbf{Decrypt_{CCA}}(\mathrm{PK}, \mathrm{SK}, \mathrm{CT}) = M' \cup \perp$

1. Run $\mathbf{Decrypt_{CCA\_KEM}}(\mathrm{PK}, \mathrm{SK}, C_0) = K'$.
2. AAD' = $\mathbf{ExtractHeader}(C_0)$
3. Run $M' = \mathbf{AuthDecrypt}(K', C_1; AAD')$ if $K' \neq \perp$.
4. Output M' if no error occurred. Otherwise, return $\perp$.

The **ExtractHeader** method extracts the following metadata from the ABE ciphertext: OpenABE library version, elliptic curve identifier, ABE algorithm identifier and a unique ciphertext identifier (or UUID).

2.3.10 Revocation

There are flexible semantics for revocation in ABE that can be enforced cryptographically by using time as an attribute. We recommend keeping things simple. For instance, consider two possible options that could be implemented with either KP-ABE or CP-ABE.

1. A key is tagged with the time of creation t. A ciphertext when created specifies a time X such that to decrypt, a key must have $t > X$. Also, the ciphertext has a flag stating whether revocation is enabled at all.
2. A ciphertext t has its time of creation t. A private key has a window of times X_0, X_1 where t must be within X_0 and X_1 to decrypt.

These approaches can be enhanced by additionally applying standard methods for revoking arbitrary private keys, such as maintaining a revocation list.

2.3.11 Public-Key Encryption

Elliptic Curves. We provide the following definition to denote what we mean by base elliptic curves. The definition is slightly adapted from the NIST FIPS 186-4 document [14] for curves over prime fields. For each prime p, a pseudo-random curve of prime order n is listed as follows:

$$E : y^2 \equiv x^3 - 3x + b \pmod{p}$$

Thus, for these curves, the cofactor is always $h = 1$. The elliptic curve generator $\mathsf{ECSetup}$ is an algorithm that on input a security parameter 1^τ outputs the parameters for the above pseudo-random curve $(p, n, SEED, c, a, b, g, \mathbb{G})$ where: the prime modulus is p, the group order is n, the 160-bit input seed SEED is to the SHA-1 based algorithm (*i.e.*, the domain parameter seed), the output c of the SHA-1 based algorithm, the coefficient $a \equiv -3$, the coefficient b (satisfying $b^2c \equiv -27 \pmod{p}$), and the x and y coordinates for the base point g.

Construction

This section expands on the Public-key Encryption Key Encapsulation variant implemented in the OpenABE. This is derived from the One-Pass Diffie-Hellman Key agreement protocol described in NIST SP800-56A Revision 1 [2, Section 6.2.2.2].

The construction consists of four algorithms. The setup algorithm takes as input a security parameter and selects the elliptic curve groups. The key generation algorithm takes as input a user identifier and outputs a public and private key pair. The encryption algorithm takes the sender's identifier, the recipient's identifier and the recipient's public key and outputs a symmetric key and the ciphertext. The decryption algorithm is used by an authorized recipient to decrypt a ciphertext.

Setup(τ). The setup algorithm takes as input a security parameter τ and runs $\mathsf{ECSetup}(1^\tau) \to (p, n, SEED, c, a, b, g, \mathbb{G})$. Let KDF be a key derivation function (see Section 2.4.1 for details) and let ℓ be the length of the derived key.

Keygen$(\mathsf{ID}) \to (\mathrm{PK}, \mathrm{SK})$. The key generation algorithm takes as input the group parameters and a key identifier $\mathsf{ID} \in \{0,1\}^k$. The algorithm first chooses random $a \in \mathbb{Z}_p$, and output the public key $\mathrm{PK} = g^a$ and the private key $\mathrm{SK} = a$.

$\mathbf{Encrypt_{KEM}}(\mathsf{ID_S}, \mathsf{ID_R}, \mathrm{PK}) \to (K, C)$. The encryption algorithm takes as input the identifier of the sender, and the public key of the recipient The algorithm first chooses a random $b \in \mathbb{Z}_p$, computes the shared key $Z = g^{ab}$, and returns the key K and ciphertext C as

$$K = \mathrm{KDF}(Z, \ell, \mathsf{AlgID}||\mathsf{ID_S}||\mathsf{ID_R}), \qquad C = \{\mathsf{ID_S}, g^b\}$$

$\mathbf{Decrypt_{KEM}}(\mathsf{ID_R}, \mathrm{SK}, C) = K$. The decryption algorithm takes as input the secret key of the recipient, the ciphertext and recovers $Z = g^{ab}$ using the recipient's secret key. Then, the algorithm outputs the derived key. $K = \mathrm{KDF}(Z, \ell, \mathsf{AlgID}||\mathsf{ID_S}||\mathsf{ID_R})$.

PKE Scheme Context

We combine the PKE KEM primitive above with an authenticated encryption primitive to build a CCA secure public-key encryption scheme. The construction is as follows:

$\mathbf{Encrypt_{CCA}}(\mathsf{ID_S}, \mathsf{ID_R}, \mathrm{PK}, M) \to \mathrm{CT}$

1. Run $\mathbf{Encrypt_{KEM}}(\mathsf{ID_S}, \mathsf{ID_R}, \mathrm{PK}) \to (K, C_0)$.
2. Run $\mathbf{AuthEncrypt}(K, M) \to C_1$.
3. Output $\mathrm{CT} = (C_0, C_1)$.

