BDABE
Blockchain-based Distributed Attribute based Encryption

Georg Bramm, Mark Gall and Julian Schütte
Fraunhofer AISEC, Garching near Munich, Germany
{

Keywords: Blockchain, Attribute based Encryption, Key Revocation, Key Issuing, Key Management, Cryptographic Access Control.

Abstract: Attribute Based Encryption (ABE) denotes asymmetric cryptographic schemes where key pairs are created for attribute owners and often applied to realize a fine-grained, cryptographic access control mechanism for outsourced data. Despite the benefits of ABE systems, there are still drawbacks when ABE systems are transformed into real world applications. Mainly, ABE systems suffer from non-efficiency or non-existence of revocation mechanisms and user key coordination problems. By introducing a consensus driven approach, we try to mitigate these issues in distributed systems. In this paper, we propose a collaborative attribute management protocol for Ciphertext-policy attribute-based encryption (CP-ABE) schemes based on our own scheme called a Blockchain-based Distributed Attribute Based Encryption (BDABE) scheme. Our construction realizes distributed issue, storage and revocation of private attribute keys by adding a consensus driven infrastructure, a blockchain. We enhance both security and efficiency of key management in distributed CP-ABE systems for the application of cloud data sharing.

1 INTRODUCTION

Ten years after the introduction of Bitcoin, the Blockchain technology has evolved from a mere cryptocurrency system to a plethora of distributed ledger systems which establish trust in the correctness of code execution and shared storage without the need for a common trusted party. High availability, Immutability, transparency, and the distribution of trust make Blockchains not only interesting for all kinds of financial applications, but also for cryptographic schemes which currently often rely on an ultimately trusted key server, such as Attribute Based Encryption (ABE).

ABE is a new class of encryption schemes that allow the encryption of data under an access policy comprised of attributes. In Ciphertext-policy attribute-based encryption (CP-ABE) schemes the access policy becomes part of the ciphertext, during the encryption of data. In classic public key cryptography data is encrypted for a specific recipient using his private key, i.e. you have to know the intended recipient of the data and her public key. It is obvious that such constructions do not scale when the set of collaborators changes continuously, as the data set would have to be encrypted respectively re-encrypted for every legitimate user whenever a collaborator is added or removed. Hybrid schemes offer a solution, but are also complex to handle, when the number of participants rises.

By contrast, a CP-ABE scheme allows a user to encrypt data for any attribute such as ”supervisor” or ”employee” without knowing exactly the respective individuals in possession of these attributes. Even further, ABE allows the decryption of ciphertexts created long before a corresponding key, in the assumption that the attributes of the key match the access policy. A salient advantage of ABE systems is that problems solved by trusting traditional access control systems can now be solved cryptographically. As a result of this, the data itself can be stored publicly, but can only be decrypted by legitimate users. A central key management service is in charge of assigning attributes to users and issuing individually generated private keys to them. However, a major challenge in ABE systems has been efficient revocation of user keys and the absolute trust in the key server with respect to correctly issuing private keys only to legitimate users. Another challenge is the lack of transparency regarding access rights from the perspective of an encryptor. We try to address these challanges in the following work.

Bramm, G., Gall, M. and Schütte, J.
BDABE - Blockchain-based Distributed Attribute based Encryption.
DOI: 10.5220/0006852600990110
In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications (ICETE 2018) - Volume 2: SECRYPT*, pages 99-110
ISBN: 978-989-758-319-3

1.1 Related Work

With Attribute Based Encryption (ABE) a fine-grained, cryptographic access control mechanism for outsourced data was introduced by Goyal et al. (2006) and Bethencourt et al. (2007), which followed and improved the idea introduced by Boneh and Franklin (2001) to build a cryptographic scheme based on attributes. Many successors expanded and improved these schemes regarding efficiency like Agrawal and Chase (2017) but also regarding functionality, like introducing a decentralized setting, such as Chase (2007), Müller et al. (2008) and Lewko and Waters (2011). But there are still a lot of open problems concerning the practical usage of ABE schemes, especially in terms of key management and revocation mechanisms. Recent work from Lin et al. (2017) as well as Jemel and Serhrouchni (2017) and Jahid et al. (2011) tries to address these issues in one way or another. While Jemel and Serhrouchni (2017) used an additional temporal dimension for revocation purposes, Lin et al. (2017) and Jahid et al. (2011) introduced additive entities such as Key Authorities or Decryption Proxies in order to solve these open issues.

Our approach differs from the previous ones in that we propose to mitigate key revocation and management issues by introducing a distributed infrastructure, a distributed Blockchain, as a solution.

DAC-MACS by Yang and Jia (2014) and DACC by Ruj et al. (2011) also offer promising approaches to realize a system for distributed cryptographic access control, utilizing ABE. However, DAC-MACS requires a global certificate authority (CA) and imposes some overhead in the management of users and authorities. A disadvantage of DACC is given by the fact that each user is given a fixed set of attributes when it registers with an authority. The set of attributes of attributes can not be changed dynamically.

Another approach for distributed cryptographic access control by Sarhan and Carr (2017), called "ADB-SMC", combines ABE with active data bundles and secure multiparty computation. Besides introducing computational and communication overhead, their system involves an additional infrastructure, but in difference to our approach it uses a Distributed Hash Table rather than a blockchain infrastructure that we build upon.

FairAccess by Ouaddah et al. (2016) is a fully decentralized pseudonymous and privacy preserving authorization management framework. FairAccess uses and adapts the blockchain as a decentralized access control management tool. The idea to use a blockchain as a distributed ledger for access control management decisions is similar to ours, but unlike ours their proposed solution does not provide complete confidentiality of the data by utilizing a cryptographic access control approach.

1.2 Our Contribution

We propose a Blockchain-based Distributed Attribute Based Encryption (BDABE) protocol as a means to apply a distributed ABE scheme to the real-world using a consensus driven infrastructure to solve general problems of ABE schemes. BDABE is a new access control mechanism, where an Attribute Authority (AU) decides which of the attributes in its domain should be assigned to a user. Users, data owners as well as data consumers, can combine different attributes from multiple AU in keys for decryption and in access policies used for the encryption of ciphertexts. The novelty of our protocol is the addition of a Blockchain, which casts the trust of the attribute to user mapping in the system from a single authority to a distributed ledger. The Blockchain acts as a reference for reflecting the mapping between all users and all attributes in the system, thus enabling a reliable, traceable chain of delegated access rights. The Blockchain provides everyone with a working proof of a decentralized trust.

1.3 Structure

The remainder of this paper is structured as follows. In section 2 we introduce the notation and give background information for our work. In section 3 we then describe our BDABE protocol by illustrating the relationship between roles, attributes and keys. After that, we introduce our distributed ABE scheme in section 4 and then show how the scheme can be used in our BDABE protocol with a blockchain in section 5. After a security analysis in section 6 and a discussion in section 8, we finally give conclusions and show future directions of research in section 9.

