Attribute-based encryption with encryption and decryption outsourcing

Muhammad Asim
Eindhoven University of Technology

Milan Petkovic
Philips Research Eindhoven, Netherlands

Tanya Ignatenko
Eindhoven University of Technology

DOI: 10.4225/75/57b65cc3343d0

12th Australian Information Security Management Conference. Held on the 1-3 December, 2014 at Edith Cowan University, Joondalup Campus, Perth, Western Australia.

ATTRIBUTE-BASED ENCRYPTION WITH ENCRYPTION AND DECRYPTION OUTSOURCING[1]

Muhammad Asim,[1,2] Milan Petković,[1,2] Tanya Ignatenko[1]
[1]Eindhoven University of Technology, [2]Philips Research Eindhoven, Netherlands
muhammad.asim@philips.com, milan.petkovic@philips.com, t.ignatenko@tue.nl

Abstract

In this paper we propose a new scheme for ciphertext-policy attribute-based encryption that allows outsourcing of computationally expensive encryption and decryption steps. The scheme constitutes an important building block for mobile applications where both the host and users use mobile devices with limited computational power. In the proposed scheme, during encryption the host involves a semi-trusted proxy to encrypt a partially encrypted (by the host) message according to an access policy provided by the host. The proxy is unable to learn the message from this partially encrypted text. A user can only decrypt the stored ciphertext if he possesses secret keys associated with a set of attributes that satisfies the associated policy. To reduce computational load in the decryption step, the user, in his turn, involves a semi-trusted proxy (e.g. a cloud) by deploying the scheme of Green et al. (2011). The cloud is given a transformation key that facilitates construction of an El Gamal-ciphertext from the original ciphertext if the user's attributes satisfy the ciphertext. This El Gamal-ciphertext can be then efficiently decrypted on the user's resource-constrained device. The resulting ABE scheme with encryption and decryption outsourcing is proven to be secure in the generic group model.

Keywords

Proxy Re-encryption, Attribute-Based Encryption, Access Policy, Outsourcing

INTRODUCTION

Nowadays we observe the spread of distributed systems in which sensitive data has to be shared with multiple parties. In order to facilitate these distributed systems, a network infrastructure can be used to allow shared storage (e.g. in the cloud) and access to the systems' resources. Although this new paradigm offers a number of advantages by eliminating organizational boundaries and increasing operational flexibility, it requires extra security mechanisms needed to protect sensitive data involved. Unlike in traditional situations where one party encrypts a message for another targeted party, distributed (collaborative) systems require more flexibility in the way access to their resources is regulated, allowing access to parties that satisfy an access policy rather than to a specified set of parties.

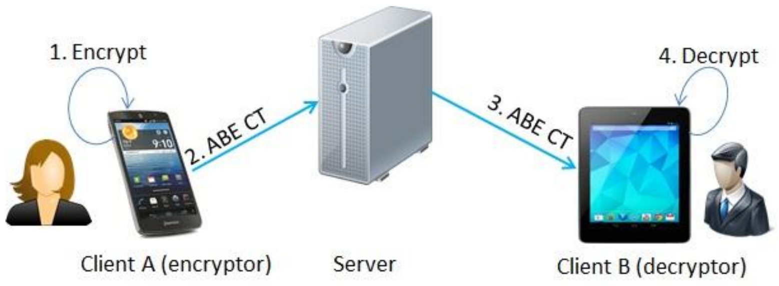

Figure 1 Architecture for a typical ABE

Attribute-based encryption (ABE), introduced by Sahai and Waters (2005), offers an expressive way to define asymmetric-key encryption schemes for policy enforcement based on attributes. Here both a user secret key and ciphertext are associated with sets of attributes. There are two flavours of ABE defined, i.e. ciphertext-policy attribute-based encryption (CP-ABE) and key-policy attribute-based encryption (KP-ABE). In CP-ABE, see e.g. Bethencourt et al. (2007), Ostrovsky et al. (2007), and Ibraimi et al. (2010), a user encrypts the data according to a predicate (access policy) defined over attributes, such that only the party that possesses a secret key associated

[1] This work has been partially funded by the EC via grant agreement no. 611659 for the AU2EU project.

with the attribute set satisfying the predicate is able to decrypt the ciphertext, *Figure 1*. In KP-ABE of Goyal et al. (2008) the idea is reversed. Here the ciphertext is associated with the attribute set and the secret key is associated with the predicate defined over the attributes. CP-ABE schemes are more desirable due to their flexibility allowing encryption according to the access policy.

