Attribute Based Group Signature with Revocation

Dalia Khader

University of Bath, Department of Computer Science,
Bath, BA27AY, UK.
ddk20@bath.ac.uk

Abstract. In real life, one requires signatures to be from people who fulfill certain criteria, implying that they should possess specific attributes. For example, Alice might want a signature from an employee in Bobs company who is a member in the IT staff, a senior manager within the biometrics team or at least a junior manager in the cryptography team. In such a case an Attribute Based Group Signature scheme (ABGS) could be applied. Group signature schemes are those where each member of a group can sign on behalf of the others. An ABGS scheme is a type of group signature scheme, where the signing member has to have certain attributes. In [12], the authors introduced the first ABGS but it lacked the ability to revoke. In this paper, we introduce a new scheme that will enable us to remove a member from a group or remove some of his attributes, when needed.

1 Introduction

Attribute Based Group Signatures were first introduced in [12]. It was proposed to serve the purpose of including attributes in a group signature scheme. Group Signatures allow a member of a group to sign on behalf of the others while in ABGS schemes the aim is to allow a member of the group only possessing certain attributes to sign on behalf of the rest.
Chaum and van Heist [9] proposed the first group signature in order to implement e-cash systems. The two underlying security notions of group signatures are anonymity and traceability. An anonymous scheme is a scheme that does not reveal the signers identity. A traceable scheme is such that a group of colluding members cannot forge a signature that will not be traced to at least one of them. Since Group Signatures was introduced, different security notions were defined; some examples are unlinkability, unforgeablitity, collusion resistance, exculpability, and framing resistance. In Bellare et al. [4] strong definitions of the core requirements were formulated and defined. The authors in that paper proposed two security notions that implies all the rest and they defined them as full traceability and full anonymity. Since the scheme introduced in our paper could be considered as a type of group signature scheme we adopt the security notions presented in Bellare et al.s work and modify them as done in [12](See Section 3.2). The reader is referred to Bellare et al.s work in [4] for further details about

the other security notions mentioned earlier.
On a separate research line, papers such as [8, 16, 1] looked at improving performance. In [10, 3, 13], authors investigated dynamic groups where revocation of users and admission of new users were considered. Complex group signatures such as hierarchal groups, multi-groups and sub-groups were proposed in [18, 2, 14, 12]. Hierarchal groups are group signatures that have different levels of group managers who are capable of tracing, adding and revoking members under their authority. In multi-group schemes, a member that belongs to an intersection of two groups could sign on behalf of both. Subgroup schemes are used when a document needs to be signed by a member of a group that belongs to a certain subgroup of the group in question. ABGS is a type of group signature that supports sub-groups. It allows any member of the group, who satisfies certain properties, to sign on behalf of the others. Our ABGS is very similar to the one in [12]. The concept of an attribute tree from Goyal et al's work in [11] is applied in our scheme as well. An attribute tree is a tree in which interior nodes are threshold gates and the leaves are linked to attributes. A threshold gate represents that the number m of n children branching from the current node need to be satisfied in order to imply that the parent node is satisfied. Satisfaction of a leaf is achieved by owning an attribute. For further explanation, consider the example in Figure 1, which demonstrates an attribute tree for the scenario mentioned in the abstract.
Similar to the scheme in [12], each public key is linked to an attribute tree. So the verifier chooses a public key that applies to his requirements. The signer has elements in his private key corresponding to the attributes he owns. When signing, he uses the needed elements to satisfy the verifiers tree.
Hence, in context of the scenario represented in the abstract, Alice decides on an attribute tree and sends it to both a key generator and Bob's company. The key generator sends Alice a verifying key and a member in Bob's company sends her a signature. Alice could now verify Bob's signature and whether he satisfies her attribute tree or not.
In this paper we begin by discussing some preliminaries that will be used in the scheme and its security proofs. Then in section 3 we define an ABGS scheme and the two security notions required. We then construct a scheme in section 4 and prove it to be secure in section 5. Finally we conclude by discussing some open problems.