2 PRELIMINARIES

Before we describe our BDABE construction, we give some background information on bilinear pairings, our notation, access structures and the complexity assumptions our protocol is based on.

2.1 Bilinear Pairings

Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be groups of prime order $p > 3$, and let g_1 and g_2 be generators of groups $\mathbb{G}_1$ and $\mathbb{G}_2$, i.e.

$\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$. Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map, with the following properties:

1. *Bilinearity*: For random $u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p^*$ we have $e(u^a, v^b) = e(u, v)^{ab}$
2. *Non-degeneracy*: $e(g_1, g_2) \neq 1$
3. e can be efficiently computed

We say that e is an asymmetric pairing, if $\mathbb{G}_1 \neq \mathbb{G}_2$ and call it a Type-3 pairing (as opposed to Type-2) iff no efficiently-computable isomorphism $\psi : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is known from $\mathbb{G}_2$ to $\mathbb{G}_1$ (or from $\mathbb{G}_1$ to $\mathbb{G}_2$) as Galbraith et al. (2008) defined.

2.2 Notation

We use $H : \{0,1\}^* \to \mathbb{Z}_p$ to denote a hash function that maps any arbitrary string to elements of the target group $\mathbb{G}_T$. With $a = \{0,1\}^*$ we denote a wallet address in a permissioned Blockchain. A Blockchain is denoted $\mathfrak{B}$, an attribute authority is given by AU, attributes are represented using $\mathcal{A}$. A policy describing the access rights to the underlying plaintext, in Disjunctive normal form (DNF), is called an access structure $\mathbb{A}$ and is defined in the following subsection.

2.3 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 $\forall B, C : B \in \mathbb{A}, B \subseteq C \Rightarrow C \in \mathbb{A}$. An access structure (respectively *monotone access structure*) or policy is a collection (respectively, *monotone collection*) $\mathbb{A}$ of non-empty subsets of $\{P_1, P_2, \ldots, P_n\}$, i.e., $\mathbb{A} \subseteq 2^{\{P_1, P_2, \ldots, P_n\}} \setminus \{\varnothing\}$. The sets in $\mathbb{A}$ are called *authorized sets*, and the sets not in A are called *unauthorized sets*. In our protocol, the role of the parties is taken by the attributes. Thus, the access structure $\mathbb{A}$ in DNF will contain one or more authorized sets of attributes, i.e. the set of attributes necessary to decrypt a ciphertext, combined by a disjunction.

2.4 Complexity Assumptions

Our distributed ABE scheme is build on an asymmetric Type-3 pairing and therefore is based on the following complexity assumption, as defined by Chatterjee and Menezes (2011):

2.4.1 Discrete Logarithm in Type 3 (DLP-3)

Given $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$ and $y \in \mathbb{G}_T$, no probabilistic polynomial-time algorithm can compute $P \in \mathbb{G}_1$ so that $e(P, g_2) = y$ or $Q \in \mathbb{G}_2$ so that $e(g_1, Q) = y$.

2.4.2 Decisional Bilinear Diffie-Hellman in Type 3 (DBDH-3)

For an arbitrary group of exponents $\alpha, \beta, \gamma \in \mathbb{Z}_p^*$, given a tuple $(g_1, g_2, {g_1}^\alpha, {g_1}^\beta, {g_1}^\gamma, {g_2}^\beta, {g_2}^\gamma, \delta)$, no probabilistic polynomial-time algorithm can distinguish the following two tuples with a non-negligible advantage:

$$\left({g_1}^\alpha, {g_1}^\beta, {g_1}^\gamma, {g_2}^\beta, {g_2}^\gamma, e(g_1, g_2)^{\alpha \cdot \beta \cdot \gamma}\right)$$
$$\left({g_1}^\alpha, {g_1}^\beta, {g_1}^\gamma, {g_2}^\beta, {g_2}^\gamma, e(g_1, g_2)^{\delta}\right)$$

3 MODEL OF A BDABE

We now introduce our Blockchain-based Distributed Attribute Based Encryption (BDABE) protocol by describing the roles of entities and the handling of attributes and keys and their interaction in the protocol. Additionally, we detail our utilization of a distributed ledger in combination with CP-ABE. The construction of our distributed ABE scheme will be detailed in the section 4, while the protocol flow will be discussed in section 5.

3.1 Roles

There are five different entities involved in our BDABE model:

1. *Distributed Blockchain (BC)*: The Blockchain is the distributed ledger used to represent the current state of delegated access rights in the system. Permissions to interact with the Blockchain are handled by the Root Authority and the Attribute Authorities.
2. *Attribute Authority (*AU*)*: An AU is responsible for a set of users and a set of attributes, which we will call a domain. Each AU may register users in its domain and hand out the attribute keys of its domain to users. Besides creating users, attribute assignment is the main purpose of an AU. It may assign attributes to users outside its own domain, i.e. a user created by AU_x may receive attributes given out by AU_y. We assume that each AU is semi-trusted in our system, i.e. it might be curious about the value of a plaintext in the system, but has no intention of tampering with it.
3. *Root Authority (RA)*: A RA is a central component in our system. It is responsible for creating the secret keys of attribute authorities and handing them to an AU. The RA is capable of key escrow and therefore needs to be fully trusted by all system participants.

Table 1: Summary of BDABE keys.

Key	Description	Usage
PK	public key	Common ground and input for nearly all algorithms. Publicly available for everyone.
MK	master key	Creation of AUs by the RA.
SK_{AU}	secret key of attribute authority AU	Used by an AU to create users and attributes.
PK_a	public key of address a	Used to generate $SK_{\mathcal{A},a}$ by an AU.
SK_a	secret key of address a	Used to decrypt a ciphertext.
$PK_{\mathcal{A}}$	public key of attribute $\mathcal{A}$	Used to encrypt a plaintext with the given attribute. Publicly available.
$SK_{\mathcal{A},a}$	secret key of attribute $\mathcal{A}$ for address a	Used to decrypt a ciphertext with a policy using attribute $\mathcal{A}$ in combination with the SK_a of address a.

4. *Data Reader (DR)*: A DR is an authorized user who intends to access encrypted data. The DR registers with an AU and obtains one or more attribute sets. If one of the DR's attribute sets satisfies an access policy associated with a ciphertext, the DR will be able to decrypt and acquire the plaintext.
5. *Data Owner (DO)*: A DO is any user, even unauthorized, who owns data to be uploaded and shared. A DO defines an access policy $\mathbb{A}$ for the data, so that only the desired DRs with matching attribute sets are granted permission to decrypt and get access to the plaintext data.

3.2 Attributes, Keys and Wallets

An attribute in our system is a tuple consisting of an identifier AU of an Attribute Authority, an identifier $\mathcal{A}$ describing the attribute itself (an arbitrary string) and a set of elements from $\mathbb{G}_1$ and $\mathbb{G}_2$ that belongs to this attribute. We will use the shortcut $\mathsf{AU}_{\mathcal{A}}$ to denote the attribute authority, who issued $\mathcal{A}$.