In practice, in distributed systems data often has to be encrypted, transferred or accessed from portable devices, such as e.g. smartphones. However, ABE schemes are typically computationally intensive, involving pairing operations and exponentiations, and their computational complexity increases linearly with the size of the access control policy (or number of attributes). Currently, an efficient realization of ABE schemes can be implemented using conventional desktop computers. Portable devices, however, have limited computational resources making it a challenging task to realize mobile applications with ABE schemes. Recently, Green et al. (2011) proposed architecture with a corresponding modified ABE scheme that allowed reducing the computational load required for ciphertext decryption on mobile devices by involving a semi-trusted server in the decryption process. In Green et al. (2011) the ABE scheme is modified such that a user has to provide the server with a transformation key that allows the server to translate any ABE ciphertext satisfied by the user's attributes into an El Gamal-style ciphertext, without being able to learn any part of the encrypted message. The resulting El Gamal-style ciphertext is then transmitted to the user, who can decrypt it in a computationally efficient way. This work, however, does not address computational load reduction for the host that creates the access policy and the ciphertext.

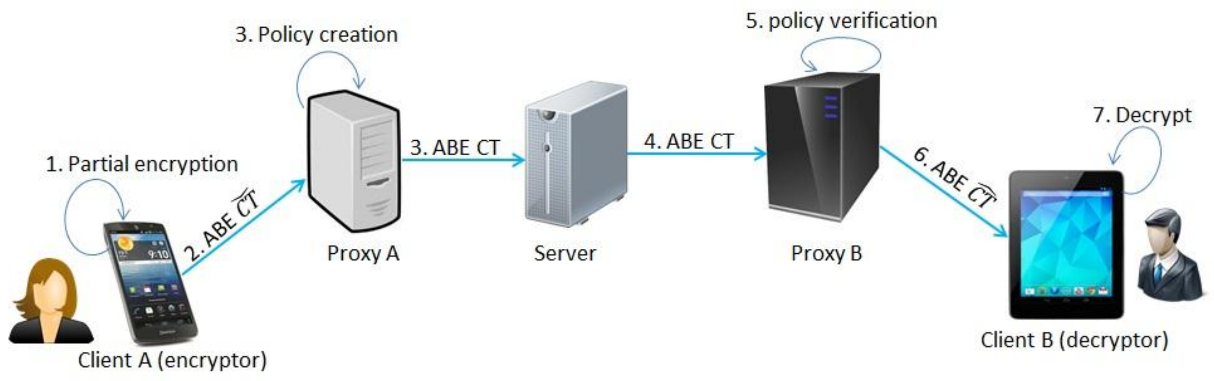

Figure 2 Architecture for ABE with Encryption and Decryption Outsourcing

Our Contribution: In this paper we propose a new attribute-based encryption scheme with encryption and decryption (end-to-end) outsourcing that reduces the computational load for both the host and the user. Unlike the original ABE scheme or the scheme of Green et al. (2011), our scheme involves two proxies, as shown in *Figure 2*. Here we allow the host (data owner) to outsource cryptographic policy creation to a semi-trusted entity or proxy (*Proxy A*), and to encrypt messages for users according to the given policy in such a way that the proxy is a) unable to learn the encrypted message; and b) is enforced to encrypt the messages based on the attributes specified by the policy. The decryption workload is reduced by allowing a user to outsource the policy verification to another semi-trusted proxy (*Proxy B*) by allowing the proxy to verify the policy given the user's transformation key attributes. The latter is realized with deploying the idea of Green et al. (2011). The security of the proposed ABE scheme with encryption and decryption outsourcing is proven in the generic group model.

Paper Organization: The remainder of this paper is organized as follows. In Background section we provide generic concepts used in the proposed scheme. ABE with encryption and decryption outsourcing section presents the formal definition of our scheme and its security model. The next section describes the construction of the proposed scheme. Finally, the last section concludes the paper.

BACKGROUND

We start with defining a number of concepts that provide the basis for our scheme.

Bilinear Groups

Our proposed scheme is based on pairings over groups of prime order. Let $\mathbb{G}_0$ and $\mathbb{G}_1$ be two multiplicative cyclic groups of prime order p, g be a generator of $\mathbb{G}_0$, and Z_p be the additive group associated with integers from $\{0, \dots, p-1\}$. A pairing or bilinear map $e: \mathbb{G}_0 \times \mathbb{G}_0 \rightarrow \mathbb{G}_1$ satisfies the following properties:

1. Bilinearity: for all $u, v \in \mathbb{G}_0$ and $a, b \in \mathbb{Z}_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$.