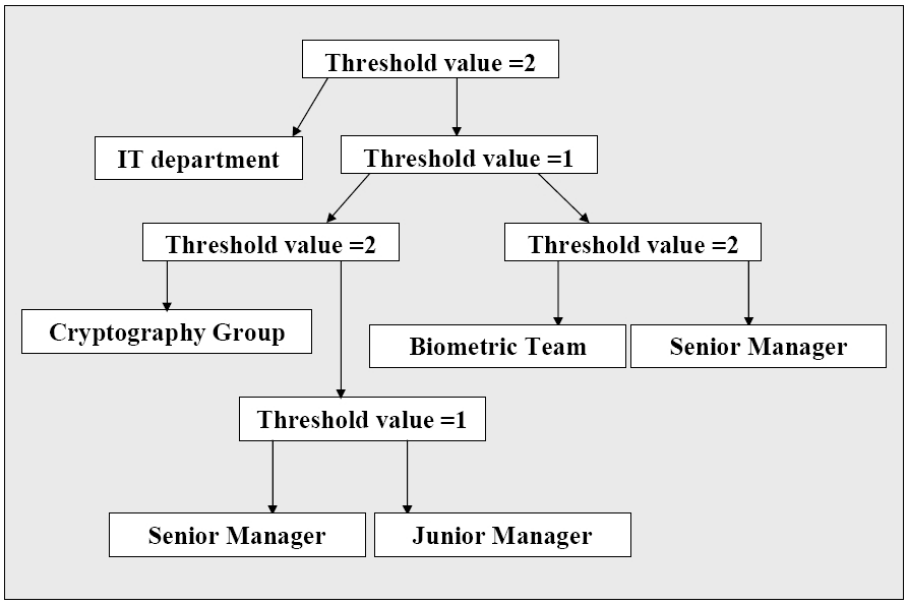

Fig. 1. Attribute Tree

2 Preliminaries

In this section we will explain some of the preliminaries that are used in constructing ABGS scheme and proving it secure.

2.1 Bilinear Maps

Bilinear Maps are used in constructing our ABGS in section 4.

Definition 1. (Bilinear Maps): *Let G_1,G_2 and G_T be three groups of order p for some large prime p. A bilinear map $\hat{e} : G_1 \times G_2 \rightarrow G_T$ must satisfy the following properties:*

- *Bilinear: We say that a map $\hat{e} : G_1 \times G_2 \rightarrow G_T$ is bilinear if $\hat{e}(g_1^a, g_2^b) = \hat{e}(g_1, g_2)^{ab}$ for any generator $g_1 \in G_1$, $g_2 \in G_2$ and any $a, b \in Z_p$.*
- *Non-degenerate: The map does not send all pairs in $G_1 \times G_2$ to the identity in G_T.*
- *Computable: There is an efficient algorithm to compute $\hat{e}(g_1, g_2)$ for any $g_1 \in G_1$ and $g_2 \in G_2$.*

A bilinear map satisfying the three properties above is said to be an admissible bilinear map.

2.2 Complexity Assumptions

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 constructed scheme. We will also define the Decision Linear Diffie-Hellman Assumption [6]. This will be used in the construction of our ABGS scheme and will lead to ensuring anonymity(See Section 5.3) of the 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 [5]:

Definition 2. (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 what is called a SDH pair and that equals* $(g_1^{1/(\gamma+x)}, x)$ *where* $x \in Z_p^*$. *An algorithm A has an advantage* ε *in solving q-SDH in* (G_1, G_2) *if:*

$$Pr[A(g_1, g_2, g_2^{\gamma}, g_2^{\gamma^2}, ..., g_2^{\gamma^q}) = (g_1^{1/(\gamma+x)}, x)] \geq \varepsilon,$$

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 [5].*

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

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 [5] 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 [5].*

Definition 3. (Decision Linear Problem in G_1) *Let* G_1 *be a group of prime order p. Let* u_0, u_1, u_2 *be generators in that group, and* $a, b \in Z_p$.
Given $\langle u_0, u_1, u_2, u_0^a, u_1^b, Z \rangle \in G_1$ *as an input,it is hard to decide whether or not* $a + b = c$ *[6].*

2.3 Forking Lemma and Heavy Row Lemma

Pointcheval et al. [17], developed the Forking Lemma as a method to prove certain security notions of digital signature scheme. 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 a message M together with σ_1. σ_2 depends only on (σ_1, h, M). The Forking Lemma is as follows [17]:

Theorem 2. (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 but a different oracle, outputs new valid signatures* $(M, \sigma_1, h, \sigma_2)$ *and* $(M, \sigma_1, \grave{h}, \grave{\sigma_2})$ *such that* $h \neq \grave{h}$.