$\mathbf{Decrypt_{CCA}}(\mathsf{ID_R}, \mathrm{SK}, \mathrm{CT}) = M' \cup \perp$

1. Run $\mathbf{Decrypt_{KEM}}(\mathsf{ID_R}, \mathrm{SK}, C_0) = K'$.
2. Run $M' = \mathbf{AuthDecrypt}(K', C_1)$.
3. Output M' or $\perp$ if error occurred during **AuthDecrypt**.

Public and Private Key Validation. The implementation also includes validation of keys according to the ECC full public/private key validation routine in [2, Section 5.6.2.5]. This validation asserts that the public key is not the point at infinity, that each coordinate of the public key has the unique correct representation of an element, that the public key is on the correct elliptic curve, and that it has the correct order/EC subgroup. For the private key, we only have to assert that it has the correct order. If all of these assertions hold, then the keys are loaded and made available for encryption/decryption. Otherwise, a key validation error is set.

2.3.12 Authenticated Encryption

OpenABE includes an authenticated symmetric-key encryption (AE) primitive for providing confidentiality and integrity of large data objects. We use the AES-Galois Counter Mode (GCM) cipher provided by the OpenSSL library [19] with 256-bit keys. We chose the AES-GCM because it is not only efficient but secure. In addition, due to pipelining support, hardware implementations can achieve high speeds with low cost (*e.g.*, Intel's AES-NI) and low latency. Because of this balance between security and high-throughput/efficiency, we believe the AES-GCM cipher is the right AE building block for OpenABE. Moreover, OpenABE ciphertexts include a header that encodes metadata which includes a scheme identifier, the underlying elliptic curve identifier, the OpenABE library version number and a unique ciphertext identifier. To protect these portions of OpenABE ciphertexts, each encryption scheme context utilize the AES-GCM associated data as part of the tag verification. This detects tampering of OpenABE ciphertext headers along with the ciphertext contents. The following is the interface to the authenticated encryption primitive:

AuthEncrypt$(K, M; AAD) \rightarrow C$. The encryption algorithm takes as input a symmetric-key, a message $M \in \{0,1\}^\ell$ and optionally the authenticated associated data $AAD \in \{0,1\}^k$. The algorithm outputs the ciphertext C.

AuthDecrypt$(K, C; AAD) = M \cup \perp$. The decryption algorithm takes as input a symmetric-key, the ciphertext C and optionally the authenticated associated data AAD. The algorithm outputs the message M if tag verification is successful. Otherwise, it returns $\perp$.

2.3.13 Digital Signatures

OpenABE includes a digital signatures primitive provided by the OpenSSL library [19]. The OpenSSL implementation is derived from the ANSI X9.62, US Federal Information Processing Standard (FIPS) 186-2 Digital Signature Standard (DSS) [12]. The implementation uses SHA-256 to compute message digests in the ECDSA primitive. We describe the simplified interface for ECDSA and refer the reader to the FIPS 186-2 DSS document [12] for details of the scheme.

Keygen$(\tau) \rightarrow (\mathrm{PK}, \mathrm{SK})$. The key generation algorithm takes as input a security parameter τ, runs the $\mathsf{ECSetup}(1^\tau)$ to select the elliptic curve parameters and outputs the public and secret key.

Sign$(\mathrm{SK}, M) \rightarrow \sigma$. The signing algorithm takes as input a secret key SK and message $M \in \{0,1\}^*$ and outputs a signature σ.

Verify$(\mathrm{PK}, M, \sigma) = \{true, false\}$. The verification algorithm takes as input a public key PK, the message M and the signature σ. The algorithm outputs $true$ if the signatures is valid with respect to M and PK. Otherwise, it outputs $false$.

2.4 Cryptographic Tools

2.4.1 Key Derivation Functions (KDF)

We implement two types of Key Derivation Functions in OpenABE. The first type is a Concatenation Key Derivation Function (KDF) described in NIST SP 800-56A [2, Section 5.8.1]. The construction uses SHA-256 as the underlying hash function. This particular KDF is used by the public-key encryption scheme context described in Section 2.3.11. The concatenated KDF algorithm is described as follows:

KDF$(Z, \ell, M) \rightarrow K \cup \perp$. The KDF algorithm takes as input the shared secret Z (as a byte string representation), a key length ℓ that indicates the bit length of the secret keying material to be generated and the metadata $M \in \{0,1\}^n$ is the concatenation of the algorithm ID, sender's ID and recipient's ID. The algorithm outputs a derived key with the specified length.

1. $reps = \lceil \ell / hashlen \rceil$.

2. If *reps* $> (2^{32} - 1)$, then output $\perp$ and stop.
3. Initialize a 32-bit, big-endian bit string *counter* as 00000001_{16}.
4. If $counter||Z||M$ is more than *max_hash_inputlen* bits long, then output $\perp$ and stop.
5. For $i = 1$ to *reps* (increments of 1), do the following:
   (a) Compute $Hash_i = H(counter||Z||M)$.
   (b) Increment *counter* mod 2^{32} treating the result as an unsigned 32-bit integer.
6. Let $Hhash$ be set to $Hash_{reps}$ if $(\ell/hashlen)$ is an integer; otherwise, let $Hhash$ be set to the $(\ell \bmod hashlen)$ leftmost bits of $Hash_{reps}$.
7. Output $K = Hash_1||Hash_2||\ldots||Hash_{reps-1}||Hhash$.

