Attribute Based Group Signature Scheme

Abstract. Alice needs a document signed by an employee in Bob's company. That employee should be part of the IT staff and is at least a junior manager in the cryptography team or a senior manager in the biometrics team. In such a scenario we need an Attribute Based Group Signature Scheme (ABGS). In this paper we define the first ABGS scheme where verifying includes authenticating a person that belongs to a certain group and owns particular attributes. We define two security notions adopted from group signature which are: traceability and anonymity. We prove our scheme to be secure under those two notions.

1 Introduction

Alice wants a document to be signed by any employee in Bob's company. Alice requires that employee to have certain attributes such as being part of the IT staff and is at least a junior manager in the cryptography team or a senior manager in the biometrics team. We derive our idea from the Group Signature Scheme.
Group signature schemes was first proposed by Chaum and van Heist [7] for implementation in e-cash systems. These schemes have been called for by numerous practical applications to facilitate the scenario of a member signing on behalf of the entire group. The two underlying notions of such schemes are anonymity and traceability. We say a scheme is anonymous if we can not figure out which member of the group signed the message. However, we say a scheme is traceable if we could ensure that we could trace all signatures to a signer or a member of a forging coalition. The mechanism of compromising between these two otherwise mutually exclusive features is what makes group signatures a focal point of research. Bellare et al [2] defined the standard mode for the scheme to be both anonymous and traceable. The authors introduced strong, formal definitions for the requirements of the two security notions. They showed how you could have a scheme that is fully traceable by the group manager, yet fully anonymous to any other verifier.
Efficiency was another important aspect for implementing group signature. In the schemes presented in [8] [5] the length of the keys and/or the signature were depending on the number of people in the group which made it unsuitable for large groups until Camenisch et al. proposed their scheme in [6]. Their scheme had constant sized sigantures and constant sized public keys. Further more, the computational effort of the scheme is independent from the number of group members.
In our scheme we extend group signatures in order to allow any member of the group who satisfies certain properties to sign on behalf of the others. We adopt the idea of an attribute tree from Goyal et al's work in [9]. Consider an attribute

tree in which each interior node of the tree is a threshold gate and the leaves are linked with attributes. A threshold gate represents the number m of n children branching from the current node need to be satisfied in order to say that the parent node is satisfied. Once we reach the leaf we say it is satisfied if and only if it is owned by the signer. (For example, we represent a tree with AND and OR gates by using respectively 2 of 2 and 1 of 2 threshold gates.) The scheme has each public key labeled with an attribute tree and embeds in each signature an attribute set that the signer owns. In order for a signer to claim ownership of an attribute, the trusted third party has to give him an extra element to add to his private key. So if the trusted third party is the company which employed the signer it gives him a private key that contains elements to authenticate himself and his properties (such as what position he works in and/or which department) if requested by the verifier.
So going back to the main scenario Alice decides on an attribute tree and sends it to both a key generator and Bob's company. Key generator sends Alice a verifying key and a member in Bob's company sends her a signature. Alice could verify now Bob's signature and whether he satisfies her attribute tree or not.
In this paper we first discuss some preliminaries that will be used in the scheme and the proof of it's security. In section[3] we define an ABGS scheme and it's security notions. We then construct a scheme in section[4] and prove it to be secure in section[5]. Finally we conclude our results and bring up some open issues.

2 Preliminaries

In this section we will explain some of the preliminaries that were either used in constructing an ABGS scheme or in proving it's security.

2.1 The Strong Diffie-Hellman Assumption

This section defines q-Strong Diffie-Hellman and states Boneh-Boyen Lemma which are two concepts that will be used in section[5.2] to prove traceability of the construct scheme. Let G_1, G_2 be cyclic groups of prime order p, where possibly $G_1 = G_2$. Assuming the generators $g_1 \in G_1$, and $g_2 \in G_2$ consider the following [3]:

Definition 1. (q-Strong Diffie-Hellman Problem) *The q-SDH problem in (G_1, G_2) is defined as follows: given a $(q+2)$ tuple $(g_1, g_2, g_2^{\gamma}, g_2^{\gamma^2}, ..., g_2^{\gamma^q})$ as an input, output a pair $(g^{1/(\gamma+x)}, x)$ where $x \in Z_p^*$. An algorithm A has an advantage ε in solving q-SDH in (G_1, G_2) if: $Pr[A(g_1, g_2, g_2^{\gamma}, g_2^{\gamma^2}, ..., g_2^{\gamma^q}) = (g^{1/(\gamma+x)}, x)] \geq \varepsilon$, where the probability is over a random choice of a generator g_2 (with $g_1 \leftarrow \psi(g_2)$), of $\gamma \in Z_p^*$ and of random bits of A [3].*

This problem is considered hard to solve in polynomial time and ε should be negligble.

Theorem 1. (Boneh-Boyen SDH Equivalence) *Given a q-SDH instance* $(\grave{g}_1, \grave{g}_2, \grave{g}_2^{\gamma}, \grave{g}_2^{\gamma^2}, ..., \grave{g}_2^{\gamma^q})$*, and then applying the Boneh and Boyen's Lemma found in [3] we could obtain* $g_1 \in G_1, g_2 \in G_2, w = g_2^{\gamma}$ *and* $(q-1)$ *SDH pairs* (A_i, x_i) *such that* $e(A_i, wg_2^{x_i}) = e(g_1, g_2)$ *for each* i*. Any SDH pair besides these* $(q-1)$ *ones can be transformed into a solution to the original q-SDH instance [3].*

2.2 Linear Encryption

In this section we will define an encryption scheme which depends on the difficulty of the Decision Linear Diffie-Hellman Assumption [4]. This scheme will be used in the construction of our ABGS scheme and will lead to ensuring anonymity(See Section[5.3]) of the scheme.