In blockchain systems users often have a *wallet*, which serves as a keychain. A wallet in BDABE is owned by a DR and can hold multiple identities. An identity consists of an address a and an attribute set. DR obtains an identity by registering with an AU. The management of attributes of a DR is implemented by means of transactions to and from a *wallet* with at least one identity.

Together with a the secret keys SK_a the attributes can be used to decrypt ciphertexts. In order to prevent collusion between DR, every attribute set is uniquely bound to a secret key SK_a linked to the identity with address a in the wallet of a DR. Table 1 shows an overview of all utilized keys in our protocol.

3.3 Attribute Creation and Assignment

Every attribute in our system is issued by an AU. In order to introduce a new attribute into the system, the AU initially issues a new attribute on the blockchain to its own address in an arbitrary quantity and creates the new $PK_{\mathcal{A}}$ for it, by using the PK and its own private SK_{AU}.

The assignment and dismissal of an attribute is done via a transaction on the Blockchain. The $PK_{\mathcal{A}}$ must be publicly available for every DO in the system, meaning the AU should maintain a publicly available list of all issued attributes and their associated $PK_{\mathcal{A}}$. The AU is also allowed to remove attributes from the wallet of a DR in his domain and thus may revoke attributes from a DR. This is possible only, because the AU knows the RSA keypair of every DR in his domain.

3.4 Encryption and Decryption

Figure 1 shows an example workflow how encryption and decryption in our BDABE protocol. In order to

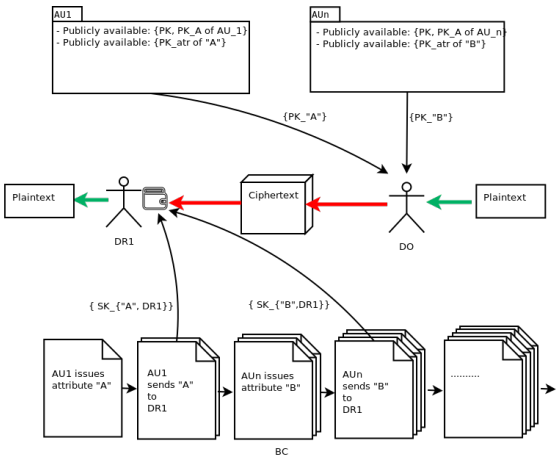

Figure 1: Workflow for encryption and decryption, according to an example $\mathbb{A} = A \wedge B$.

encrypt a plaintext, a DO needs the PK as well as all $PK_{\mathcal{A}}$s occuring in the DNF access policy $\mathbb{A}$. The resulting ciphertext may be decrypted by any registered DR with a sufficient authorized set of attributes, bound to his address a, matching any of the conjunctions in the disjunctive policy.

4 DISTRIBUTED ABE

We construct an efficient distributed Attribute Based Encryption scheme by expanding the scheme by Müller et al. (2008) and applying it on an asymmetric Type-3 pairing. Additionally we modified it according to our needs, as follows:

4.1 Setup

The *Setup* algorithm is run by the RA. It chooses bilinear groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order $p > 3$ and a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Next it chooses two generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, two random points $P_1 \in \mathbb{G}_1$ and $P_2 \in \mathbb{G}_2$, as well a random exponent $y \in \mathbb{Z}_p$ and a global hash function $H : \{0,1\}^* \to \mathbb{Z}_p$. The public key of the system is given by:

$$PK = \begin{Bmatrix} \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, H \\ e, g_1, g_2, P_1, P_2, \\ e(g_1, g_2)^y \end{Bmatrix}$$

The secret master key of the system is given by:

$$MK = \{y\}$$

4.2 CreateAuthority(PK, MK, a)

The algorithm *CreateAuthority* is also run by the RA. The algorithm chooses a value $r_{\mathsf{AU}} \in \mathbb{Z}_p$ uniformly at random. This value is used to randomize the result of the hash function H, modeled as random oracle for each authority. The MK is split into two components. This is done by selecting two random exponents $\alpha, \beta \in \mathbb{Z}_p$, such that $\alpha + \beta = y \equiv MK$. The randomization factor of the hash function r_{AU}, the address of the authority a together with two elements ${g_1}^{\alpha}$ and ${g_2}^{\beta}$, are used as the secret key of attribute authority AU:

$$SK_{\mathsf{AU}} = \begin{Bmatrix} SK_{\mathsf{AU},\mathrm{I}} = {g_1}^{\alpha}, SK_{\mathsf{AU},\mathrm{II}} = {g_2}^{\beta}, \\ SK_{\mathsf{AU},\mathrm{III}} = r_{\mathsf{AU}}, SK_{\mathsf{AU},\mathrm{IV}} = a \end{Bmatrix}$$

4.3 CreateUser(PK, SK_{AU}, a)

The algorithm *CreateUser*, run by the AU, chooses a secret value $r_a \in \mathbb{Z}_p$ and outputs the public user key of address a as:

$$PK_a = \{PK_{a,\mathrm{I}} = {g_1}^{r_a}, PK_{a,\mathrm{II}} = {g_2}^{r_a}, PK_{a,\mathrm{III}} = a\}$$

Using the secret key of attribute authority AU (SK_{AU}), the secret key of address a (SK_a) is computed as:

$$\begin{aligned} SK_a &= \{SK_{a,\mathrm{I}} = SK_{\mathsf{AU},\mathrm{I}} \cdot {P_1}^{r_a}, SK_{a,\mathrm{II}} = SK_{\mathsf{AU},\mathrm{II}} \cdot {P_2}^{r_a}\} \\ &= \left\{ {g_1}^{\alpha} \cdot {P_1}^{r_a}, {g_2}^{\beta} \cdot {P_2}^{r_a} \right\} \end{aligned}$$

4.4 RequestAttributePK(PK, SK_{AU}, $\mathcal{A}$)

The AU has to reach a decision which attributes are part of its domain. Once it decided that an attribute is part of its domain, it issues the asset to its own address. Additional units may be issued in the future by the same key, which signed the original issuance, i.e. by the AU. Furthermore, the AU has to decide if it is the exclusive authority for attribute $\mathcal{A}$. If other authorities should be able to issue the same attribute it must grant permissions to do so. If the AU has not authority over $\mathcal{A}$ (i.e. $\mathsf{AU}_{\mathcal{A}} \neq a$), the algorithm *RequestAttributePK* returns NULL. Otherwise, using SK_{AU}, it calculates an exponent $\varepsilon(\mathcal{A}, \mathsf{AU}) = H(\mathcal{A}) \cdot SK_{\mathsf{AU},\mathrm{III}} \cdot H(SK_{\mathsf{AU},\mathrm{IV}}) = H(\mathcal{A}) \cdot r_{\mathsf{AU}} \cdot H(a)$ and returns the public attribute key of $\mathcal{A}$, which consists of three parts:

$$PK_{\mathcal{A}} = \begin{Bmatrix} PK_{\mathcal{A},\mathrm{I}} = {g_1}^{\varepsilon(\mathcal{A},\mathsf{AU})}, \\ PK_{\mathcal{A},\mathrm{II}} = {g_2}^{\varepsilon(\mathcal{A},\mathsf{AU})}, \\ PK_{\mathcal{A},\mathrm{III}} = e(g_1, g_2)^{y \cdot \varepsilon(\mathcal{A},\mathsf{AU})} \end{Bmatrix}$$

This public attribute, once requested should be publicly available for everyone, although it can only be produced by the respective authority, because it requires the random factor r_{AU} from SK_{AU} as input.