2. Non-degeneracy: $e(g, g) \neq 1$.

Observe that bilinear map also enjoys the symmetry property, i.e. $e(g^a, g^b) = e(g, g)^{ab} = e(g^b, g^a)$.
Group $\mathbb{G}_0$ is said to be a *bilinear group* if the group operation in $\mathbb{G}_0$ and the bilinear map $e: \mathbb{G}_0 \times \mathbb{G}_0 \rightarrow \mathbb{G}_1$ can be computed efficiently.

Access Tree

Another important concept used in this paper is the concept of an access tree. Let τ be an access tree associated with an access policy. A leaf node K in the access tree τ represents an attribute from the attribute set $\omega \in \Omega$, where Ω is a universe of attributes. A non-leaf node k in τ represents a threshold gate, which is described by its child nodes and a threshold value. Let num_k be the number of children of a node k and T_k its threshold value, then $0 < \mathrm{T}_k \leq \mathrm{num}_k$. If $\mathrm{T}_k = 1$, then k corresponds to an OR gate; if $\mathrm{T}_k = \mathrm{num}_k$, the node k is an AND gate. For leaf nodes, $\mathrm{T}_K = 1$.

Next we define the following functions on access trees. Denote by $\mathrm{parent}(k)$ the function that returns the parent of a node k in an access tree τ. Moreover, we define an ordering between children of a certain node in τ in the following manner. Let the children nodes of k be numbered from 1 to num_k, then $\mathrm{index}(\kappa)$ returns the order value associated with a child node κ. Finally, function $\mathrm{att}(K)$ returns the attribute associated with a leaf node K of the access tree τ.

Shamir's Secret Sharing Scheme

The idea of Shamir's secret sharing is to divide a secret D into n shares $D_1, D_2, \dots, D_n$ such that knowing k or more secret shares one can easily recover D, while the knowledge of only $k-1$ or less shares of D does not suffices to reconstruct D. Shamir's secret sharing is information-theoretically secure. Schemes based on Shamir's secret sharing are also known as threshold secret sharing schemes and can be realized using polynomial interpolation.

Let $y = f(x)$ denote a polynomial of degree $k-1$. The scheme then consists of two steps: 1) distribution of shares – where secret $\alpha_0 = D$ is distributed among users U_i by giving each user one of k distinct points (shares) of a random polynomial f(x), which is created using α_0 and $k-1$ randomly selected coefficients $\alpha_1, \cdots, \alpha_{k-1}$; 2) reconstruction of secret D, where any group of k or more users can reconstruct key D by combining their distinct shares and reconstructing the polynomial f(x) using e.g. Lagrange interpolation. Then the secret is given by $f(0)$.

ABE WITH ENCRYPTION AND DECRYPTION OUTSOURCING

Next we define algorithms that constitute our proposed ABE scheme with encryption and decryption outsourcing capability and present the security game for it.

Algorithms of the Proposed ABE Scheme

In the proposed scheme the host and the user can outsource part of encryption and decryption functionalities to two independent semi-trusted entities, called proxies. We call the proxy used to outsource encryption *proxy A* and the one for decryption *proxy B* (see also Figure 2). Our proposed scheme uses the following algorithms.

- $\mathbf{Setup}(\lambda)$**:** This algorithm is run by a trusted authority. It takes as input a security parameter λ, and outputs public parameters *PK* and a master secret key *MK*.

- $\mathbf{KeyGeneration}_d(\mathrm{MK}, \mathrm{PK}, \omega)$**:** This algorithm is also run by the trusted authority. It takes as input the public parameters *PK*, the master secret key *MK* and a set of user's attributes ω. The output of this step is the secret key SK_ω for a user with the attribute set ω. Here SK_ω is composed of two parts, i.e. $\mathrm{SK}_{\mathrm{pdec}}$ and $\mathrm{SK}_{\mathrm{udec}}$, where $\mathrm{SK}_{\mathrm{pdec}}$ can be used by *proxy B* to assist in decryption, while $\mathrm{SK}_{\mathrm{udec}}$ is used directly by the user to recover a plain message M from the partially decrypted ciphertext $\widehat{\mathrm{CT}}$ constructed by *proxy B*.

- $\mathbf{KeyGeneration}_e(\mathrm{MK}, \mathrm{PK})$**:** This algorithm is used to generate a unique (per proxy) secret key for *proxy A* that assists the host with encryption by constructing the access policy. The algorithm takes as input the master secret key *MK* and public parameters *PK*. The output of this algorithm is $\mathrm{SK}_{\mathrm{penc}}$.