The second type of KDF we provide is for enabling password-based key derivation. This KDF utilizes the OpenSSL library's PBKDF2 implementation and we set the underlying pseudo-random function as HMAC-SHA256. This alternative KDF takes as input a password, the length of the key, a salt[9] for the key, and an iteration count. The algorithm outputs the derived key with the specified length. At a minimum, we execute PBKDF2 with an iteration count of $10,000$ to resist brute-force attacks.

2.4.2 Pseudo-random Generator (PRG)

OpenABE includes a PRG which produces a sequence of pseudo-random bits of a desired length. The implementation is instantiated using an NIST approved block cipher, AES-256 in Counter Mode (CTR), and described in detail in NIST SP 800-90A, Rev 1 [3]. In particular, we implement Section 10.2 titled "DRBG mechanisms based on Block Ciphers." The construction is comprised of four procedures that we summarize below:

1. A **CTR_DRBG_Update** function [3, Section 10.2.1.2] that updates the internal state of the PRG using some provided data. This provided data must be exactly *seedlen* bits.

```
$\textbf{update}(provided\_data, K, V) \rightarrow (K, V)$:
(a) $temp = Null$
(b) while $(\textbf{len}(temp) < seedlen)$ do:
      i. $V = (V + 1) \bmod 2^{outlen}$
     ii. $output\_block = \textbf{AES\_ECB}(K, V)$
    iii. $temp = temp \mid\mid output\_block$
(c) $temp = \textbf{leftMostBits}(temp, seedlen)$
(d) $temp = temp \oplus provided\_data$
(e) $K = \textbf{leftMostBits}(temp, keylen)$
(f) $V = \textbf{rightMostBits}(temp, outlen)$
(g) return $K$ and $V$.
```

2. An **CTR_DRBG_Instantiate** function [3, Section 10.2.1.3.2] that is called prior to generating pseudo-random bits. It checks the input parameters, initializes the seed material and derives the initial working state of the PRG. Let **Block_Cipher_df** be the derivation function specified in [3, Section 10.3.2].

   $\textbf{CTR\_DRBG\_Instantiate}(entropy\_input, nonce, person\_string) \rightarrow (K, V, reseed\_counter)$:

[9] salt is random data used as an additional entropy

(a) $seed\_material = entropy\_input \mid\mid nonce \mid\mid person\_string$
(b) $seed\_material = \textbf{Block\_Cipher\_df}(seed\_material, seedlen)$
(c) $K = 0^{keylen}$
(d) $V = 0^{outlen}$
(e) $(K, V) = \textbf{CTR\_DRBG\_Update}(seed\_material, K, V)$
(f) $reseed\_counter = 1$
(g) return K, V and $reseed\_counter$ as the initial working state

3. A **CTR_DRBG_Reseed** function [3, Section 10.2.1.4.2] that inserts additional entropy input into the generation of pseudo-random bits. This is triggered when over $100,000$ requests have been made to the PRG (default recommended by NIST but can be adjusted as necessary).

   $\textbf{CTR\_DRBG\_Reseed}(seed\_material, K, V) \rightarrow (K, V, reseed\_counter)$:

   (a) $seed\_material = entropy\_input \mid\mid additional\_input$. Ensure that the length of the $seed\_material$ is exactly $seedlen$ bits.
   (b) $seed\_material = \textbf{Block\_Cipher\_df}(seed\_material, seedlen)$
   (c) $(K, V) = \textbf{CTR\_DRBG\_Update}(seed\_material, K, V)$
   (d) $reseed\_counter = 1$.
   (e) return K, V, and $reseed\_counter$ as the new working state.

4. A **CTR_DRBG_Generate** function [3, Section 10.2.1.5.2] that produces pseudorandom bits after instantiation or reseeding. It checks the validity of input parameters and calls the reseed function to obtain sufficient entropy (if deemed necessary). Then, it generates the requested pseudorandom bits using the generate algorithm. Finally, it updates the working state of the PRG and returns the requested pseudo-random bits to the calling application.

   $\textbf{CTR\_DRBG\_Generate}(K, V, reseed\_counter, \ell) \rightarrow (prand\_bits, K', V', reseed\_counter')$:

```
(a) If ($reseed\_counter > reseed\_interval$), then set $error\_status = \textbf{RESEED}$ to indicate that a reseed is required.
(b) if ($add\_input \neq Null$), then
      • $addl\_input = \textbf{Block\_Cipher\_df}(addl\_input, seedlen)$
      • $(K, V) = \textbf{CTR\_DRBG\_Update}(addl\_input, K, V)$
(c) else $addl\_input = 0^{seedlen}$
(d) $temp = Null$
(e) $outlen = \ell$
(f) while ($\textbf{len}(temp) < outlen$) do:
      i. $V = (V + 1) \bmod 2^{outlen}$
      ii. $output\_block = \textbf{AES\_ECB}(K, V)$
      iii. $temp = temp \mid\mid output\_block$
(g) $prand\_bits = \textbf{leftMostBits}(temp)$
(h) $(K', V') = \textbf{update}(addl\_input, K, V)$
(i) $reseed\_counter = reseed\_counter + 1$
(j) return $prand\_bits, K', V'$ and $reseed\_counter$ as the new working state.
```