Definition 2. (Decision Linear Problem in G_1) *Let* G_1 *be a group of prime order* p *and* u, v, h *are generators in that group. Given* $u, v, h, u^a, v^b, h^c \in G_1$ *as an input,it is hard to decide whether or not* $a + b = c$ *[4].*

Definition 3. (Linear Encryption Scheme) *In the Linear Encryption scheme a user's public key is* $u, v, h \in G_1$ *[4]. The private key is the exponents* $\xi_1, \xi_2 \in Z_p$ *such that* $u^{\xi_1} = v^{\xi_2} = h$*. To encrypt a messsage* M *choose random elements* $\alpha, \beta \in Z_p$ *and output the triple* $\langle C_1, C_2, C_3 \rangle = \langle u^{\alpha}, v^{\beta}, M.h^{\alpha+\beta} \rangle$*. To decrypt compute* $C_3/(C_1^{\xi_1} C_2^{\xi_2})$*. LE has been proven to be semantically secure.*

2.3 Lagrange Interpolation

Lagrange, in Numerical Analysis, is a way of interpolating a polynomial. In this paper it will be used in order to get the public key for the ABGS scheme(See Section[4])

Theorem 2. *Given* $n + 1$ *points-*$(x_i, f(x_i))$ *on a polynomial* f *of degree* n *we could identify the polynomial uniquely by calculating:*
$f(x) = \sum_{i=1}^{n} (f(x_i)(\Pi_{1 \leq k \neq i \leq n} (x - x_k)/(x_j - x_k)))$

2.4 Forking Lemma

D.Pointcheval et al, developed the forking lemma technique in constructing a proof of security in their digital signature scheme [12]. We will be using it in proving our scheme to be traceable(See Section[5.2]). Assume any signature scheme produces the triple $\langle \sigma_1, h, \sigma_2 \rangle$ where σ_1 takes its values randomly from a set,h is the result of hashing the message M together with σ_1, and σ_2 depends only on (σ_1, h, M). The Forking Lemma is as follows [12]:

Theorem 3. (The Forking Lemma) *Let* A *be a Probabilistic Polynomial Time Turing machine, given only the public data as input. If* A *can find, with non-negligible probability, a valid signature* $(M, \sigma_1, h, \sigma_2)$ *then, with non-negligible probability, a replay of this machine, with the same random tape and a different oracle, outputs two valid signatures* $(M, \sigma_1, h, \sigma_2)$ *and* $(M, \sigma_1, \grave{h}, \grave{\sigma}_2)$ *such that* $h \neq \grave{h}$*.*

2.5 Heavy Row Lemma

In this section we define Boolean Matrix and then a Heavy Row in that matrix [11]. Those definitions will be used for the Heavy Row Lemma [11] which will be used in proving traceability of our scheme together with the Forking lemma(See Section[5.2]).

Definition 4. (Boolean Matrix of Random Tapes) *Consider a hypothetical matrix M whose rows consists of all possible random choices of an adversary and the columns consist of all possible random choices of a challenger. Let each entry be either $\perp$ when adversary fails or $\top$ if adversary manages to win the game.*

Definition 5. (Heavy Row) *A row in M is called heavy if the fraction of $\top$ along the row is less than $\varepsilon/2$ where ε is the advantage of adversary succeeding in attack.*

Lemma 1. Heavy Row Lemma *Let M be a boolean matrix, given any entry that equal $\top$, then the probability that it lies in a heavy row is at least 1/2.*

3 ABGS Scheme

In this section we will first define our scheme and its security notions. Later on in the section we construct an implementable scheme(See Section[4]).

3.1 General Definition of the ABGS scheme

In an ABGS scheme there are five algorithms: Setup, KeyGen, Sign, Verify and Open. The following is a general description of each of the algorithms.

- Setup: Setup is a randomized algorithm. It takes no input. It generates a set of parameters S_{para} that will be used in the Key Generation algorithm and a tracing key $gmsk$ that will be used in the Open algorithm.
- KeyGen(S_{para}, n): KeyGen is an algorithm that takes the parameters of the setup and a number n that defines the number of users. It then generates public keys for attribute trees gpk, and private keys for users gsk. Private keys are created using a private key bases gsk_{base} and a set of attributes that the user i owns.
- Sign($gpk, gsk[i], M$): Given a public key of an attribute tree, a private key of a user i and a message. Output a signature σ
- Verify(gpk, M, σ, ζ): Given a message, a public key of a certain attribute tree, a signature and a set ζ that describes the set of attribute that satisfy the tree; Output either an acceptance or a rejection for the signature.
- Open($S_{para}, gmsk, M, \sigma$): Given a signature on a particular message, a public key and the tracing key. Trace to the signer i even if it is a member in forging coalition.

3.2 General Security Notions of the ABGS scheme

An ABGS scheme should be proved to be correct, anonymous and traceable. In this section we give a general definition for each property. We start with the definition of correctness.