We now define a Boolean Matrix, and a Heavy Row in that matrix [15]. The definitions are used in the Heavy Row Lemma [15] 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 equals $\top$, the probability that it lies in a heavy row is at least 1/2.*

3 ABGS Scheme

In this section we will first define an ABGS scheme. Then we define two of it's security notions.

3.1 General Definition of the ABGS scheme

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

- *Setup*: Setup is a randomized algorithm. It takes a security parameter as an input. It generates a set of parameters S_{para} that will be used in the KeyGen algorithm.
- $KeyGen(S_{para}, n)$: KeyGen is an algorithm that takes the system parameters, and a number n that defines the number of users. It generates what is called private key bases $gsk[i]_{base}$ for any user i. It generates public keys and private keys using two sub-algorithms as follows:
  $KeyGen_{public}(\Gamma)$*:* This algorithm generates public keys gpk for attribute trees described in Γ(See Figure 1 as an example).
  $KeyGen_{priv}(gsk[i]_{base}, \Upsilon_i)$*:* Creates the private key $gsk[i]$ for user i to enable him to authenticate himself and his properties which are described in Υ_i.
- $Sign(gpk, gsk[i], M)$: Given a public key of an attribute tree, a private key of a user i and a message. Output a signature σ and ζ_i. ζ_i is a set that describes the set of attribute that satisfy the tree. Notice that $\zeta_i \subseteq \Upsilon_i$
- $Verify(gpk, M, \sigma, \zeta_i, R)$: Given a message M, a public key of a certain attribute tree gpk, a signature σ, a revocation table R and a set ζ_i. Output either an acceptance or a rejection for the signature.
- $Revoke(i, \varphi)$: Revoke gets an index of a user i and a set of attributes φ to be revoked. If φ is an empty set then revoke the user i rather than an attribute of his. The output of this algorithm is a modified revocation table R

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 correctly.*

Defining anonymity and traceability is not as straight forward as with correctness. We need to introduce an adversarial model. We adopt some of our ideas from [6, 4] models.
For defining anonymity we introduce this game between an adversary *Adam* and a *Challenger*. The game consists of six steps: Init, Setup, Phase1, Challenge, Phase2, and finally Guess as follows:

- **Init:** *Adam* chooses the attribute tree Γ he would like to be challenged upon.
- **Setup:** *Challenger* runs the *Setup* and $KeyGen$ algorithms without running $KeyGen_{priv}$. *Challenger* produces a public key for the attribute tree Γ and n private key bases gpk_{bases}.
- **Phase 1:** *Challenger* runs a signature oracle, a private key oracle and a revocation oracle. In the signature oracle, *Adam* sends a message M, index of user i and a set of attributes ζ_i that satisfy the tree. *Challenger* responds back with a signature σ. In the private key oracle *Adam* sends an index i and a set of attributes Υ_i. *Challenger* responds back with a private key $gsk[i]$. This oracle is equivalent to the $KeyGen_{priv}$. Finally, *Challenger* could query a revocation oracle, where it submits an index of user i and a set of attributes φ which could be an empty set. He gets back a modified revocation table R.
- **Challenge:** *Adam* decides when to request his challenge. He sends the *Challenger* the two triples $\langle i_0, M, \zeta\rangle$ and $\langle i_1, M, \zeta\rangle$. *Challenger* replies back with a signature σ_b where $b \in \{0, 1\}$ and σ_b is user i_b's signature.
- **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 scheme 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 traceability, we need to prove that a group of colluding members can not generate a valid signature, which does not trace to any of them. In other words, a forged signature is said to be untraceable, if it is verifiable even after revoking all users. In [4] they viewed full-traceability as a strong form of collusion-resistance. In order to prove full-traceability in ABGS schemes, we define the following game between an adversary *Adam* and the *Challenger*:

- **Init:** *Adam* chooses the attribute tree Γ he would like to be challenged upon.
- **Setup:** *Challenger* runs the *Setup* and the $KeyGen$ algorithm except for the $KeyGen_{priv}$. *Challenger* produces a public key gpk for the attribute tree and n private key bases $gsk[i]_{base}$.
- **Querying Oracles:** *Challenger* runs three oracles, a signature oracle, a private key oracle and a revocation oracle. *Adam* issues a number of queries to all oracles. He sends in every query to the signature oracle a message M, index of user i and a set of attributes ζ_i that satisfies the tree. *Challenger* responds back with a signature σ. When querying the private key oracle *Adam* sends an index i and a set of attributes Υ_i. *Challenger* responds back with a valid private key $gsk[i]$. For the revocation oracle *Adam* sends an index i and a set φ. He gets back a modified revocation list R.
- **Output:** If *Adam* is successful he outputs a forged signature σ that *Challenger* could verify but can not trace it to any user. Otherwise *Adam* fails.