With the above procedures defined, we now define the high-level PRG function used in the schemes as follows:

$\mathbf{G}(K, \ell) \rightarrow R$. At a high-level, the algorithm takes as input a key (or initial seed) $K \in \{0,1\}^n$, an output length ℓ and the algorithm returns a pseudo-random sequence of bits $R \in \{0,1\}^\ell$. In addition, we will use a collision-resistant hash function $H_1 : \{0,1\}^n \rightarrow \{0,1\}^m$ to produce the nonce required by the CTR DRBG specification. The algorithm is as follows:

```
1. $entropy\_input = K$ and $nonce = H_1(K)$.
2. $person\_string = Null$. Because personalization strings are optional for the DRBG, we exclude them in the implementation. This could change in the future.
3. if not $PRG\_init$, then
   (a) $(Key, V, reseed\_counter) \leftarrow \mathbf{CTR\_DRBG\_Instantiate}(entropy\_input, nonce, person\_string)$
   (b) $PRG\_init = true$
4. $(R, Key', V', reseed\_counter') \leftarrow \mathbf{CTR\_DRBG\_Generate}(Key, V, reseed\_counter, \ell)$
5. if $error\_status = RESEED$, then
   (a) $\mathbf{CTR\_DRBG\_Reseed}(seed\_material, Key', V')$
6. return pseudo-random bits $R$.
```

2.4.3 Random Number Generation (RNG)

OpenABE includes a default global random number generator (RNG) that wraps the cryptographically strong pseudo-random number generator provided by the OpenSSL library [19]. During initialization of OpenABE , we first seed the OpenSSL RNG (which by default pulls entropy from /dev/urandom). We note here that the PRG (see Section 2.4.2) primitive extends the RNG interface such that the RNG can be swapped with a PRG at any place in OpenABE. In addition, we provide a callback mechanism for controlling how the pairing and elliptic curve modules (see Section 2.5) obtain randomness for selecting group elements and so on. In effect, this approach makes it possible to potentially support *any* elliptic curve/pairing math library without needing to rely on or trust that library's source of randomness.

2.4.4 Keystore

The OpenABE Keystore manages public and private key material for all core cryptographic algorithms supported in the library. It provides key storage, import/export of keys to/from memory with support for encrypting keys under a given passphrase via PBKDF2 (see Section 2.4.1). The keystore implementation includes some features for easing the management of ABE keys by including searching functionality for keys to match a given ciphertext. ABE keys can be further queried based on a number of metrics such as efficiency, key expiration, and so on. In addition, the keystore provides encapsulation of key material and simply allows reference to keys by string identifiers. This approach facilitates integration of OpenABE into existing applications in a seamless and intuitive manner.

2.5 Zeutro Math Library

The Zeutro Math Library (ZML) provides a generic mathematics API to support base elliptic curves and other curves that support bilinear operations. As mentioned before, the centralized nature of this API makes it easy to update or replace subcomponents without affecting higher level scheme implementations. Moreover, design approach enables deploying different versions of OpenABE that take advantage of specific strengths in different external mathematics libraries.

In the sections which follow, we describe the components that comprise ZML and its core functionalities.

2.5.1 Pairing Module

The ZML pairing module provides pairing-friendly, ordinary elliptic curves to support the attribute-based encryption schemes. This module is built on top of the RELIC library [1] which supplies all of the bilinear operations (including the pairing operation − see Section 2.3.1). We instantiate the schemes using the state-of-the-art Barreto-Naehrig (BN) curves [5] with the embedding degree $k = 12$ (or commonly referred to as BN-254).[10] This particular asymmetric curve is known to yield a very efficient pairing implementation and a security level equivalent to AES-128. As a result, this boosts the overall performance of ABE scheme implementations over prior efforts. Other benefits of BN curves include the ability to compress the representation of group elements. This directly translates to making ABE ciphertexts more compact which considerably reduces transmission costs. One downside is that because BN curves are *Type-III* pairings, it only permits efficient hashing to the group $\mathbb{G}_1$.

Alternatively, the pairing module could also be instantiated with similar asymmetric curves that yield less efficient pairing implementations but provide higher security levels. These include the Barreto-Lynn-Scott (BLS) curve [4] with the embedding degree $k = 24$ and the Kachisa-Schaefer-Scott (KSS) curves [13] with embedding degree $k = 18$. In general, the choice of pairing curve will depend on a variety of factors including the number of pairing computations required and non-pairing operations needed (*e.g.*, exponentiations, multiplication, hashing, etc). As new pairing-friendly curves are discovered, these curves can be integrated if they maximize efficiency while preserving security.

2.5.2 Elliptic Curve (EC) Module

ZML exposes only standardized, NIST-recommended elliptic curves to support public-key encryption and digital signature algorithms. We build this module on top of the OpenSSL EC library [19]. Optionally, the EC module can also be instantiated with the base elliptic curve component of the RELIC library [1].