- **Encryption**$(\mathrm{PK}, \mathrm{M}, \tau)$**:** The encryption algorithm takes as input the public parameters *PK*, a message *M*, and an access tree τ over a universe of attributes Ω. It produces a partial ciphertext $\breve{\mathrm{CT}}$**.** This partial ciphertext includes the access tree (structure) τ, but no cryptographic access policy associated with τ.

- ***PolicyCreation***$(\breve{\mathrm{CT}}, \mathbf{SK_{penc}})$**:** This algorithm is run by *proxy A* in order to create the final ciphertext *CT*. It takes as input the partial ciphertext $\breve{CT}$ and the proxy encryption secret key $\mathrm{SK_{penc}}$ and creates the cryptographic access policy related to the access tree τ.

  Note that the proxy secret key $\mathrm{SK_{penc}}$ does not suffice to recover the message *M*, and the proxy is only trusted with cryptographic access policy creation.

- **PolicyVerification**$(\mathrm{SK_{pdec}}, \mathrm{CT})$**:** This algorithm is run by *proxy B* that takes as input the proxy decryption key $\mathrm{SK_{pdec}}$ related to SK_ω and the ciphertext CT. The output of this algorithm is a partially decrypted ciphertext $\widehat{\mathrm{CT}}$ (called El Gamal style ciphertext) if ω satisfies access tree τ.

- **Decryption**$(\mathrm{SK_{udec}}, \widehat{\mathrm{CT}})$: The user runs the decryption algorithm. The decryption algorithm takes as input the partially decrypted ciphertext $\widehat{CT}$ and a user's secret key $\mathrm{SK_{udec}}$ (called El Gamal style private key). The output of this stage is the decrypted message M if ω satisfies τ, otherwise the output is an error, denoted by $\perp$.

Security Model for ABE with Encryption and Decryption Outsourcing

Now we define two security games for an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ for the ABE scheme with encryption and decryption outsourcing capability (ABE-EDO). In the first game, $\mathcal{A}$ asks for the secret keys of *proxy A*, i.e. $\mathrm{SK_{penc}}$, while in the second game, $\mathcal{A}$ asks for the secret keys of the user, i.e. SK_ω. These games correspond to the assumption we use in our model that *proxy A* and *proxy B* are independent of each other and will not collude. The idea behind this assumption is to achieve a good separation of duties.

Game 1

Setup: The challenger $\mathcal{C}$ runs *Setup* algorithm and gives the adversary $\mathcal{A}$ the public parameters, while keeping the master secret key to itself.

Phase 1: The adversary $\mathcal{A}$ performs a polynomially bounded number of queries asking for secret keys $\mathrm{SK_{penc_i}}, \forall \mathrm{i} \in \{1,2,\dots,\mathrm{Q}\}$ of *proxy A*. The challenger returns these secret keys to $\mathcal{A}$.

Challenge: In this phase the adversary $\mathcal{A}$ submits two equal length plaintexts M_0 and M_1 from the message space, on which $\mathcal{A}$ wants to be challenged. The challenger flips a random coin $\mathrm{b} \in \{0,1\}$ and returns the partial encryption of $\mathrm{M_b}$ (i.e. the encryption that does not contain the cryptographic policy) to the adversary $\mathcal{A}$.

Phase 2: Repeat Phase 1 querying for the secret keys that have not already been queried for in Phase 1.

Guess: In this phase, $\mathcal{A}$ outputs a guess $\mathrm{b}' \in \{0,1\}$ and wins if $\mathrm{b}' = \mathrm{b}$. The advantage of the adversary in attacking the scheme is $|\Pr[\mathrm{b}' = \mathrm{b}] - \frac{1}{2}|$.

Game 2

Setup: The challenger $\mathcal{C}$ runs *Setup* algorithm and gives the adversary $\mathcal{A}$ the public parameters, while keeping the master secret key to itself.

Phase 1: The adversary $\mathcal{A}$ performs a polynomially bounded number of queries and asks for the user secret keys corresponding to the attribute sets $\omega_1, \omega_2, \dots, \omega_\mathrm{Q}$. The challenger returns the secret keys $\mathrm{SK}_{\omega_\mathrm{i}}, \forall \mathrm{i} \in \{1,2,\dots,\mathrm{Q}\}$ to $\mathcal{A}$.