Definition 6. (ABGS Scheme is Correct:) *We say an ABGS Scheme is correct if and only if honestly-generated signatures verify and trace correctly.*

For defining anonymity we introduce this game between an adversary *Adam* and a *Challenger*. The game demonstrates how with the access to a signature oracle an adversary should not be able to distinguish between signers unless they have unique attributes that identify them. The game consists of six phases: Init, Setup, Phase1, Challenge, Phase2, and finally Guess. A detailed desciption about the phases is described below:

- **Init:***Adam* chooses the attribute tree he would like to be challenged upon.
- **Setup:** *Challenger* runs the setup algorithm and keygen. *Challenger* produces a public key for the attribute tree and n private key bases gpk_{bases} that will be used in the signature oracle later in the game.
- **Phase 1:***Challenger* runs the signature oracle. *Adam* issues a certain number of queries to that oracle. *Adam* sends in every query a message M, index of user i and a set of attributes ζ that satisfy the tree. *Challenger* responds back with a signature σ.
- **Challenge:** *Adam* decides when to request his challenge. He sends the *Challenger* two indices (i_0, i_1), a message M and ζ. The triple $\langle i_0, M, \zeta\rangle$ and $\langle i_1, M, \zeta\rangle$ should not have been queried before in Phase 1 and should not be queried after this point in Phase 2. *Challenger* replies back with a signature σ_b where $b \in \{0, 1\}$.
- **Phase 2:** Phase two is exactly the same as phase one.
- **Guess:***Adam* tries to guess $\grave{b} \in \{0, 1\}$. If $b = \grave{b}$, *Adam* wins otherwise he fails.

We refer to an adversary like *Adam* as the selective anonymity attack (SAA) adversary and we define the advantage of attacking the scheme as $Adv_{SAA} = Pr[b = \grave{b}] - 1/2$.

Definition 7. (Selective Anonymity:) *We say a sheme is secure under a SAA attack if for any polynomial time SAA-Adversary advantage in winning the game is negligible. That is $Adv_{SAA} < \varepsilon$ where ε is negligible.*

For defining traceablity, we need to prove all signatures even the ones created by the collusion of multiple users trace to a member of the forging coalition. In order to do so we define the following game between an adversary *Adam* and the *Challenger*:

- **Init:***Adam* chooses the attribute tree he would like to be challenged upon.

- **Setup:** *Challenger* runs the setup algorithm and part of the keygen. *Challenger* produces a public key for the attribute tree and n private key bases that will be used in the signature oracle and private key oracle later in the game.
- **Querying a Signature/Private key Oracle:***Challenger* runs two oracles, a signature oracle and a private key oracle. *Adam* issues a number of queries to both oracles. He sends in every query to the signature oracle a message M, index of user i and a set of attributes ζ that satisfy the tree. *Challenger* responds back with a signature σ. When querying the private key oracle *Adam* sends an index and a set of attributes ζ. *Challenger* responds back with a valid private key.
- **Output:**If *Adam* is successful it outputs a forged signature σ that *Challenger* fails to trace using the open algorithm. Otherwise *Adam* fails.

We call an attack similiar to *Adam*'s a Forging Signature Attack (FSA). We represent the advantage of the adversary in winning the attack as Adv_{FSA}.

Definition 8. (ABGS Scheme is Traceable:) *We say a scheme is secure under a FSA attack if for any polynomial time the advantage of an adversary winning the game is negligible. That is* $Adv_{FSA} < \varepsilon$ *where* ε *is negligible.*

4 Construction of an ABGS Scheme

In this section we construct an ABGS scheme based on Boneh et al's work in [4].

- Setup: Consider a bilinear pair (G_1, G_2) with a computable isomorphism ψ. Suppose that SDH assumption holds on (G_1, G_2) and the linear assumption holds on G_1. Define the bilinear map $\hat{e} : G_1 \times G_2 \to G_T$. All three groups G_1, G_2, G_T are multiplicative and of a prime order p. Select a hash function $H : \{0,1\}^* \to Z_p$. Select a generator $g_2 \in G_2$ at random and then set $g_1 \leftarrow \psi(g_2)$. Select $h \in G_1$ and ξ_1, ξ_2 randomly from Z_p. $gmsk = \langle \xi_1, \xi_2 \rangle$ will be used later in the open algorithm. Set $u, v \in G_1$ such that $u^{\xi_1} = v^{\xi_2} = h$. Select a random γ from Z_p and set $w = g_2^{\gamma}$. Define a universe of attributes $U = \{1, 2, ..., m\}$ and for each attribute $j \in U$ choose a number t_j at random from Z_p. Let $S_{para} = \langle G_1, G_2, G_T, \hat{e}, H, g_1, g_2, h, u, v, gmsk, \gamma, w \rangle$.
- KeyGen(S_{para}, n): This algorithm generates a public key for a specific access structure and a private key for each user. Using γ generate for each user $i, 1 \leq i \leq n$ an SDH pair (A_i, x_i). Get x_i randomly from Z_p^* and $A_i = g_1^{1/(\gamma + x_i)} \in G_1$. For every attribute j that user i owns calculate $T_{i,j} = g_1^{t_j/(\gamma + x_i)}$. The private key for a user i will be the tuple $gsk[i] = \langle A_i, x_i, T_{i,1} ..., T_{i,\mu} \rangle$, where μ is the number of attributes a user owns. We consider the bases of the private key $gsk[i]_{base}$ to be equal to the pair $\langle A_i, x_i \rangle$. To generate a public key for a certain attribute tree we will need to choose a polynomial q_{node} of degree $d_{node} = k_{node} - 1$ for each node in the tree. That is done in top-down manner. Starting from the root $q_{root}(0) = \gamma$ and

other points in the polynomial will be random. The other nodes we set $q_{node}(0) = q_{parent}(index(node))$ and choose the rest of the points of the polynomial randomly. Once all polynomials have been decided the public key for a certain structure will be $gpk = \langle g_1, g_2, h, u, v, w, D_{leaf_1}, ..., D_{leaf_\mu}, h_1, ..., h_\mu \rangle$ where $D_{leaf_j} = g_2^{q_{leaf_j}(0)/t_{leaf_j}}$, $h_j = h^{t_j}$, μ is the number of leafs and $1 \leq j \leq \mu$.