Although there are several special NIST curves to choose from [14], we restrict the EC module to prime fields of the following sizes: P-192, P-224, P-256, P-384, and P-521. By default, the schemes are initialized with Curve P-256 (equivalent to AES-128 bit security). Using these special curves helps to optimize the efficiency of the elliptic curve operations which include exponentiation (or scalar multiplication) and multiplication (or point addition).

[10]Note that we can also use BN-638 which provides a higher-security level (roughly equivalent to AES-256).

Chapter 3

Security Proofs

3.1 Proof of Security for CCA Transformation of Section 2.3.8

3.1.1 Proving KEM to Message Encryption Transformation

We first prove security of our transformation from a KEM scheme into a secure CPA scheme. We do so by providing a sequence of hybrid games with the first one being IND-CPA security for an ABE scheme. We both show that any probabilistic polynomial time (PPT) attacker has negligible difference of advantage between consecutive games and that any attacker's advantage in the last game is 0. Therefore any PPT adversary has at most negligible advantage.

We begin by describing the sequence of games.

Game $_{\text{CPA}}$

1. The challenger runs setup and give the ABE scheme's public key, PK to the attacker.
2. The attacker submits two messages M_0, M_1 to the challenger.
3. The challenger then flips a coin $\beta \in \{0, 1\}$.
4. The challenger runs $\mathbf{Encrypt}_{\mathbf{CPA\_KEM}}(\text{PK}, \mathbb{A}; u) \to (K', \text{CT}_0^*)$ it then sets $K = K'$.
5. The challenger computes $\mathbf{G}(K, \ell) \to y$
6. Finally it computes $\text{CT}_1^* = y \oplus M_\beta$
7. The challenge ciphertext $\text{CT}^* = (\text{CT}_0^*, \text{CT}_1^*)$ is sent to the attacker.
8. The attacker submits $\beta' \in \{0, 1\}$ and wins iff $\beta = \beta'$.

Game $_1$

1. The challenger runs setup and give the ABE scheme's public key, PK to the attacker.
2. The attacker submits two messages M_0, M_1 to the challenger.
3. The challenger then flips a coin $\beta \in \{0, 1\}$.
4. The challenger runs $\mathbf{Encrypt}_{\mathbf{CPA\_KEM}}(\text{PK}, \mathbb{A}; u) \to (K', \text{CT}_0^*)$ it then chooses K randomly from the KEM key space.
5. The challenger computes $\mathbf{G}(K, \ell) \to y$.

6. Finally it computes $\mathrm{CT}_1^* = y \oplus M_\beta$.
7. The challenge ciphertext $\mathrm{CT}^* = (\mathrm{CT}_0^*, \mathrm{CT}_1^*)$ is sent to the attacker.
8. The attacker submits $\beta' \in \{0,1\}$ and wins iff $\beta = \beta'$.

Game $_2$

1. The challenger runs setup and give the ABE scheme's public key, PK to the attacker.
2. The attacker submits two messages M_0, M_1 to the challenger.
3. The challenger then flips a coin $\beta \in \{0,1\}$.
4. The challenger runs $\mathbf{Encrypt}_{\mathbf{CPA\_KEM}}(\mathrm{PK}, \mathbb{A}; u) \to (K', \mathrm{CT}_0^*)$ it then chooses K randomly from the KEM key space.
5. The challenger chooses $y \in \{0,1\}^\ell$ uniformly at random.
6. Finally it computes $\mathrm{CT}_1^* = y \oplus M_\beta$.
7. The challenge ciphertext $\mathrm{CT}^* = (\mathrm{CT}_0^*, \mathrm{CT}_1^*)$ is sent to the attacker.
8. The attacker submits $\beta' \in \{0,1\}$ and wins iff $\beta = \beta'$.

Lemma 3.1.1 *If our underlying ABE KEM is IND-CPA secure, then the difference of advantage of any PPT attacker in* Game $_1$ *and* Game $_{\mathrm{CPA}}$ *is negligible.*

Consider an algorithm $\mathcal{B}$ that plays the ABE-KEM IND-CPA security game. It first receives a public key PK from the ABE challenger and passes this on to $\mathcal{A}$ for step 1. It then runs steps 2-3 itself. Next it receives the KEM challenge ciphertext CT_0^* and the key K. Depending on the KEM challenger's coin flip K is either randomly chosen or generated as $\mathbf{Encrypt}_{\mathbf{CPA\_KEM}}(\mathrm{PK}, \mathbb{A}; u) \to (K', \mathrm{CT}_0^*)$. $\mathcal{B}$ runs steps $5-8$ using the challenge value K given by the PRG challenger. If the attacker wins (i.e. $\beta = \beta'$) $\mathcal{B}$ outputs 1 to indicate that K was generated from a call to encrypt; otherwise, it outputs 0 to indicate it was randomly chosen.

We observe that if K was generated from encryption $\mathcal{B}$ simulates Game $_{\mathrm{CPA}}$ and if it was randomly chosen it simulates Game $_1$. Therefore, the difference of advantage of any attacker in Game $_1$ from Game $_{\mathrm{CPA}}$ will be the advantage of $\mathcal{B}$ in the IND-CPA security game.