Challenge: In this phase the adversary $\mathcal{A}$ submits two equal length plaintexts M_0 and M_1 from the message space, on which $\mathcal{A}$ wants to be challenged. Moreover, $\mathcal{A}$ also gives the challenger an access tree τ^* such that the queried secret keys from Phase 1 do not satisfy τ^*. The challenger flips a random coin $\mathrm{b} \in \{0,1\}$ and returns the encryption of $\mathrm{M_b}$ under τ^* to the adversary $\mathcal{A}$.

Phase 2: Repeat Phase 1 querying for the secret keys that do not satisfy τ^* and that have not already been queried for in Phase 1.

Guess: In this phase, $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$ and wins if $b' = b$. The advantage of the adversary in attacking the scheme is $|\Pr[b' = b] - \frac{1}{2}|$.

Definition 1. An ABE-EDO scheme is secure if all polynomial time adversaries have at most negligible advantage in the aforementioned ABE-EDO security games, where the advantage is defined to be $|\Pr[b' = b] - \frac{1}{2}|$.

Theorem 1. Let q be an upper bound on the total number of group elements that an adversary $\mathcal{A}$ can receive from queries she makes to the challenger $\mathcal{C}$ for elements from the hash function $H(\cdot)$, group $\mathbb{G}_0$, $\mathbb{G}_1$, bilinear map $e(\cdot,\cdot)$, and from her interaction in the ABE-EDO security games. The advantage of the adversary in the security game is $O(q^2/p)$.

The security of *ABE-EDO* scheme can be proved using arguments similar to those in Shoup (1997), Bethencourt et al (2007), or Boneh et al. (2005).

CONSTRUCTION OF THE PROPOSED SCHEME

In this section we present the formal description of the algorithms for the ABE scheme with encryption and decryption outsourcing capabilities. First, however, we outline its main idea.

In our scheme we assume that the host and the user can involve two independent semi-trusted entities (proxies) to outsource computationally expensive encryption and decryption operations. During encryption a message M will be encrypted according to an access policy τ such that only a user whose attribute set ω satisfies τ will be able to decrypt it. The resulting ciphertext consists of the encrypted message and cryptographic policy components. Since policy creation is computationally expensive, this step is performed by involving a proxy (*Proxy A* in Figure 2). This proxy is only trusted with policy creation. Therefore the host performs partial encryption and gives to the proxy the information that consists of encrypted message, encrypted secret key for the policy creation and a set of encrypted authorized attributes. In this way the proxy cannot learn the message and is forced to create the access policy for the specified attributes. During decryption, a user will involve the second proxy to perform computationally expensive pairing operations required to evaluate the access policy. The user will provide the proxy (*Proxy B* in Figure 2) with a transformation key that allows the proxy to evaluate the access policy and produce a partially decrypted ciphertext (El Gamal type ciphertext), if the user's attributes satisfy the access policy. The proxy is not able to learn the message using only ciphertext and the key provided by the user. Note that our assumption and trust model explicitly excludes the collusion between that *Proxy B* and *Proxy A*, otherwise it would be able to decrypt the message. Finally, the resulting El Gamal type ciphertext can be efficiently decrypted by the user.

Let $e: \mathbb{G}_0 \times \mathbb{G}_0 \to \mathbb{G}_1$ denote the bilinear map, defined in Background section. A security parameter λ determines the size of the groups. Moreover, let $H_1: \{0,1\}^* \to \mathbb{G}_0$ and $H_2: \mathbb{G}_1 \to \{0,1\}^n$ be two collision resistant hash functions, where $\{0,1\}^*$ denotes a binary sequence of an arbitrary length. Finally, we define the Lagrange coefficient $\triangle_{v,\Omega}(V) = \prod_{v' \in \Omega, v' \neq v} \frac{V - v'}{v - v'}$, for $V, v \in \mathbb{Z}_p$ and Ω being a set of elements from $\mathbb{Z}_p$.