- Sign$(gpk, gsk[i], M)$: For signing user i needs to choose $\alpha, \beta \in Z_p$ and compute the linear encryption of A_i and $T_{i,j}$ where $1 \leq j \leq \mu$. The ciphertext of the encryption will equal $C_1 = u^\alpha, C_2 = v^\beta, C_3 = A_i h^{\alpha+\beta}, CT_j = T_{i,j} h_j^{\alpha+\beta}$. Let $\delta_1 = x_i\alpha, \delta_2 = x_i\beta$. Choose randomly $r_\alpha, r_\beta, r_x, r_{\delta_1}$ and r_{δ_2}. Calculate $R_1 = u^{r_\alpha}$, $R_2 = v^{r_\beta}$, $R_3 = \hat{e}(C_3, g_2)^{r_x} \hat{e}(h, w)^{-r_\alpha - r_\beta}.\hat{e}(h, g_2)^{-r_{\delta_1} - r_{\delta_2}}$, $R_4 = C_1^{r_x} u^{-r_{\delta_1}}$, and $R_5 = C_2^{r_x} v^{-r_{\delta_2}}$. Let $c = H(M, C_1, C_2, C_3, R_1, R_2, R_3, R_4, R_5) \in Z_p$. Construct the values $s_\alpha = (r_\alpha + c\alpha)$, $s_\beta = (r_\beta + c\beta)$, $s_x = (r_x + cx)$, $s_{\delta_1} = (r_{\delta_1} + c\delta_1)$, and $s_{\delta_2} = (r_{\delta_2} + c\delta_2)$.
  Signature will be $\sigma = \langle C_1, C_2, C_3, c, CT_1, ..., CT_\mu, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2} \rangle$
- Verify(gpk, M, σ, ζ): To verify the signature we first define a recursive algorithm $VerNode$. If the node we are currently on is a leaf in the tree the algorithm returns the following:

$$VerNode(leaf) = \begin{cases} \hat{e}(CT_{leaf_j}, D_{leaf_j}) = \hat{e}(A_i h^{\alpha+\beta}, g_2)^{q_{leaf_j}(0)} \\ \text{Otherwise return } \perp \end{cases}$$

  For a node ρ which is not a leaf the algorithm proceeds as follows: For all children z of the node ρ it calls $VerNode$ and stores output as F_z. Let S_ρ be an arbitrary k_ρ sized set of children nodes z such that $F_z \neq \perp$ and if no such set exist return $\perp$.
  Otherwise let $\Delta_{S_\rho, index(z)} = \Pi_{j \in \{index(z): z \in S_\rho - index(z)\}}(-j/(index(z) - j))$ and compute
  $F_\rho = \Pi_{z \in S_\rho} F_z^{\Delta_{S_\rho, index(z)}}$
  $F_\rho = \Pi_{z \in S_\rho} \hat{e}(A_i h^{\alpha+\beta}, g_2)^{q_z(0).\Delta_{S_\rho, index(z)}}$
  $F_\rho = \Pi_{z \in S_\rho} \hat{e}(A_i h^{\alpha+\beta}, g_2)^{q_{parent(z)}(index(z)).\Delta_{S_\rho, index(z)}}$
  $F_\rho = \hat{e}(A_i h^{\alpha+\beta}, g_2)^{q_\rho(0)}$
  To verify the signature calculate F_{root}. If the tree is satisfied then $F_{root} = \hat{e}(C_3, w)$ according to Lagrange interpolation. Calculate $\bar{R}_1 = u^{s_\alpha} C_1^{-c}$, $\bar{R}_2 = v^{s_\beta} C_2^{-c}$, $\bar{R}_4 = C_1^{s_x} u^{-s_{\delta_1}}$, $\bar{R}_5 = C_2^{s_x} v^{-s_{\delta_2}}$,
  $\bar{R}_3 = \hat{e}(C_3, g_2)^{s_x}.\hat{e}(h, w)^{-s_\alpha - s_\beta}.\hat{e}(h, g_2)^{-s_{\delta_1} \cdot - s_{\delta_2}}.(F_{root}/\hat{e}(g_1, g_2))^c$.
  If $c = H(M, C_1, C_2, C_3, \hat{R}_1, \hat{R}_2, \hat{R}_3, \hat{R}_4, \hat{R}_5)$ then accept signature otherwise reject it.
- Open$(S_{para}, gmsk, M, \sigma)$:This algorithm traces a signature to a signer. To do so the group manager (which in this case is the key generator himself for reasons explained later on) will be using:
  The $S_{para} = \langle G_1, G_2, G_T, \hat{e}, H, g_1, g_2, h, u, v, gmsk, \gamma, w \rangle$. The group masters tracing key $gmsk = \langle \xi_1, \xi_2 \rangle$.
  A signature $\sigma = (C_1, C_2, C_3, c, CT_1, ..., CT_\mu, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$, on the message M. Step one in the tracing will be verifying the signature. Afterwards,

the group manager could recover A_i by calculating $A_i = C_3/(C_1^{\xi_1} C_2^{\xi_2})$. Now the manager could look up the user with index A_i. After looking up the user, manager could further up verify the attributes by calculating $T_{i,j} = CT_j/(C_1^{\xi_1} C_2^{\xi_2})^{t_j}$ and trying to calculate $\hat{e}(A_i, w)$ using the attribute tree and the recursive function shown below:
If the node we are currently on is a leaf in the tree the algorithm returns the following:

$$OpenNode(leaf) = \begin{cases} \hat{e}(T_{leaf_j}, D_{leaf_j}) = \hat{e}(A_i, g_2)^{q_{leaf_j}(0)} \\ \text{Otherwise return } \perp \end{cases}$$

For a node ρ which is not a leaf the algorithm proceeds as follows: For all children z of the node ρ it calls *OpenNode* and stores output as F_z. Let S_ρ be an arbitrary k_ρ sized set of children nodes z such that $F_z \neq \perp$ and if no such set exist return $\perp$.
Otherwise let $\Delta_{S_\rho,index(z)} = \Pi_{j \in \{index(z):z \in S_\rho - index(z)\}}(-j/(index(z) - j))$ and compute $F_\rho = \Pi_{z \in S_\rho} F_z^{\Delta_{S_\rho,index(z)}}$.
The reason behind limiting the possibility of being the group manager to the key generator was the need to use t_j when calculating $T_{i,j}$. That is a minor drawback in our system where it would be nice if their was some kind of hierarchy. For example, it would be practical if a senior manager could trace junior employees in his department rather than refering to the company everytime.