Lemma 3.1.2 *If our underlying pseudo random generator G is secure, then the difference of advantage of any PPT attacker in* Game $_2$ *and* Game $_1$ *is negligible.*

Consider an algorithm $\mathcal{B}$ that plays the PRG security game. It takes in a challenge $y \in \{0,1\}^\ell$ uniformly at random or is generated as $\mathbf{G}(K, \ell) \to y$. The reduction algorithm runs steps 1-4 of Game $_1$ itself, except it does not choose K. Then it runs steps $6-8$ using the challenge value y given by the PRG challenger. If the attacker wins (i.e. $\beta = \beta'$) $\mathcal{B}$ outputs 1 to indicate that y was pseudorandomly chosen; otherwise, it outputs 0 to indicate it was randomly chosen.

We observe that if y was pseudorandomly chosen $\mathcal{B}$ simulates Game $_1$ and if it was randomly chosen it simulates Game $_2$. Therefore, the difference of advantage of any attacker in Game $_2$ from Game $_1$ will be the advantage of $\mathcal{B}$ in the PRG security game.

Lemma 3.1.3 *The advantage of any (not necessarily PPT) attacker in* Game $_2$ *is 0.*

This follows from the fact that since y is chosen uniformly at random that $y \oplus M_\beta$ will be distributed as a uniformly random string and not carry any information about β.

Theorem 3.1.4 *Assuming the underlying PRG is secure and the underlying CPA KEM is secure, the ABE CPA KEM to ABE message encryption scheme transformation produces and IND-CPA secure scheme.*

The theorem follows immediately from the above three lemmas.

3.1.2 Proving CCA Tranformation

We now prove security of our transformation from a IND-CPA secure ABE scheme into a CCA secure CPA scheme. We do so by providing a sequence of hybrid games with the first one being IND-CCA security for an ABE scheme. We both show that any probabilistic polynomial time (PPT) attacker has negligible difference of advantage between consecutive games and that any attacker's advantage in the last game is 0. Therefore any PPT adversary has at most negligible advantage. Our security proof will model a hash function H as a random oracle.

We begin by describing the sequence of games.

Game $_{\text{CCA}}$

1. The challenger runs setup and give the ABE scheme's public key, PK to the attacker.
2. The challenger records all queries $(r||K||\mathbb{A})$ and responds with a random output u if this is the first time the input has been queried on. Otherwise, it gives back the previous response. This oracle operation is run throughout the game.
3. The challenger runs the decryption on any ciphertext CT submitted for any polynomial number of times.
4. The challenger receives an challenge access structure $\mathbb{A}^*$ from the attacker.
5. The challenger then flips a coin $\beta \in \{0, 1\}$. Next it computes $\textbf{Encrypt}_{\textbf{CCA\_KEM}}(\text{PK}, \mathbb{A}^*) \rightarrow (K', \text{CT}^*)$. If $\beta = 0$ it sets $K^* = K'$; otherwise it chooses K at random.
6. The values CT^*, K^* are sent to the attacker.
7. The challenger runs the decryption on any ciphertext $\text{CT} \neq \text{CT}^*$ submitted for any polynomial number of times.
8. The attacker submits $\beta' \in \{0, 1\}$ and wins iff $\beta = \beta'$.

Game $_1$

1. The challenger runs setup and give the ABE scheme's public key, PK to the attacker.
2. The challenger records all queries $(r||K||\mathbb{A})$ and responds with a random output u if this is the first time the input has been queried on. In addition compute $\textbf{Encrypt}_{\textbf{CPA}'}(\text{PK}, \mathbb{A}, M = (K, r); u) \rightarrow \text{CT}$ and enter both CT, u into the table for $(r||K||\mathbb{A})$. (Only u is given back as a response.)
3. When given a ciphertext CT the challenger checks if CT appears as a ciphertext entry in the random oracle table. If so, it outputs the corresponding K value in the table; otherwise, it outputs $\perp$ to reject.
4. The challenger receives an challenge access structure $\mathbb{A}^*$ from the attacker.
5. The challenger then flips a coin $\beta \in \{0, 1\}$. Next it computes $\textbf{Encrypt}_{\textbf{CCA\_KEM}}(\text{PK}, \mathbb{A}) \rightarrow (K', \text{CT}^*)$. If $\beta = 0$ it sets $K^* = K'$; otherwise it chooses K^* at random.
6. The values CT^*, K^* are sent to the attacker.
7. When given a ciphertext $\text{CT} \neq \text{CT}^*$ the challenger checks if CT appears in the random oracle table. If so, it outputs the corresponding K value in the table; otherwise, it outputs $\perp$ to reject.
8. The attacker submits $\beta' \in \{0, 1\}$ and wins iff $\beta = \beta'$.