We call an attack similar to *Adam*'s an Un-Traceability Attack (UTA). We represent the advantage of the adversary in winning the attack as Adv_{UTA}.

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

In the following section we will construct an ABGS scheme. We will later on (See section 5) prove it to be secure under UTA and SAA attacks.

4 Construction of an ABGS Scheme

In this section we construct an ABGS scheme based on Boneh et al.'s [7].

- *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 decision linear assumption holds on G_2. Define the bilinear map $\hat{e} : G_1 \times G_2 \rightarrow G_T$. All three groups G_1, G_2, G_T are multiplicative and of a prime order p. Select a hash function $H : \{0,1\}^* \rightarrow Z_p$. Select a hash function H_0 with respected range G_2^2. Select a generator $g_2 \in G_2$ at random and then set $g_1 \leftarrow \psi(g_2)$. 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 $S_{para} = \langle G_1, G_2, G_T, \hat{e}, H, H_0, g_1, g_2, \gamma, w, t_1, ..., t_m \rangle$.

- $KeyGen(S_{para}, n, \Gamma)$: This algorithm generates a public key for a specific access structure Γ and a private key for each user i. Using γ generate for each user $i, 1 \leq i \leq n$ a private key base $gsk[i]_{base} = \langle A_i, x_i \rangle$. All $gsk[i]_{base}$ should be an SDH pair, where x_i is chosen randomly from Z_p^* and $A_i = g_1^{1/(\gamma + x_i)} \in G_1$.

  $KeyGen_{public}(\Gamma)$: To generate a public key for a certain attribute tree Γ we will need to choose a polynomial q_{node} of degree $d_{node} = k_{node} - 1$

for each node in the tree. k_{node} is the threshold gate value of every node. In other words k_{node} children need to be satisfied in order to consider the parent satisfied. Choosing the polynomials 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 $gpk = \langle g_1, g_2, w, D_{leaf_1}, ..., D_{leaf_\kappa} \rangle$ where $D_{leaf_j} = \langle D_{(0,j)}, D_{(1,j)} \rangle =$ $\langle g_2^{q_{leaf_j}(0)/t_{leaf_j}}, g_2^{q_{leaf_j}(0)} \rangle$, κ is the number of leafs and $1 \leq j \leq \kappa$.

$KeyGen_{priv}(gsk[i]_{base}, \Upsilon)$: For every attribute $j \in \Upsilon$ calculate $T_{i,j} = A_i^{t_j}$. 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 size of Υ.

– $Sign(gpk, gsk[i], M)$: For signing the user i takes the public key gpk, user private key $gsk[i]$ and the message M. User picks randomly an r from Z_p and obtains $(\hat{u}, \hat{v})$ from $H_0(gpk, M, r)$. Then compute their images $u \leftarrow \psi(\hat{u})$ and $v \leftarrow \psi(\hat{v})$. Then randomly user i chooses α from Z_p. Then computes

$C_1 = u^\alpha$, $C_2 = A_i v^\alpha$, $CT_1 = \langle \beta_{(0,i,1)}, \beta_{(1,i,1)} \rangle$,...,$CT_\mu = \langle \beta_{(0,i,\mu)}, \beta_{(1,i,\mu)} \rangle$,

where $\beta_{(0,i,j)} = T_{i,j} v^\alpha$ and $\beta_{(1,i,j)} = \hat{e}(A_i^{t_j}, D_{(1,j)}).\hat{e}(v^\alpha, D_{(0,j)})$.

Lets $\delta = x_i \alpha$. Pick randomly r_α, r_x, and r_δ from Z_p.

Let $R_1 = u^{r_\alpha}$, $R_2 = \hat{e}(C_2, g_2)^{r_x} \hat{e}(v, w)^{-r_\alpha} \hat{e}(v, g_2)^{-r_\delta}$ and $R_3 = C_1^{r_x} u^{-r_\delta}$.