1. ***Setup***(λ)**:** This algorithm is run by a trusted authority in order to generate system parameters, i.e. the public key PK and master secret key MK. The algorithm selects a random generator $g \in \mathbb{G}_0$ of prime order p and random variables $\alpha, \beta, \gamma, \theta, \vartheta \in \mathbb{Z}_p$. In addition, it selects cryptographic hash functions $H_1: \{0,1\}^* \to \mathbb{G}_0$ and $H_2: \mathbb{G}_1 \to \{0,1\}^n$, and sets $A = e(g,g)^{\alpha\beta}$. The public key PK and master secret key MK are set to be:

$$\begin{aligned} PK &= \left(\mathbb{G}_0, \mathbb{G}_1, g, g^{\beta}, g^{\vartheta}, A, H_1, H_2\right), \\ MK &= (g^{\alpha}, \beta, \gamma, \theta, \vartheta). \end{aligned}$$

2. **KeyGeneration**$_d(MK, PK, \omega)$**:** This algorithm is run by the trusted authority in order to generate the secret key for a user with an attribute set $\omega \in \Omega$. The algorithm selects random variables $r_u, z \in Z_p$ and sets SK_{ω} to be:

$$SK_{\omega} = (SK_{pdec}, SK_{udec}),$$

with

$$SK_{pdec} = \Big(\quad D^{(1)} = \left(g^{\alpha} g^{\gamma\theta r_u}\right)^{\frac{1}{z}} = g^{\frac{\alpha}{z}} g^{\gamma\theta t},$$

$$D^{(2)} = \left(g^{\gamma\beta r_u}\right)^{\frac{1}{z}} = g^{\gamma\beta t},$$
$$\forall a_j \in \omega: D_j^{(3)} = \left(H_1(a_j)^{\gamma r_u}\right)^{\frac{1}{z}} = H_1(a_j)^{\gamma t}),$$

where $t = \frac{r_u}{z}$, and

$$SK_{udec} = z.$$

Here SK_{udec} is a short El Gamal type private key for the user, while SK_{pdec} is a transformation key that can be shared with a semi-trusted *proxy B* assisting the user with computationally expensive policy evaluation.

3. **KeyGeneration**$_e(MK, PK)$**:** This algorithm generates the secret key for *proxy A*. The proxy will use this key to create the access policy specified by the host:

$$SK_{penc} = \frac{\theta}{\vartheta}.$$

This key is used by *proxy A* to recover the component to be shared across the access structure τ in order to compute the cryptographic access policy.

Remark: If there are a number of proxies in the system, as a result of this algorithm, each proxy gets a unique key SK_{penc} (i.e. unique ϑ will be used).

4. ***Encryption*** (PK, M, τ): This algorithm is run by the host (encryptor). To reduce computational load on the host, the algorithm only produces a partial ciphertext, that consists of encrypted message $M \in \mathbb{G}_1$, encrypted secret key and the set of authorized attributes. These latter components are used by *Proxy A* to which calculation of cryptographic policy is outsourced. The algorithm selects a random value $s \in Z_p$ and computes a partial ciphertext as follow:

$$\widetilde{CT} = \left(C = M \oplus H_2(A^s), \check{C} = g^{\beta s}, \tilde{C} = g^{\vartheta s}, \forall\, a_j \in \tau: \widetilde{C_j} = H_1(a_j)^{-s}\right).$$

5. ***PolicyCreation***$\left(\widetilde{CT}, \mathbf{SK_{penc}}\right)$: This algorithm is run by *proxy A* in order to generate access policy $F(\omega)$ related to the access tree τ that has to be associated with the ciphertext C. The proxy performs the following steps:

    a) Decrypt $\tilde{C}$:

    $$C' = \tilde{C}^{SK_{penc}} = g^{\theta s} = g^{\mathfrak{s}},$$

    where $\mathfrak{s} = \theta s$.

    b) Create the access policy $F(\omega)$:

    In this step, the proxy creates cryptographic components related to the access policy τ. It uses Shamir's secret sharing to distribute $g^{\mathfrak{s}}$ among the leaf nodes of τ. More precisely, the algorithm chooses a polynomial $q_k(\cdot)$ for each node k in τ in a top-down manner, starting from the root node R. First, for each node k in the tree, it sets the degree d_k of the polynomial $q_k(\cdot)$ to be one less than the threshold value T_k of that node, i.e. $d_k = T_k - 1$. Then, starting with the root node R, the algorithm sets $q_R(0) = C'$ and selects at random d_R other points of the polynomial $q_R(\cdot)$ in order to define the polynomial completely. For any other node k, the algorithm sets $q_k(0) = q_{parent(k)}(index(k))$ and selects the rest d_k points randomly to completely define $q_k(\cdot)$.

    c) Create ciphertext:

    The final ciphertext CT is composed as follows:

    $$CT = \left(C, \check{C}, \forall a_j \in \tau: C_j = g^{\mathfrak{s}_j} H_1(a_j)^{-s}\right).$$

6. $\boldsymbol{Policy\,Verification}(SK_{pdec}, CT)$: This algorithm is run by *proxy B*. It takes as input the transformation secret key $\mathrm{SK_{pdec}}$, provided by a user and associated with a set of his attributes ω, and a ciphertext CT. The algorithm verifies a user's attribute set ω satisfies the access tree τ, and if so computes and outputs a partially decrypted El Gamal style ciphertext $\widehat{\mathrm{CT}}$.