4.5 RequestAttributeSK(PK, SK_{AU}, $\mathcal{A}$, PK_a)

When a DR requests the secret key for an attribute, the AU runs the *RequestAttributeSK* algorithm. After determining that the attribute $\mathcal{A}$ is handled by AU (e.g. by querying the blockchain if $\mathsf{AU}_{\mathcal{A}} = a$), the authority AU figures out, if the user with address a is eligible for the attribute $\mathcal{A}$. This decision must be taken by AU. If the user with address a is not eligible to hold $\mathcal{A}$, *RequestAttributeSK* returns NULL, otherwise *RequestAttributeSK* calculates the exponent $\varepsilon(\mathcal{A}, \mathsf{AU}) = H(\mathcal{A}) \cdot SK_{\mathsf{AU},\mathrm{III}} \cdot H(\mathsf{AU}) = H(\mathcal{A}) \cdot r_{\mathsf{AU}} \cdot H(\mathsf{AU})$ and returns the secret key of attribute $\mathcal{A}$ for address a ($SK_{\mathcal{A},a}$) on the fly:

$$SK_{\mathcal{A},a} = \left\{ \begin{array}{l} SK_{\mathcal{A},a,\mathrm{I}} = (PK_{a,\mathrm{I}})^{\varepsilon(\mathcal{A},\mathsf{AU})}, \\ SK_{\mathcal{A},a,\mathrm{II}} = (PK_{a,\mathrm{II}})^{\varepsilon(\mathcal{A},\mathsf{AU})} \end{array} \right\} = \left\{ {g_1}^{r_a \cdot \varepsilon(\mathcal{A},\mathsf{AU})}, {g_2}^{r_a \cdot \varepsilon(\mathcal{A},\mathsf{AU})} \right\}$$

4.6 Encrypt(PK, M, $\mathbb{A}$, $PK_{\mathcal{A}_1}, \ldots, PK_{\mathcal{A}_N}$)

The *Encrypt* algorithm is run by a DO whenever he wants to encrypt a plaintext. The access policy $\mathbb{A}$ must be written in Disjunctive normal form (DNF). A policy in DNF consists of n different sets of conjunctions $\{S_1, S_2, \ldots, S_n\}$. The encryption algorithm initially iterates over all sets $j = 1, 2, \ldots, n$ to generate a random value $r_j \in \mathbb{Z}_p$ for each conjunction of attributes. Afterwards the ciphertext CT_j is calculated as:

$$CT_j = \left\{ \begin{array}{l} E_{j,\mathrm{I}} = M \cdot \left(\prod_{\mathcal{A} \in S_j} PK_{\mathcal{A},\mathrm{III}} \right)^{r_j}, \\ E_{j,\mathrm{II}} = {P_1}^{r_j}, \\ E_{j,\mathrm{III}} = {P_2}^{r_j}, \\ E_{j,\mathrm{IV}} = \left(\prod_{\mathcal{A} \in S_j} PK_{\mathcal{A},\mathrm{I}} \right)^{r_j}, \\ E_{j,\mathrm{V}} = \left(\prod_{\mathcal{A} \in S_j} PK_{\mathcal{A},\mathrm{II}} \right)^{r_j} \end{array} \right\}$$

The resulting ciphertext CT is finally obtained as:

$$CT = \{CT_1, CT_2, \ldots, CT_n\}$$

4.7 Decrypt(PK, CT, $\mathbb{A}$, SK_a, $\{SK_{\mathcal{A}_1,a}, \ldots, SK_{\mathcal{A}_n,a}\}$)

The *Decrypt* algorithm is run by a DR whenever he wants to access a plaintext. To decrypt a ciphertext, *Decrypt* first checks if any conjunction in the given policy $\mathbb{A}$ can be satisfied by attribute set of address a: $\{SK_{\mathcal{A}_1,a}, \ldots, SK_{\mathcal{A}_n,a}\}$. If this is not the case, the algorithm outputs NULL. Otherwise it uses the set with the smallest amount of matching attributes to compute the plaintext message M as:

$$M = E_{j,\mathrm{I}} \frac{\mathrm{e}\left(E_{j,\mathrm{II}}, \prod_{i \in S_j} SK_{\mathcal{A},a,\mathrm{II}} \right) \cdot \mathrm{e}\left(E_{j,\mathrm{III}}, \prod_{i \in S_j} SK_{\mathcal{A},a,\mathrm{I}} \right)}{e\left(E_{j,\mathrm{IV}}, SK_{a,\mathrm{II}}\right) \cdot e\left(E_{j,\mathrm{V}}, SK_{a,\mathrm{I}}\right)}$$

In order to see that the decryption is correct, see Appendix 9.

5 BDABE AND THE BLOCKCHAIN

Most of the current blockchain solutions are designed with a crypto currency in mind. In our BDABE protocol, we define an authorization attribute rather than a Bitcoin. This attribute represents the access i.e. decryption rights defined by an Attribute Authority (AU), the creator of the transaction to its receiver.

BDABE provides several useful mechanisms using the blockchain. In BDABE, like in FairAccess, the blockchain is considered to be a database that stores all access control attributes in the form of transactions. Furthermore, it is able to provide a judicially usable proof of evidence of the access i.e. decryption rights of all users in the distributed system. Additionally the access right delegation to users by multiple authorities is tamper proof.