5 Security of the scheme

In this section we prove the scheme to be correct, anonymous, and traceable. We say the scheme is correct if honest signatures verify. The scheme is anonymous if signatures of same attributes do not reveal signers identity. The scheme is traceable if we could ensure that we could trace any signature even those created by a collusion of multiple users to a member of the forging coalition.

5.1 ABGS Scheme Correctness

Theorem 4. *The ABGS scheme is correct*

Proof. In order to do so we need to prove that $\bar{R_1} = R_1, \bar{R_2} = R_2, \bar{R_3} = R_3, \bar{R_4} = R_4, \bar{R_5} = R_5$ because that leads $c = H(M, C_1, C_2, C_3, \bar{R_1}, \bar{R_2}, \bar{R_3}, \bar{R_4}, \bar{R_5})$ which means the signature is accepted.
$\bar{R_1} = u^{s_\alpha} C_1^{-c} = u^{r_\alpha + c\alpha}.(u^\alpha)^{-c} = u^{r_\alpha} = R_1$
$\bar{R_2} = v^{s_\beta} C_2^{-c} = u^{r_\beta + c\beta}.(v^\beta)^{-c} = v^{r_\beta} = R_2$
$\bar{R_4} = C_1^{s_x}.u^{-s_{\delta_1}} = u^{\alpha(r_x + cx)}.u^{(-r\delta_1 - c\delta_1)} = C_1^{r_x}.u^{-r\delta_1} = R_4$
$\bar{R_5} = C_2^{s_x}.v^{-s_{\delta_2}} = v^{\beta(r_x + cx)}.v^{(-r\delta_2 - c\delta_2)} = C_2^{r_x}.v^{-r\delta_2} = R_5$
Finally,$\bar{R_3} = R_3$ holds for the following reasons:
$\hat{e}(C_3, g_2)^{s_x}.\hat{e}(h, w)^{-s_\alpha - s_\beta}.\hat{e}(h, g_2)^{-s_{\delta_1} \cdot -s_{\delta_2}}$
$= \hat{e}(C_3, g_2)^{r_x + cx}.\hat{e}(h, w)^{-r_\alpha - r_\beta - c\alpha - c\beta}.\hat{e}(h, g_2)^{-r_{\delta_1} - r_{\delta_2} - cx\alpha - cx\beta}$

$= \hat{e}(C_3, g_2^x)^c.\hat{e}(h^{-\alpha-\beta}, wg_2^x)^c(\hat{e}(C_3, g_2)^{r_x}.\hat{e}(h, w)^{-r_\alpha - r_\beta}.\hat{e}(h, g_2)^{-r_{\delta_1} - r_{\delta_2}})$
$= \hat{e}(C_3 h^{-\alpha-\beta}, wg_2^x)^c.\hat{e}(C_3, w)^{-c}(R_3)$
$= (\hat{e}(A, wg_2^x)/\hat{e}(C_3, w))^c R_3$
$= (\hat{e}(g_1, g_2)/F_{root})^c R_3$

5.2 ABGS scheme Traceablity

Theorem 5. *If SDH is hard on group (G_1, G_2) then the selective model of the Attribute Based Group Signature Scheme is fully-traceable. In other words, if there exists an adversary that attacks the FSA security of the scheme then there exist an adversary that could solve the SDH problem.*

Proof. In order to prove that we need three steps. Defining a security model for proving full-traceability, introducing two types of signature forger, and then we show that the existence of such forgers implies that SDH is easy. Suppose we are given an adversary *Adam* that breaks the full traceability of the signature scheme. The security model will be defined as an interacting framework between the *Challenger* and *Adam* as follows:

- **Init:** The *Challenger* runs *Adam*. *Adam* chooses the attribute tree it would be challenged upon.
- **Setup:** The *Challenger* runs the setup algorithm as in section [3] with a bilinear pair (G_1, G_2). It selects the generators g_1, g_2, a hash function H, ξ_1, ξ_2, u,v,h, and γ such that they all satisfy properties mentioned in section [3]. It also chooses a t_j for all attributes j in the tree *Adam* gave. The *Challenger* has to come up with the pairs $\langle A_i, x_i\rangle$ for an $i = 1, ..., n$. Some of those pairs have $x_i = \star$ which implies that x_i corresponding to A_i is not known; Other pairs is a valid SDH pair. In Setup the *Challenger* creates a public key for the same attribute tree. So *Adam* is given $gpk = \langle g_1, g_2, h, u, v, w, D_{leaf_1}, ..., D_{leaf_\mu}, h_1, ..., h_\mu\rangle$ and (ξ_1, ξ_2).
- **Hash Queries:** When the *Challenger* asks *Adam* for the hash of $(M, C_1, C_2, C_3, R_1, R_2, R_3, R_4, R_5)$, *Adam* responds with a random element in G_1 and saves the answer just incase the same query is requested again.
- **Signature Queries:** *Adam* asks for a signature on a message M by a key index i and a set of attributes ζ; where ζ satisfies the attribute tree chosen in Setup. If $x_i \neq \star$ *Challenger* calculates $T_{i,j} = A_i^{t_j}$ for all attributes in ζ and signs the message normally to obtain σ and give it to *Adam*. If $x_i = \star$ then *Challenger* picks randomly $\alpha, \beta \in Z_p$ sets $C_1 = u^\alpha, C_2 = v^\beta, C_3 = A_i g_1^{\alpha+\beta}$, and $CT_j = (A_i g_1^{\alpha+\beta})^{t_j}$ for every attribute in ζ. Now *Challenger* could get σ as shown in the signature algorithm and gives it to *Adam*
- **Private Key Queries:** *Adam* asks for the private key in a certain index i for an attribute set ζ. If $x_i \neq \star$ *Challenger* returns back $\langle A_i, x_i, T_{i,1}, ..., T_{i,\mu}\rangle$ where $T_{i,j} = A_i^{t_j}$ otherwise *Challenger* declares failure.
- **Output:** If *Adam* is successful, it outputs a forged signature on a message M. Such that the *Challenger* calculates A^* using ξ_1, ξ_2, C_1, C_2 and C_3. It calculates $T^*_{i,j}$ for all attributes j using $CT_j, \xi_1, \xi_2, C_1, C_2$ and t_j each time.