The algorithm makes use of the recursive function $\mathrm{DecryptNode}(\mathrm{CT}, \mathrm{D}^{(3)}_{j,\omega}, \mathrm{k})$. We define this function for (a) leaf nodes K, and for (b) internal nodes k of τ.

a) $\mathrm{DecryptNode}(\mathrm{CT}, \mathrm{SK_{pdec}}, \mathrm{K})$:
Observe that each leaf node of the access tree is associated with a real-valued attribute. Then let $\mathrm{j} = \mathrm{att}(\mathrm{K})$. Now, if $\mathrm{j} \in \omega$, we have

$$\begin{aligned}\mathrm{DecryptNode}(\mathrm{CT}, \mathrm{SK_{pdec}}, \mathrm{K}) &= e(\mathrm{C_j}, \mathrm{D}^{(2)}) \cdot e(\check{\mathrm{C}}, \mathrm{D}_j^{(3)}) \\ &= e(g^{\$_j}\mathrm{H_1}(a_j)^{-s}, g^{\gamma\beta t}) \cdot e(g^{\beta s}, \mathrm{H_1}(a_j)^{\gamma t}) \\ &= e(g,g)^{\$_j\gamma\beta t} \cdot e(\mathrm{H_1}(a_j), g)^{-s\gamma\beta t} \cdot e\left(g, \mathrm{H_1}(a_j)\right)^{s\gamma\beta t} \\ &= e(g,g)^{\$_j\gamma\beta t}.\end{aligned}$$

If $\mathrm{j} \notin \omega$, then $\mathrm{DecryptNode}(\mathrm{CT}, \mathrm{SK_{i,\omega}}, \mathrm{K}) = \perp$, where ⊥ denotes failure.

b) $\mathrm{DecryptNode}(\mathrm{CT}, \mathrm{SK_{pdec}}, \mathrm{k})$:
For all nodes κ that are children of k, the algorithm calls $\mathrm{DecryptNode}(\mathrm{CT}, \mathrm{SK_{pdec}}, \kappa)$. Its output stored as F_κ is used to determine whether the user has enough attributes to satisfy the policy. Note that to satisfy the policy, there should be enough data points (i.e. satisfied child nodes) to reconstruct the polynomial in the node k and thus to reconstruct $q_k(0)$. Let Ω_k be an arbitrary $\mathrm{T_k}$-sized set of child nodes κ such that $\mathrm{F}_\kappa \neq \perp, \forall \kappa \in \Omega_k$. If there exists no such a set, then node k is not satisfied and the function returns ⊥. Otherwise, using polynomial interpolation, the algorithm evaluates the following function:

$$\begin{aligned}\mathrm{F_k} &= \prod_{\kappa \in \Omega_k} \mathrm{F}_\kappa^{\Delta_{v,\Omega_k}(0)}, \textit{where } \mathrm{v} = \mathrm{index}(\kappa) \\ &= \prod_{\kappa \in \Omega_k} \left(e(g,g)^{\$_j\beta\gamma t}\right)^{\Delta_{v,\Omega_k}(0)} \\ &= e(g,g)^{\$_k\gamma\beta t}.\end{aligned}$$

Partial decryption at the proxy: *Proxy B* has to verify if the user satisfies the access control policy associated with τ, and to create an El Gamal ciphertext (partially decrypted ciphertext) that can be further decrypted by the user. The proxy first evaluates the $\mathrm{DecryptNode}(\cdot)$ function on the root node R of τ. If $\mathrm{DecryptNode}(\mathrm{CT}, \mathrm{SK_{pdec}}, \mathrm{R})$ returns ⊥, then τ is not satisfied by the attribute set ω associated with the key $\mathrm{SK_{pdec}}$ and thus the user's secret key SK_ω. In this case, decryption fails and the algorithm returns ⊥. Otherwise, if τ is satisfied, the decryption algorithm performs the following steps:

$$\begin{aligned}\mathrm{Z}^{(1)} &= \mathrm{DecryptNode}(\mathrm{CT}, \mathrm{SK_{pdec}}, \mathrm{R}) = e(g,g)^{\$\gamma\beta t} = e(g,g)^{\theta\gamma\beta st}, \\ \mathrm{Z}^{(2)} &= e(\check{\mathrm{C}}, \mathrm{D}^{(1)}) = e\left(g^{\beta s}, g^{\frac{\alpha}{z}} g^{\gamma\theta t}\right) = e(g,g)^{\frac{\alpha\beta s}{z}} \cdot e(g,g)^{\theta\gamma\beta st}, \\ \mathrm{Z}^{(3)} &= \frac{\mathrm{Z}^{(2)}}{\mathrm{Z}^{(1)}} = \frac{e(g,g)^{\frac{\alpha\beta s}{z}} \cdot e(g,g)^{\theta\gamma\beta st}}{e(g,g)^{\theta\gamma\beta st}} = e(g,g)^{\frac{\alpha\beta s}{z}},\end{aligned}$$

and outputs $\widehat{\mathrm{CT}} = (\mathrm{C}, \mathrm{Z}^{(3)})$.

7. **Decryption**$(\mathrm{SK_{udec}}, \widehat{\mathrm{CT}})$: This algorithm is run by the user. It takes as input the El Gamal style private key $\mathrm{SK_{udec}}$ and the partially decrypted ciphertext $\widehat{\mathrm{CT}}$. The plain message can now be recovered as follows

$$\begin{aligned}\mathrm{M} &= \mathrm{M} \oplus \mathrm{H_2}(\mathrm{A}^s) \oplus \mathrm{H_2}\left((\mathrm{Z}^{(3)})^z\right) \\ &= \mathrm{M} \oplus \mathrm{H_2}(\mathrm{A}^s) \oplus \mathrm{H_2}\left(e(g,g)^{\frac{\alpha\beta s}{z} \cdot z}\right) \\ &= \mathrm{M}.\end{aligned}$$

CONCLUSIONS AND FUTURE WORK

In this paper we have presented a new ABE scheme with encryption and decryption outsourcing capabilities. The scheme relies on the use of two semi-trusted proxies, one used to outsource computationally expensive encryption steps and another to outsource decryption steps. During the encryption process, a host involves the encryption proxy to create cryptographic policy components for a set of specified attributes, in such a way that the proxy cannot reveal the original message and is enforced to use the given attributes. During decryption, the decryption proxy is used for policy evaluation. Upon successful evaluation (i.e. a user possesses an authorized set of attributes), the proxy transforms the original ciphertext into the El Gamal type of ciphertext. The latter can be then efficiently decrypted by the user. To guarantee security of the scheme, two proxies used in our scheme have to be independent and non-colluding. The security of our scheme is proved in the generic group model. The presented scheme plays an important role in applications where both the host and users are using computationally constrained devices (e.g. mobile devices).

REFERENCES

Bethencourt, J., Sahai, A., Waters, B. (2007). Ciphertext-policy attribute-based encryption. In: Proc. of Security and Privacy 2007, pp.321–334. IEEE Computer Society

Boneh, D., Boyen, X., Goh, E.J. (2005). Hierarchical identity based encryption with constant size ciphertext. In: Proc. of EUROCRYPT’05, pp. 440–456. LNCS 3494, Springer

Boneh, D., Franklin, M.K. (2003). Identity-based encryption from the weil pairing. SIAM J. Comput. 32(3), pp.586–615

Goyal, V., Jain, A., Pandey, O., Sahai, A. (2008). Bounded ciphertext policy attribute based encryption. In: ICALP (2), pp. 579–591. LNCS 5126, Springer

Green, M., Hohenberger, S., Waters, B. (2011). Outsourcing the decryption of ABE ciphertexts. In: Proc. of the 20th USENIX Conference on Security (SEC'11), pp.34

Ibraimi, L., Asim, M., Petkovic, M. (2010). An encryption scheme for a secure policy updating. In: Proc. of SECRYPT’10, pp. 399–408. SciTePress

Ostrovsky, R., Sahai, A., Waters, B. (2007). Attribute-based encryption with non-monotonic access structures. In: Proc. of CCS’07, pp. 195–203. ACM

Sahai, A., Waters, B. (2005). Fuzzy identity-based encryption. In: Proc. of EUROCRYPT’05, pp.457–473. LNCS 3494, Springer

Shamir, A. (1979). How to share a secret. Communications of the ACM 22(11), pp.612–613

Shoup, V. (1997). Lower bounds for discrete logarithms and related problems. In: Proc. of EUROCRYPT’97, pp. 256–266