5.1 Integration of BDABE into the Blockchain

Our protocol consists of six fundamental protocol operations: $Setup_{\mathfrak{B}}$, $CreateAuthority_{\mathfrak{B}}$, $CreateUser_{\mathfrak{B}}$, $CreateAttribute_{\mathfrak{B}}$, $IssueAttribute_{\mathfrak{B}}$ and $RevokeAttribute_{\mathfrak{B}}$. The details of the $Encrypt_{\mathfrak{B}}$ and $Decrypt_{\mathfrak{B}}$ protocol operations are omitted here, as the retrieval of public key of attribute $\mathcal{A}$ ($PK_{\mathcal{A}}$) is carried out away from the blockchain and the reception of secret key of attribute $\mathcal{A}$ for address a ($SK_{\mathcal{A},a}$) to the wallet address a is included in the $IssueAttribute_{\mathfrak{B}}$ protocol operation. The protocol operations utilize algorithms of our distributed ABE scheme in combination with operations on the blockchain and and will be introduced now:

1. $Setup_{\mathfrak{B}}()$: The purpose of the initial setup protocol is to setup the BDABE protocol by spreading key material and by initializing the blockchain. This is done by setting up a distributed ABE scheme as well as a permissioned chain through the RA. The new chain is created by the RA on the blockchain, including mining the genesis block, and the scheme is setup by running the *Setup*() algorithm once. The *Setup* algorithm generates a public key (PK) and a master key (MK). The PK is made publicly available for everyone, while the MK needs to be kept secret and is only used to generate SK_{AU}s for AUs. Figure 2 gives an overview of the different roles and interactions during the setup of the system.
2. $CreateAuthority_{\mathfrak{B}}(\mathsf{AU})$: The master key ($MK$) is used by the RA to generate a new SK_{AU} for each and every AU in the system. The

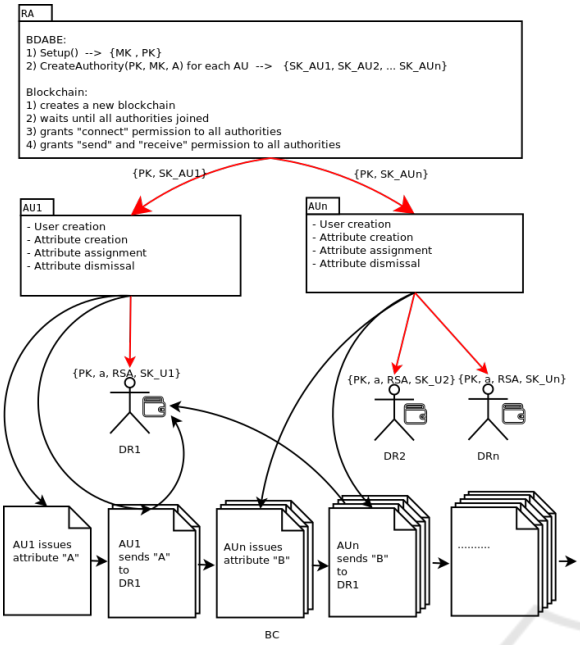

Figure 2: Creation and issuing of keys and attributes in the Blockchain-based Distributed Attribute Based Encryption protocol.

RA generates a new address and RSA keypair $\{a, RSA_{priv}, RSA_{pub}\}$ on the blockchain. The SK_{AU} is generated using address a as authority name. After giving the SK_{AU} the public key (PK) as well as the newly generated RSA keypair to all AUs in a secure manner, all AUs receive the "connect", "send", "receive" and "issue" permission by the RA. This can be seen in Figure 3. If an additional AU wants to join the system at a later date, this procedure needs to be repeated by the RA.

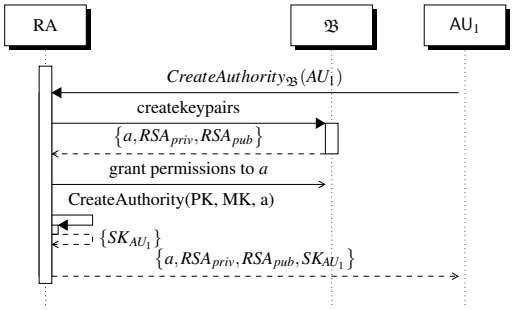

Figure 3: Protocol operation $CreateAuthority_{\mathfrak{B}}(AU_1)$.

3. $CreateUser_{\mathfrak{B}}(a)$: A user is created by an AU and can only be created by an AU because a SK_{AU} is needed as input to *CreateUser*. This means that an address a in combination with a linked RSA keypair, as well as the personal SK_a, linked to a user address a are created by an AU. This is done by first calling "createkeypairs" on the blockchain, which generates a new tuple $\{a, RSA_{priv}, RSA_{pub}\}$. The address a receives the "connect", "send" and "receive" permission by the AU. A new SK_a, linked to a user address a is created and finally the set $\{a, RSA_{priv}, RSA_{pub}, SK_a\}$ is given to the user in a secure manner. The RSA keypair is kept by the AU. This enables the AU to "revoke" attributes from address a later on. An overview of the protocol operation is given in Figure 4.

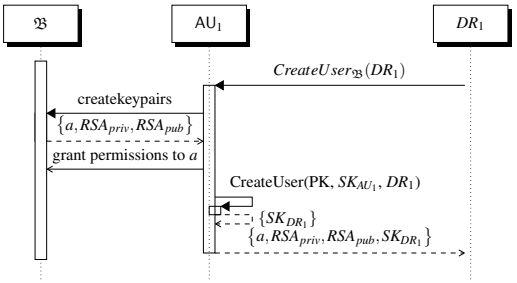

Figure 4: Protocol operation $CreateUser_{\mathfrak{B}}(DR_1)$.

4. $CreateAttribute_{\mathfrak{B}}(\mathsf{AU}, \mathcal{A})$: This operation is initialized by an Attribute Authority and creates an new attribute $\mathcal{A}$ in the belongings of an Attribute Authority. An AU gives out the $PK_{\mathcal{A}}$s of attributes on request and releases these public attribute keys available for everyone. Everyone in the system is allowed to request $PK_{\mathcal{A}}$s from the AU, as $PK_{\mathcal{A}}$s are needed by everyone during encryption. An asset representing the attribute string is created on the blockchain by issuing it to the address of the AU. Afterwards the AU must decide if other AUs should also be able to give out this attribute to users. If it decides to do so it should grant the "issue" permission to all the address of AUs that should be able to issue. In order to track the assignment of the attributes to users, the Attribute Authority "subscribes" to the issued attribute. By subscribing, the AU instructs its node to start tracking the issued attribute. This protocol operation can be seen in Figure 5.
5. $IssueAttribute_{\mathfrak{B}}(\mathsf{AU}, \mathcal{A}, a)$: This operation issues an attribute $\mathcal{A}$ from an Attribute Authority AU to a user with address a, if the user is eligible to own attribute $\mathcal{A}$. Attribute Authority has to decide if the user is eligible. The Attribute Authority sends the asset in a transaction on the blockchain, including the secret key of attribute $\mathcal{A}$ for address a in the metadata, to the address a in a quantity of "1". This protocol operation can be seen in Figure 6.
6. $RevokeAttribute_{\mathfrak{B}}(\mathcal{A}, a)$: This operation revokes an attribute $\mathcal{A}$ from an user with address a. The operation is executed by an AU and is possible only, because the AU still knows the RSA key-

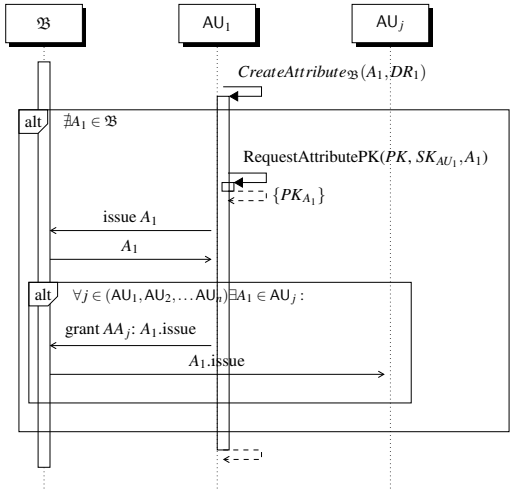

Figure 5: Protocol operation $CreateAttribute_{\mathfrak{B}}(A_1, AU_1)$ with AU_j being also able to issue A_1 in the future.