Now challenger runs $OpenNode$ as shown in (Section[3]) and if he outputs a result that does not equal $\hat{e}(A_i, w)$ then declare failure and terminate. Otherwise, if $A^* \neq A_i$ for all i output σ. If $A^* = A_{i^*}$ for some i^* and if $s_{i^*} = \star$ output σ. Last possibility is having $A^* = A_{i^*}$ but $s_i \neq \star$ $Challenger$ declares failure.

From this model of security there are two types of forgery. Type-I outputs a signature that could be traced to some identity which is not part of $\{A_1, ..., A_n\}$. Type-II has $A^* = A_{i^*}$ where $1 \leq i^* \leq n$ but $Adam$ did not do a private key query on i^*. We should prove that both forgeries are hard.

Type-I: If we consider Lemma 1 for a $(n+1)$SDH, we could obtain g_1,g_2 and w. We could also use the n pairs (A_i, x_i) to calculate the private keys $\langle A_i, x_i, A_i^{t_1}, ..., A_i^{t_\mu} \rangle$. We use these values in interacting with $Adam$. $Adam$'s success leads to forgery of Type-I and the probability is ε.
Type-II: Using the same Lemma 1 but for a nSDH this time, we could obtain g_1, g_2 and w. Then we could also use the $n-1$ pairs (A_i, x_i) to calculate the private keys $\langle A_i, x_i, A_i^{t_1}, ..., A_i^{t_\mu} \rangle$. In a random index i^*, we could choose the missing pair randomly where $A_{i^*} \in G_1$ and set $x_{i^*} = \star$. The random private key $\langle A_{i^*}, x_{i^*}, A_{i^*}^{t_1}, ..., A_{i^*}^{t_\mu} \rangle$. $Adam$ in the security model will fail if he queries the private key oracle in index i^*. Other private key queries will succeed. In the signature oracle and because the hashing oracle is used it will be hard to distinguish between signatures with a SDH pair and ones without. As for the output algorithm the probability of tracing to a forged signature that leads to index i^* is equal to ε/n.
Next is showing how the Forking Lemma (Section[3]) could be applied here to prove that we could generate new SDH pairs if a forgery of any type exists. Let $Adam$ be a forger of any type in which the security model succeeds with probability $\grave{\varepsilon}$. A signature will be represented as $\langle M, \sigma_0, c, \sigma_1, \sigma_2 \rangle$. M is the signed message. $\sigma_0 = \langle C_1, C_2, C_3, R_1, R_2, R_3, R_4, R_5 \rangle$. c is the value derived from hashing σ_0. $\sigma_1 = \langle s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2} \rangle$ which are values used to calculate the missing inputs for the hash function. Finally $\sigma_2 = \langle CT_1, ..., CT_\mu \rangle$ the values that depend on the set of attributes in each signature oracle.
One simulated run of the adversary is described by a random string ω used by the adversary $Adam$ and a vector ℓ of the responses made by the hash oracle. Let S be a set of the pairs $\langle \omega, \ell \rangle$ where $Adam$ successfully forges the signature $(M, \sigma_0, c, \sigma_1, \sigma_2)$. Let $Ind(\omega, \ell)$ be the index of ℓ on which $Adam$ queried (M, σ_0). Let $\nu = Pr[S] = \grave{\varepsilon} - 1/p$ which represents the probability of the security model ending with a success subtracting the possibility that $Adam$ guessed the hash of (M, σ_0) without the help of the hash oracle. For each χ, $1 \leq \chi \leq q_H$, let S_χ be a set of pairs $\langle \omega, \ell \rangle$ where $Ind(\omega, \ell) = \chi$. Let $\varPhi$ be the set of indices χ where $Pr[S_\chi | S] \geq 1/2q_H$ causing $Pr[Ind(\omega, \ell) \in \varPhi | S] \geq 1/2$.
Let $\ell|_a^b$ be the restriction of ℓ to its elements at indices $a, a+1, ..., b$. For each $\chi \in \varPhi$ consider the heavy row lemma (See Section[2.5]) with a matrix with rows indexed with$(\omega, \ell|_1^{\chi-1})$ and columns $(\ell|_\chi^{q_H})$. If (x, y) is a cell, then $Pr[(x, y) \in S_\chi] \geq \nu/2q_H$. Let the heavy rows $\varOmega_\chi$ be the ones such that $\forall (x, y) \in$