Game $_2$

1. The challenger runs setup and gives the ABE scheme's public key, PK to the attacker.
2. The challenger records all queries $(r||K||\mathbb{A})$ and responds with a random output u if this is the first time the input has been queried on. In addition, compute $\textbf{Encrypt}_{\textbf{CPA}'}(\text{PK}, \mathbb{A}, M = (K, r); u) \to C$ and enter both C, u into the table for $(r||K||\mathbb{A})$. (Only u is given back as a response.)
3. When given a ciphertext CT the challenger checks if CT appears the random oracle table. If so, it outputs the corresponding K value in the table; otherwise, it outputs $\perp$ to reject.
4. The challenger receives an challenge access structure $\mathbb{A}^*$ from the attacker.
5. The challenger then flips a coin $\beta \in \{0, 1\}$. Next it computes the challenge ciphertext by choosing random $K' \in \{0, 1\}^n$, random $r \in \{0, 1\}^n$ and random u. (Note u is not chosen by calling the random oracle.) It then sets $\textbf{Encrypt}_{\textbf{CPA}'}(\text{PK}, \mathbb{A}^*, M = (K', r); u) \to \text{CT}^*$. If $\beta = 0$ it sets $K^* = K'$; otherwise it chooses K^* at random.
6. The values CT^*, K^* are sent to the attacker.
7. When given a ciphertext $\text{CT} \neq \text{CT}^*$ the challenger checks if CT appears n the random oracle table. If so, it outputs the corresponding K value in the table; otherwise, it outputs $\perp$ to reject.
8. The attacker submits $\beta' \in \{0, 1\}$ and wins iff $\beta = \beta'$.

Game $_3$

1. The challenger runs setup and gives the ABE scheme's public key, PK to the attacker.
2. The challenger records all queries $(r||K||\mathbb{A})$ and responds with a random output u if this is the first time the input has been queried on. In addition, compute $\textbf{Encrypt}_{\textbf{CPA}'}(\text{PK}, \mathbb{A}, M = (K, r); u) \to C$ and enter both C, u into the table for $(r||K||\mathbb{A})$. (Only u is given back as a response.)
3. When given a ciphertext CT the challenger checks if CT appears as some C in the random oracle table. If so, it outputs the corresponding K value in the table; otherwise, it outputs $\perp$ to reject.
4. The challenger receives an challenge access structure $\mathbb{A}^*$ from the attacker.
5. The challenger then flips a coin $\beta \in \{0, 1\}$. Next it computes the challenge ciphertext by choosing random $K', \tilde{K} \in \{0, 1\}^n$, random $r \in \{0, 1\}^n$ and random u. (Note u is not chosen by calling the random oracle.) It then sets $\textbf{Encrypt}_{\textbf{CPA}'}(\text{PK}, \mathbb{A}^*, M = (\tilde{K}, r); u) \to \text{CT}^*$. If $\beta = 0$ it sets $K^* = K'$; otherwise it chooses K at random.
6. The values CT^*, K^* are sent to the attacker.
7. When given a ciphertext $\text{CT} \neq \text{CT}^*$ the challenger checks if CT appears as some C in the random oracle table. If so, it outputs the corresponding K value in the table; otherwise, it outputs $\perp$ to reject.
8. The attacker submits $\beta' \in \{0, 1\}$ and wins iff $\beta = \beta'$.

Lemma 3.1.5 *If the underlying scheme is IND-CPA secure, then the difference advantage between* Game CCA *and* Game $_1$ *of any PPT attacker is negligible.*

The only difference in the two games is how decryption queries are handeled. We consider two cases. In the first case, a query CT is given where CT is some ciphertext entry in the random oracle table. However, in that case CT is exactly the ciphertext produced when encrypting to access structure $\mathbb{A}$ with randomness K, r. Since it is a correct encryption of the key K, decryption should produce K. Therefore decryption is correct.

In the second case CT is not on the list. In this case the original game might decrypt, but Game $_1$ will always reject. We argue that the chance the original game would decrypt successfully is negligible. Consider such a ciphertext CT where **ExtractAccessStructure**(PK, CT) = $\mathbb{A}$. Now suppose that $\mathbf{Decrypt_{CPA'}}(\mathrm{SK}, \mathrm{CT}) = M' = (K', r')$ and $u' = H(r||K||\mathbb{A})$. The CCA decryption algorithm will reject except in the event that $\mathbf{Encrypt_{CPA'}}(\mathrm{PK}, \mathbb{A}, M' = (K', r'); u') \rightarrow \mathrm{CT}$. However, since the random oracle was not queries on $(r||K||\mathbb{A})$, the probability that this event happens is bounded by the probability of apriori guessing a ciphertext output by an encryption for a given message (without knowing the randomness used to encrypt). If the underlying scheme is IND-CPA secure, this must occur with negligible probability.

Since the probability of producing a ciphertext that decrypts differently is negligible, the difference of advantage is negligible.

Lemma 3.1.6 *If the underlying scheme is IND-CPA secure, then the difference advantage in* Game 1 *and* Game 2 *of any PPT attacker is negligible.*

Let $(\mathbb{A}^*, K', r)$ be the tuple used to create the challenge ciphertext. Let EVENT be the event that the attacker queries the random oracle on this tuple. We observe that the attacker's view in games Game $_1$ and Game $_2$ are information theoretically identical up until this event happens. Thus we can argue that the difference in advantage is negligibly close if EVENT happens with negligible probability.

We now argue that EVENT does indeed occur with negligible probability. Suppose there was some attacker $\mathcal{A}$ that triggered EVENT with non-negligible probability ϵ, then we can break the underlying CPA security. We create a reduction algorithm $\mathcal{B}$ that does the following. It runs the security game as in Game $_1$, but where it receives the ABE public key PK from the IND-CPA challenger. Note in Game $_1$ it is able to answer decryption keys without the secret key. It receives $\mathbb{A}^*$ and then submits $(\mathbb{A}^*, M_0 = (K', r))$ and $(\mathbb{A}^*, M_1 = (K', \tilde{r}))$ to the IND-CPA challenger for randomly chosen $r, \tilde{r}$ and receives back ciphertext CT* which it uses to simulate the game. It runs the simulation until there is a query either for $(\mathbb{A}^*, (K', r))$ or $(\mathbb{A}^*, (K', \tilde{r}))$. If it is for the former it guesses 0; otherwise, it guesses 1. If neither is queried it takes a random guess.