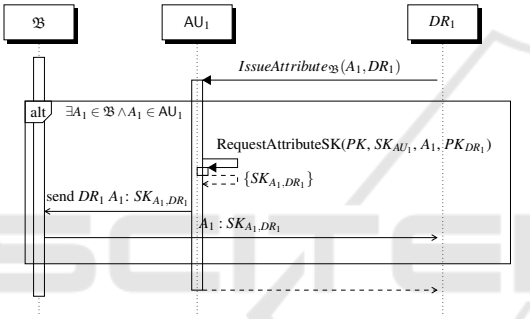

Figure 6: Protocol operation $IssueAttribute_{\mathfrak{B}}(A_1, DR_1)$.

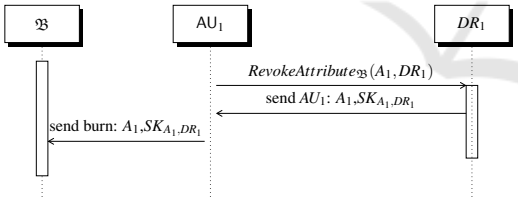

Figure 7: Protocol operation $RevokeAttribute_{\mathfrak{B}}(A_1, DR_1)$.

pair of a. The implications of this are staggering, as this means that the AU that created the DR initially is also responsible for the attributes in the DRs custody, even attributes issued by other AUs. The AU that created the SK_a for a is also responsible for the revocation of all attributes from that user.

6 SECURITY

We begin by establishing a security model for our distributed ABE scheme and conclude with a formal security analysis.

6.1 Security Model

We model the security of the distributed ABE scheme similar to the DABE security model of Müller et al. (2008) in the form of a security game. The challenger takes over the role of all attribute authorities and is responsible for the setup.

6.1.1 Setup

The challenger runs the *Setup* algorithm to obtain PK and MK and sends PK to the adversary.

6.1.2 Phase 1

The challenger creates empty dictionaries L_{Atb}, L_{Aut}, L_a and L_u. The adversary can ask the challenger for an arbitrary number of user and attribute keys with a couple of restrictions. On each query the challenger checks, if there exits an entry for AU_a in L_{Aut}. If not, it calls *CreateAuthority* and stores $(\mathsf{AU}_a, SK_{\mathsf{AU}})$ in L_{Aut}. It answers the adversary's queries as follows:

- User Key Query (a, AU_a): If there is an entry (a, AU) in L_a with $\mathsf{AU} \neq \mathsf{AU}_a$ return $\perp$. Otherwise, call *CreateUser*, store (a, AU_a) in L_a and return (PK_a, SK_a).
- Attribute Key Query ($\mathcal{A}$, $\mathsf{AU}_\mathcal{A}$, PK_a): If there is an entry $(\mathcal{A}, \mathsf{AU})$ in L_{Atb} with $\mathsf{AU} \neq \mathsf{AU}_\mathcal{A}$ return $\perp$. Otherwise, call *RequestAttributePK* and *RequestAttributeSK*, store $(\mathcal{A}, \mathsf{AU}_\mathcal{A})$ in L_{Atb}, update the entry for the user key PK_a in L_u by adding attribute $\mathcal{A}$ to it, and return $(PK_\mathcal{A}, SK_{\mathcal{A},a})$.

Users and attributes belong to a single authority, the challenger will reject requests for the same user or attribute key from different authorities. For a user key request the adversary sends the address a and an identifier for the authority AU_a responsible for a to the challenger. The challenger checks if L_a contains the address. If L_a does not contain the address, or the stored dictionary value is AU_a the challenger answers the request, otherwise he rejects it. If authority AU_a does not exist, the challenger calls *CreateAuthority*. To answer the request the challenger calls *CreateUser*, stores the tuple (a, AU_a) in L_a and sends the resulting private and public user keys to the adversary.

For an attribute request, the adversary sends the public user key PK_a, the authority identifier $\mathsf{AU}_\mathcal{A}$ responsible for the attribute and the attribute identifier $\mathcal{A}$ to the challenger. The challenger checks if L_{Atb} contains the attribute. If L_{Atb} does not contain the attribute, or the authority identifier stored in L_{Atb} is $\mathsf{AU}_\mathcal{A}$ the challenger answers the request, otherwise

he rejects it. If authority $\mathsf{AU}_{\mathcal{A}}$ does not exist, the challenger calls *CreateAuthority*. The challenger calls *RequestAttributePK* and *RequestAttributeSK*, stores the tuple $(a, \mathsf{AU}_{\mathcal{A}})$ in L_{Atb}, updates the entry for PK_a in L_u by adding the the attribute $\mathcal{A}$ and returns the public attribute key $PK_{\mathcal{A}}$ and the private attribute key $SK_{\mathcal{A},a}$ to the adversary.

6.1.3 Challenge

The adversary submits two messages M_0 and M_1 and an access policy $\mathbb{A}$ with the restriction that no user in L_u may satisfy $\mathbb{A}$. If any user in L_u has the attributes to satisfy $\mathbb{A}$, the challenger returns $\perp$. If the access policy $\mathbb{A}$ contains at least one attribute $\mathcal{A}$ for which there is no entry in L_{Atb}, the challenger returns $\perp$. The challenger flips a coin, b encrypts M_b under $\mathbb{A}$ and sends the ciphertext ct to the adversary.

6.1.4 Phase 2

Similar to Phase 1, the adversary can ask the challenger for an arbitrary number of user and attribute keys with the same restrictions. Additionally, the challenger will reject any attribute key query that would result in creating a user with an attribute set that satisfies $\mathbb{A}$. In this case the challenger sends $\perp$.

6.1.5 Guess

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

6.2 Security Analysis

In order to decrypt a ciphertext an adversary needs to find $e(g_1, g_2)^{y \cdot \varepsilon(j) \cdot r_j}$ for any $1 \leq j \leq n$, while $\varepsilon(j) = \sum_{\mathcal{A} \in S_j} H(\mathcal{A}) \cdot r_{\mathsf{AU}} \cdot H(a)$. This would allow an adversary to obtain M from $E_{j,\mathrm{I}}$ by computing a pairing of g^{γ} and $g^{y \cdot \delta}$ for some $\gamma, \delta \in \mathbb{Z}_p$, such that $\gamma \cdot \delta = \varepsilon(j) \cdot r_j$.