$\Omega_\chi : Pr_{\grave{y}}[(x,\grave{y}) \in S_\chi] \geq \nu/(4q_H)$. By the heavy row lemma $Pr[\Omega_\chi|S_\chi] \geq 1/2$ which leads to $Pr[\exists\chi \in \Phi : \Omega_\chi \cap S_\chi|S] \geq 1/4$.
Therefore *Adam*'s probability in forging a signature is about $\nu/4$. That signature derives from the heavy row $(x,y) \in \Omega_\chi$ for some $\chi \in \Phi$, hence execution (ω,ℓ) such that the $Pr_{\grave{\ell}}[(\omega,\grave{\ell}) \in S_j|\grave{\ell}|_1^{j-1} = \ell|_1^{j-1}] \geq \nu/(4q_H)$. In other words if we have another simulated run of the adversary with $\grave{\ell}$ that differs from ℓ starting the jth query *Adam* will forge another signature $\langle M, \sigma_0, \grave{c}, \grave{\sigma_1}, \sigma_2\rangle$ with the probability $\nu/(4q_H)$. Now we show how we could extract from $\langle\sigma_0, c, \sigma_1, \sigma_2\rangle$ and $\langle\sigma_0, \grave{c}, \grave{\sigma_1}, \sigma_2\rangle$ a new SDH tuple. Let $\Delta c = c - \grave{c}$, $\Delta s_\alpha = s_\alpha - \grave{s}_\alpha$, and similarly for Δs_β,Δs_x,Δs_{δ_1}, and Δs_{δ_2}.
Divide two instances of the equations used previously(See Theorem[4] proof) where one instance is with $\grave{c}$ and the other is with c to get the following:

- Dividing $R_1/\grave{R}_1$ we get
  $u^{\tilde{\alpha}} = C_1$; where $\tilde{\alpha} = \Delta s_\alpha/\Delta c$
- Dividing $R_2/\grave{R}_2$ we get
  $v^{\tilde{\beta}} = C_2$; where $\tilde{\beta} = \Delta s_\beta/\Delta c$
- Dividing $C_1^{s_x}/C_1^{\grave{s}_x} = u^{s_{\delta_1}}/u^{\grave{s}_{\delta_1}}$ will lead to
  $\Delta s_{\delta_1} = \tilde{\alpha}\Delta s_x$
- Similarly dividing $C_2^{s_x}/C_2^{\grave{s}_x} = v^{s_{\delta_2}}/u^{\grave{s}_{\delta_2}}$ will lead to
  $\Delta s_{\delta_2} = \tilde{\beta}\Delta s_x$
- Calculating the following equality:
  $(\hat{e}(g_1,g_2)/F_{root})^{\Delta c} = (\hat{e}(g_1,g_2)/\hat{e}(C_3,w))^{\Delta c}$
  $= \hat{e}(C_3,g_2)^{\Delta s_x}.\hat{e}(h,w)^{-\Delta s_\alpha - \Delta s_\beta}.\hat{e}(h,g_2)^{-\Delta s_{\delta_1} - \Delta s_{\delta_2}}$
  $= \hat{e}(C_3,g_2)^{\Delta s_x}.\hat{e}(h,w)^{-\Delta s_\alpha - \Delta s_\beta}.\hat{e}(h,g_2)^{-\tilde{\alpha}\Delta s_x - \tilde{\beta}\Delta s_x}$

From the equations above if we let $\tilde{x} = \Delta s_x/\Delta c$ and $\tilde{A} = C_3 h^{-(\tilde{\alpha}+\tilde{\beta})}$ we get the following equation:
$\hat{e}(g_1,g_2)/\hat{e}(C_3,w) = \hat{e}(C_3,g_2)^{\tilde{x}}.\hat{e}(h,w)^{-\tilde{\alpha}-\tilde{\beta}}\hat{e}(h,g_2)^{-\tilde{x}(\tilde{\alpha}+\tilde{\beta})}$
$\hat{e}(g_1,g_2) = \hat{e}(\tilde{A}, wg_2^{\tilde{x}})$
Hence we obtain a new SDH pair $(\tilde{A},\tilde{x})$ breaking Boneh and Boyens Lemma(See Section[1]). Now putting things together we get the following claims:

Theorem 6. *We could solve an instance of* $(n+1)$ *SDH with a probability* $(\varepsilon - 1/p)^2/16q_H$ *using a Type-I forger Adam*

Theorem 7. *We could solve an instance of* n *SDH with a probability* $(\varepsilon/n - 1/p)^2/16q_H$ *using a Type-II forger Adam*

5.3 ABGS Scheme Anonymity

Theorem 8. *If the linear encryption is semantically secure then the ABGS scheme is fully anonymous under the same attribute tree. In other words, if there exists an adversary that breaks the scheme's SSA security then there exists an adversary that breaks the semantic security of linear encryption.*

Assuming *Adam* is an adversary that breaks the anonymity of the ABGS scheme. We will prove that there is an adversary *Eve* that breaks the semantic security of the linear encryption using *Adam*'s talent. Note that *Eve* in this game plays a challenger role when it comes to interacting with *Adam* and an adversary role when she interacts with *Challenger*.So the game is demonstrated below:

- **Init:** *Adam* decides the attribute tree he would like to be challenged upon and gives it to *Eve*.
- **Setup:** *Eve* is given the public key $LE_{PK} = \langle u, v, h \rangle$ from the $Challenger$. *Eve* chooses a random γ from Z_p and $t_1, ..., t_m$. Using the LE_{PK} key and the random values, *Eve* could calculate an ABGS public key for the attribute tree $gpk = \langle u, v, h, w, D_{leaf_1}, ..., D_{leaf_\mu}, h_1, ..., h_\mu \rangle$ for the ABGS scheme. *Eve* also calculates n private key bases $gsk[i]_{base} = \langle A_i, x_i \rangle$ where $1 \leq i \leq n$.
- **Phase 1:** *Eve* runs three oracles a signature oracle, private key oracle and a hash oracle. The hash oracle has a list that saves a unique random value for each 9-element tuple. That random value is the response of the oracle. The hash oracle should guarantee that no 9-element tuple have the same random value and that each time it responds with the same random value for the same 9-element tuple. In the signature oracle *Adam* sends an index i, a random message M and a set of attributes ζ to *Eve* where ζ satisfies the tree. *Eve* responds back with a signature $\sigma = \langle C_1, C_2, C_3, c, CT_1, ...CT_\mu, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2} \rangle$ on that message from user i. c is the response of the hash oracle for the tuple $\langle M, C_1, C_2, C_3, R_1, R_2, R_3, R_4, R_5 \rangle$. Finally, the private key oracle *Adam* sends an index i and a set of attributes ζ and *Eve* responds back with $\langle A_i, x_i, A_i^{t_1}, ..., A_i^{t_\mu} \rangle$.
- **Challenge:** Now *Adam* could request from *Eve* his anonymity challenge by choosing two indices (i_0 and i_1), set of attributes ζ and a message M asking for a signature of one of them. *Eve* sends $Challenger$ both $\langle A_{i_0}, A_{i_1} \rangle$ as messages requesting a challenge. $Challenger$ responds back with the ciphertext $\bar{C} = \langle C_1, C_2, C_3 \rangle$ of A_{i_b} where $b \in \{0, 1\}$. *Eve* generates a signature from $\bar{C} = \langle C_1, C_2, C_3, C_3^{t_1}, ..., C_3^{t_\mu} \rangle$ and sends it to *Adam*.
- **Phase 2:** *Adam* goes back to issuing further queries as done in Phase one.
- **Guess:***Adam* returns a $\grave{b}$ to *Eve*.

Eve outputs $\grave{b}$ as her answer to the $Challenger$. *Eve* has an high advantage on guessing the right $\grave{b} = b$ if and only if *Adam* could break into the anonymity of the ABGS scheme.

6 Conclusion

In this paper we define the first attribute based group signature scheme. An ABGS scheme enables signing with attributes rather than just a private key. It also verifies a person and his characteristics. We define a security model for anonymity, and traceability for such schemes. We then construct a scheme based on group signatures [4] and the idea of attribute trees [9].
We still have some open problems for future work.

- The revocation problem: This problem could be for removing certain attributes from a certain user. For example, an employee in Bob's company could be transferred to a different department. The revocation problem also includes removing a user from the system and that is handy when an employee at Bob's company quits his job.
- The attribute set anonymity: This problem involves ensuring more privacy for the signer. His attributes should be kept a secret. For example when Alice sets her attribute tree and asks for a corresponding signature, she does not need to know what sub-tree the employee in Bob's company satisfies. In other words, all she needs to know is whether in general the employee owns enough attributes for her to accept the signature. No need to know the attributes themselves. It is a stronger anonymity level that we would have liked to achieve.
- The Attribute set size effects efficiency: Finally, our scheme has a disadvantage when it comes to having a huge number of attributes since the keys and signature are dependent on the size of the attribute set a user owns or requests.

Our paper solves one of the open issues pointed out in [1] which is having subgroups in a group signature scheme. Having such a feature enables us to identify the subgroup a signature belongs to when needed. You could consider a group which satisfies a certain attribute tree as a subgroup from the original scheme in [4]. The concept of subgroups is different then the concept of multigroup signature. Those signatures are used when a document needs to be signed by two different groups. A members that belongs to the intersection could sign on behalf of both groups with one signature [10] [13]. The idea of having subgroups in a group signature scheme was introduced in [1] and as long as we know there has not been any scheme that provides such property and our scheme is would be the first step towards achieving it.

References

1. G. Ateniese and G. Tsudik. Some open issues and new directions in group signatures. In *Financial Crypto'99*, volume 1648, pages 196–211, 1999.
2. M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions simplified requirements and a construction based on general assumptions. In *Proceedings of Eurocrypt 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 614–629. Springer-Verlag, 2003.
3. D. Boneh and X. Boyen. Short signatures without random oracles. In *Proceedings of Eurocrypt 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 382–400. Springer-Verlag, 2004.
4. D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *Proceedings of Crypto 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41 – 55. Springer-Verlag, 2004.
5. J. Camenisch. Efficient and generalized group signatures. In *Proceedings of Eurocrypt 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 465–479. Springer-Verlag, 1997.

6. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *In Advances in Cryptology CRYPTO'97*, volume 1296 of *Lecture Notes in Computer Science*, pages 410–424. Springer-Verlag, 1997.
7. D. Chaum and V. Heyst. Group signatures. In *Proceedings of Eurocrypt 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer-Verlag, 1991.
8. L. Chen and T.P. Pedersen. New group signature schemes. In *Proceedings of Eurocrypt 1994*, volume 950 of *Lecture Notes in Computer Science*, pages 171–181. Springer-Verlag, 1995.
9. V. Goyal, O. Pandeyy, A. Sahaiz, and B. Waters. 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, 2006.
10. S. Kim, S. Park, and D. Won. Group signatures for hierarchical multigroups. In *Proceedings of the First International Workshop on Information Security table of contents*, volume 1396, pages 273 – 281, 1997.
11. Kazuo Ohta and Tatsuaki Okamoto. On concrete security treatment of signatures derived from identification. In *Advances in Cryptology - CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 354–??, 1998.
12. D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. In *Journal of Cryptography*, volume 13 of *Number 3*, pages 361–396. Springer-Verlag, 2000.
13. M. Trolin and D. Wikstrom. Hierarchical group signatures. In *Automata, Languages and Programming*, volume 3580, pages 446–458, 2005.