We can see that if the challenge ciphertext were an encryption of (K', r) an oracle query on $(\mathbb{A}^*, (K', r))$ would occur with probability ϵ and a query on $(\mathbb{A}^*, (K', \tilde{r}))$ would occur with negligible probability since the $\tilde{r}$ is of security parameter length. Likewise, if challenge ciphertext were an encryption of $(K', \tilde{r})$ an oracle query on $(\mathbb{A}^*, (K', \tilde{r}))$ would occur with probability ϵ and a query on $(\mathbb{A}^*, (K', r))$ would occur with negligible probability. It follows that $\mathcal{B}$ breaks IND-CPA with advantage negligibly close to ϵ.

Lemma 3.1.7 *If the underlying scheme is IND-CPA secure, then the difference advantage in* Game 2 *and* Game 3 *of any PPT attacker is negligible.*

The security proof maps immediately to an IND-CPA game since (1) no secret key is used for decryption and (2) the randomness in creating the IND-CPA portion of challenge ciphertext is chosen truly at random. A reduction algorithm $\mathcal{B}$ will run steps $1 - 4$ itself and will submit $K'|r$ and a random $\tilde{K}|r$ as messages along with the access structure $\mathbb{A}^*$ (given from the attacker) to the IND-CPA challenger and get back an IND-CPA ciphertext C. It then uses this to play the rest of Game $_2$. The view of the attacker is the same as Game $_3$ when $\tilde{K}$ is encrypted. When K' is encrypted by the IND-CPA challenger it is the same as the advantage of the attacker in Game $_2$.

Lemma 3.1.8 *If the underlying scheme is IND-CPA secure, then the advantage in* Game 3 *of any PPT attacker is 0.*

The advantage of the attacker of distinguishing between $K = K'$ and randomly chosen K is negligible when $\tilde{K}$ is encrypted (instead of K').

Theorem 3.1.9 *Assuming the underlying ABE scheme is IND-CPA secure then the transformed scheme is CCA secure in the random oracle model.*

The theorem follows immediately from the above lemmas.

Bibliography

[1] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient Library for Cryptography. http://code.google.com/p/relic-toolkit/.

[2] Elaine Barker, Don Johnson, and Miles Smid. Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography. NIST Special Publication 800-56 Rev 1 of March 2007.

[3] Elaine Barker and John Kelsey. Recommendation for Random Number Generation Using Deterministic Random Bit Generators. NIST Special Publication 800-90A Rev 1 of January 2012.

[4] PauloS.L.M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Giuseppe Persiano, and Clemente Galdi, editors, *Security in Communication Networks*, volume 2576 of *Lecture Notes in Computer Science*, pages 257–267. Springer Berlin Heidelberg, 2003.

[5] PauloS.L.M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer Berlin Heidelberg, 2006.

[6] Amos Beimel. *Secure Schemes for Secret Sharing and Key Distribution*. PhD thesis, Israel Institute of Technology, Technion, Haifa, Israel, 1996.

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

[8] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the rsa encryption standard pkcs #1. In *CRYPTO*, pages 1–12, 1998.

[9] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: a new vision for public-key cryptography. *Communications of the ACM*, 55(11):56–64, 2012.

[10] J. Daemen and V. Rijmen. FIPS 197: Announcing the Advanced Encryption Standard (AES). NIST Special Publication of November 2001.

[11] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 89–98, New York, NY, USA, 2006. ACM.

[12] Timothy A. Hall and Sharon S. Keller. The FIPS 186-4 Elliptic Curve Digital Signature Algorithm Validation System (ECDSA2VS). NIST Special Publication of March 2014.

[13] EzekielJ. Kachisa, EdwardF. Schaefer, and Michael Scott. Constructing brezing-weng pairing-friendly elliptic curves using elements in the cyclotomic field. In StevenD. Galbraith and KennethG. Paterson, editors, *Pairing-Based Cryptography Pairing 2008*, volume 5209 of *Lecture Notes in Computer Science*, pages 126–135. Springer Berlin Heidelberg, 2008.

[14] Cameron F. Kerry and Patrick D. Gallagher. FIPS 186-4 Digital Signature Standard (DSS). Federal Information Processing Standards Publication (Issued July 2013).

[15] Matthew Pirretti, Patrick Traynor, Patrick McDaniel, and Brent Waters. Secure attribute-based systems. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 99–112, New York, NY, USA, 2006. ACM.

[16] Matthew Pirretti, Patrick Traynor, Patrick McDaniel, and Brent Waters. Secure attribute-based systems. *Journal of Computer Security*, 18(5):799–837, 2010.

[17] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[18] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[19] The OpenSSL Project. OpenSSL: The open source toolkit for SSL/TLS, September 2014. www.openssl.org.

[20] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *Public Key Cryptography PKC 2011*, volume 6571 of *Lecture Notes in Computer Science*, pages 53–70. Springer Berlin Heidelberg, 2011.