The adversary can only use keys that he has obtained before in a security game such as in subsection 6.1 to create such a pairing. We now show that an adversary is not able to compute this value, assuming the attributes of the adversary may not satisfy the access policy $\mathbb{A}$.

The value of g^y appears in the secret user key ($y = \alpha + \beta$), besides from the pairing in the PK and the MK itself. Thus the adversary has to use the SK_a to calculate the pairing for some λ:

$$e(\lambda \cdot SK_{a,\mathrm{I}}, g_2{}^{\gamma}) = e(g_1{}^{\alpha}, g_2{}^{\gamma}) \cdot e(P_1{}^{r_a}, g_2{}^{\gamma}) \cdot e(\lambda, g_2{}^{\gamma})$$

$$e(g_1{}^{\gamma}, \lambda \cdot SK_{a,\mathrm{II}}) = e(g_1{}^{\gamma}, g_2{}^{\beta}) \cdot e(g_1{}^{\gamma}, P_2{}^{r_a}) \cdot e(g_1{}^{\gamma}, \lambda)$$

Given all values that the adversary knows, the only useful choice for $g_1{}^{\gamma}$ and $g_2{}^{\gamma}$ are $E_{j,\mathrm{IV}}$ and $E_{j,\mathrm{V}}$ for any conjunction of attributes in a ciphertext. Pairing $E_{j,\mathrm{IV}}$ with $SK_{a,\mathrm{II}}$ gives:

$$e(E_{j,\mathrm{IV}}, SK_{a,\mathrm{II}}) = e(g_1, g_2)^{\beta \cdot \varepsilon(j) \cdot r_j} \cdot e(g_1, P_2)^{r_a \cdot \varepsilon(j) \cdot r_j}$$

While pairing $E_{j,\mathrm{V}}$ with $SK_{a,\mathrm{I}}$ gives:

$$e(E_{j,\mathrm{V}}, SK_{a,\mathrm{I}}) = e(g_1, g_2)^{\alpha \cdot \varepsilon(j) \cdot r_j} \cdot e(g_1, P_1)^{r_a \cdot \varepsilon(j) \cdot r_j}$$

To obtain $e(g_1, g_2)^{\alpha \cdot \varepsilon(j) \cdot r_j}$ and $e(g_1, g_2)^{\beta \cdot \varepsilon(j) \cdot r_j}$, the second factors of the previous mentioned pairing equations have to be eliminated each. Despite all three exponents of $e(g_1, P_1)^{r_a \cdot \varepsilon(j) \cdot r_j}$ and $e(g_1, P_2)^{r_a \cdot \varepsilon(j) \cdot r_j}$ being unknown to the adversary, he cannot compute the value of $y = \alpha + \beta$.

7 IMPLEMENTATION

The BDABE protocol was implemented using a pairing cryptography library written in pure Rust by the developers of z-cash, called "bn" library. It makes use of the Barreto-Naehrig (BN) curve construction of Ben-Sasson et al. (2014) to provide two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$, with an efficient bilinear pairing. The communication protocol between the distributed ABE scheme and the Multichain is implemented using its JSON-RPC API. The Hash function H is implemented using blake2b (cf. Aumasson et al. (2013)) in our solution. The source code to our BDABE protocol in rust, as well as the implementation of the JSON-RPC protocols, the DR *Decryption* algorithm and the DO *Encryption* algorithm will be made available publicly.

We developed the attribute backend using MultiChain by Greenspan (2015) as blockchain technology. MultiChain is a permissioned blockchain technology which is based on Bitcoin core. The developers of MultiChain have introduced additional features, such as support for asset transactions. This means that assets can be created on the blockchain and can be send between addresses. We used MultiChain as an inter organizational blockchain for exchanging attribute assets, in other words for mapping attributes from authorities to users.

8 DISCUSSION

In this section, we accentuate the implications of issuing, revocation and delegation regarding attributes

on a blockchain.

8.1 Issuing of Attributes

An attribute is introduced into the system by an AU through issuing it on the blockchain to itself. The attribute should be issued openly, meaning it should be possible that additional units might be issued in future by the same key which signed the original issuance. The outcome of this is that a specific attribute may only be issued by the AU that issued it initially. An AU may additionally issue an attribute to the address of another AU, but the sovereignty over the attribute assignment stays within the AU.

8.2 Evaluation of the Target Function

By subscribing to an issued attribute, any participant in the blockchain network is able to track the transactions of a specific attribute.[1] In order to lookup which addresses are able to decrypt a specific access policy $\mathbb{A}$, a users needs to subscribe to all attributes in the access policy. By listing all transactions recipients and by combining them with mathematical set operations, each user is able to evaluate the target function of an access policy live. As access policies in our protocol are given in DNF, i.e. consisting of n different sets of conjunctions $\{S_1, S_2, \ldots, S_n\}$, while each conjunction consists of an arbitrary number of up to m attributes $S_j = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m\}$, the list of recipients can be evaluated live using:

$$recipients = \bigcup_{j=1}^{n} \left(\bigcap_{k=1}^{m} listassettransactions(\mathcal{A}_k)_a \right)$$

With $listassettransactions_a$ beeing the multichain function that lists all asset transactions for the asset $\mathcal{A}$ with recipient address a.

8.3 Revocation of Attributes

Specific attributes of users may only be revoked by the AU that created the user in the first place. Attributes of other AU may also only be removed by the AU that created the user key. This is because the AU that initially created the user still owns its RSA keypair. Altogether this means that a user may mix attributes from different AUs, but only the one AU that the user registered with is in the responsibility of conducting revocations.

The revocation mechanism via the blockchain is not cryptographically forward secure, as a user might export an attribute key he received via his wallet and use it later on to decrypt ciphertexts besides the blockchain. This could be done by rewriting the decryption software, to include attributes off the blockchain. This issue is currently open and will be addressed in a future work.

[1] By using multichains rescan option the node will reindex all transactions from when the assets was created.

8.4 Delegation of Attributes

In a BDABE scenario with separate algorithms for the creation of users and secret keys for attributes tied to users, the delegation of attributes between users cannot be supported since it would lead to collusions. In order to prevent the delegation of an attribute by a users, the attribute he receives via a transaction on the blockchain is specifically tied to his secret user key. The attribute alone, without an according user key is useless.

8.5 Performance of the Framework

The performance of the protocol operations mostly depends on the utilized blockchain. In a multichain environment, with default settings, the delay for confirming transactions is in average about one minute. This leads to attributes not being assigned instantly, but with a delay of about a minute in average. This is not fast enough to support real time access decisions, but by way of comparison fast enough for example for decentralized authentication systems in web based scenarios.

9 CONCLUSION AND FUTURE WORK

In this paper, we proposed the concept of a Blockchain-based Distributed Attribute Based Encryption (BDABE) protocol as a way to apply an adaption of the DABE scheme by Müller et al. (2008) to the real-world, that supports an arbitrary number of distributed attribute authorities. It allows the generation of users by an attribute authority. Furthermore it allows the creation and deletion of new attributes dynamically at any time by a transaction on the blockchain. We provided an efficient construction of the distributed ABE scheme and showed how the most fundamental operations can be integrated into the blockchain for efficient key and attribute management. The integration of a forward secure revocation mechanism is left open for future development.

ACKNOWLEDGMENTS

This work has been funded by the Fraunhofer Cluster of Excellence "Cognitive Internet Technologies" http://www.cit.fraunhofer.de

REFERENCES

Agrawal, S. and Chase, M. (2017). Fame: Fast attribute-based message encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 665–682. ACM.

Aumasson, J.-P., Neves, S., Wilcox-O'Hearn, Z., and Winnerlein, C. (2013). Blake2: simpler, smaller, fast as md5. In *International Conference on Applied Cryptography and Network Security*, pages 119–135. Springer.

Ben-Sasson, E., Chiesa, A., Tromer, E., and Virza, M. (2014). Succinct non-interactive zero knowledge for a von neumann architecture. In *USENIX Security Symposium*, pages 781–796.

Bethencourt, J., Sahai, A., and Waters, B. (2007). Ciphertext-policy attribute-based encryption. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 321–334. IEEE.

Boneh, D. and Franklin, M. (2001). Identity-based encryption from the weil pairing. In *Annual international cryptology conference*, pages 213–229. Springer.

Chase, M. (2007). Multi-authority attribute based encryption. In *Theory of Cryptography Conference*, pages 515–534. Springer.

Chatterjee, S. and Menezes, A. (2011). On cryptographic protocols employing asymmetric pairings-the role of ψ revisited. *Discrete Applied Mathematics*, 159(13):1311–1322.

Galbraith, S. D., Paterson, K. G., and Smart, N. P. (2008). Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121.

Goyal, V., Pandey, O., Sahai, A., and Waters, B. (2006). Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm.

Greenspan, G. (2015). Multichain private blockchain, white paper. *URl: http://www. multichain. com/download/MultiChain-White-Paper. pdf*.

Jahid, S., Mittal, P., and Borisov, N. (2011). Easier: Encryption-based access control in social networks with efficient revocation. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 411–415. ACM.

Jemel, M. and Serhrouchni, A. (2017). Decentralized access control mechanism with temporal dimension based on blockchain. In *2017 IEEE 14th International Conference on e-Business Engineering (ICEBE)*, pages 177–182. IEEE.

Lewko, A. and Waters, B. (2011). Decentralizing attribute-based encryption. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 568–588. Springer.

Lin, G., Hong, H., and Sun, Z. (2017). A collaborative key management protocol in ciphertext policy attribute-based encryption for cloud data sharing. *IEEE Access*, 5:9464–9475.

Müller, S., Katzenbeisser, S., and Eckert, C. (2008). Distributed attribute-based encryption. In *International Conference on Information Security and Cryptology*, pages 20–36. Springer.

Ouaddah, A., Abou Elkalam, A., and Ait Ouahman, A. (2016). Fairaccess: a new blockchain-based access control framework for the internet of things. *Security and Communication Networks*, 9(18):5943–5964.

Ruj, S., Nayak, A., and Stojmenovic, I. (2011). Dacc: Distributed access control in clouds. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pages 91–98. IEEE.

Sarhan, A. Y. and Carr, S. (2017). A highly-secure self-protection data scheme in clouds using active data bundles and agent-based secure multi-party computation. In *Cyber Security and Cloud Computing (CSCloud), 2017 IEEE 4th International Conference on*, pages 228–236. IEEE.

Yang, K. and Jia, X. (2014). Dac-macs: Effective data access control for multi-authority cloud storage systems. In *Security for Cloud Storage Systems*, pages 59–83. Springer.

APPENDIX

Decryption of an BDABE Ciphertext

In order to see that the decryption is correct, let $\varepsilon(j) = \sum_{\mathcal{A} \in S_j} H(\mathcal{A}) \cdot r_{\mathsf{AU}} \cdot H(a)$, then $E_{j,\mathrm{I}} = M \cdot e(g_1, g_2)^{y \cdot r_j \cdot \varepsilon(j)}$, $E_{j,\mathrm{IV}} = {g_1}^{r_j \cdot \varepsilon(j)}$, $E_{j,\mathrm{V}} = {g_2}^{r_j \cdot \varepsilon(j)}$ and thus:

$$E_{j,\mathrm{I}} \frac{e\left(E_{j,\mathrm{II}}, \prod_{i \in S_j} SK_{\mathcal{A},U,\mathrm{II}}\right) \cdot e\left(E_{j,\mathrm{III}}, \prod_{i \in S_j} SK_{\mathcal{A},U,\mathrm{I}}\right)}{e\left(E_{j,\mathrm{IV}}, SK_{a,\mathrm{II}}\right) \cdot e\left(E_{j,\mathrm{V}}, SK_{a,\mathrm{I}}\right)}$$

$$= M \cdot e(g_1, g_2)^{y \cdot r_j \cdot \varepsilon(j)} \cdot \frac{e(P_1{}^{r_j}, g_2{}^{r_a \cdot \varepsilon(j)}) \cdot e(g_1{}^{r_a \cdot \varepsilon(j)}, P_2{}^{r_j})}{e(g_1{}^{r_j \cdot \varepsilon(j)}, g_2{}^{\beta} \cdot P_2{}^{r_a}) \cdot e(g_1{}^{\alpha} \cdot P_1{}^{r_a}, g_2{}^{r_j \cdot \varepsilon(j)})}$$

$$= M \cdot e(g_1, g_2)^{y \cdot r_j \cdot \varepsilon(j)} \cdot \frac{e(P_1, g_2)^{r_j \cdot r_a \cdot \varepsilon(j)} \cdot e(g_1, P_2)^{\cdot r_j \cdot r_a \cdot \varepsilon(j)}}{e(g_1, g_2)^{\beta \cdot r_j \cdot \varepsilon(j)} \cdot e(g_1, P_2)^{r_j \cdot r_a \cdot \varepsilon(j)} \cdot e(g_1, g_2)^{\alpha \cdot r_j \cdot \varepsilon(j)} \cdot e(P_1, g_2)^{r_j \cdot r_a \cdot \varepsilon(j)}}$$

$$= M \cdot \frac{e(g_1{}^{r_j \cdot \varepsilon(j)}, g_2{}^{y})}{e(g_1{}^{r_j \cdot \varepsilon(j)}, g_2{}^{\alpha+\beta})}$$

$= M$, exactly when $\alpha + \beta = y \equiv MK$