Compute $c = H(gpk, M, r, C_1, C_2, R_1, R_2, R_3)$, $s_\alpha = r_\alpha + c\alpha$, $s_x = r_x + cx_i$ and $s_\delta = r_\delta + c\delta$.

Finally, output the signature $\sigma = (r, C_1, C_2, c, s_\alpha, s_x, s_\delta, CT_1, ..., CT_\mu)$.

– $Verify(gpk, M, \sigma, \zeta, R)$: The verification algorithm takes as input a signature σ, a public key gpk, a message M, a set of attributes ζ and a revocation table R. To verify the signature we first define a recursive algorithm Ver_{Node}. If the node we are currently on is a leaf in the tree the algorithm returns the following:

$$Ver_{Node}(leaf)^1 = \begin{cases} \text{If } (j \in \zeta); \text{ return } \frac{\hat{e}(\beta_{(0,i,j)}, D_{(0,j)} D_{(1,j)})}{\beta_{(1,i,j)}} = \hat{e}(A_i v^\alpha, g_2)^{q_{leaf_j}(0)} \\ \text{Otherwise return } \perp \end{cases}$$

For a node ρ which is not a leaf the algorithm proceeds as follows: For all children z of the node ρ it calls Ver_{Node} and stores output as F_z. Let $\hat{S}_\rho$ be an arbitrary k_ρ sized set of children nodes z such that $F_z \neq \perp$ and if no such

[1] Correctness of the equation is proved in Section 5.1

set exist return $\perp$.
Otherwise let $\Delta_{\hat{S}_\rho,index(z)} = \Pi_{\iota\in\{index(z):z\in\hat{S}_\rho-index(z)\}}(-\iota/(index(z)-\iota))$ and compute

$F_\rho = \Pi_{z\in\hat{S}_\rho} F_z^{\Delta_{\hat{S}_\rho,index(z)}}$

$F_\rho = \Pi_{z\in\hat{S}_\rho} \hat{e}(A_i v^\alpha, g_2)^{q_z(0).\Delta_{\hat{S}_\rho,index(z)}}$

$F_\rho = \Pi_{z\in\hat{S}_\rho} \hat{e}(A_i v^\alpha, g_2)^{q_{parent(z)}(index(z)).\Delta_{\hat{S}_\rho,index(z)}}$

$F_\rho = \hat{e}(A_i v^\alpha, g_2)^{q_\rho(0)}$

To verify the signature calculate F_{root}. If the tree is satisfied then $F_{root} = \hat{e}(C_2, w)$(See Section 5.1 for prove). The verifier could then calculate $(\hat{u}, \hat{v}) = H_0(gpk, M, r)$ then $u \leftarrow \psi(\hat{u})$, and $v \leftarrow \psi(\hat{v})$. Verifier redrives R_1,R_2 and R_3 by calculating

$\bar{R}_1 = u^{s_\alpha}/C_1^c,$

$\bar{R}_3 = C_1^{s_x} u^{-s_\delta}$

$\bar{R}_2 = \hat{e}(C_2, g_2)^{s_x} \hat{e}(v, w)^{-s_\alpha} \hat{e}(v, g_2)^{-s_\delta} . (\frac{F_{root}}{\hat{e}(g_1,g_2)})^c.$

If $c \neq H(gpk, M, r, C_1, C_2, \bar{R}_1, \bar{R}_2, \bar{R}_3)$ reject it the signature otherwise check whether the user is in the revocation table.

The revocation table R has all values of $A_{revoked}$ that belong to a revoked user in one column. Then each other column in the table represents an attribute and the list of people who no longer own that attribute. In other words the rest of the columns contain values of $T_{(user,revoked)}$. If for all revoked users $\hat{e}(C_2/A_{revoked}, \hat{u}) = \hat{e}(C_1, \hat{v})$ does not hold then user i still is a valid user and if for attributes user i owns, $\hat{e}(\beta_{(0,i,j)}/T_{user,revoked}, \hat{u}) = \hat{e}(C_1, \hat{v})$ does not hold then user still owns the attribute j. Finally if user has not been revoked and still own all his attributes then accept signature.

– *Revoke*(i, φ): Revoke is about building a revocation table R. The algorithm gets an index of a user i and a set of attributes φ to be revoked. $\forall j \in \varphi$, add $T_{i,j}$ to the column j in the revocation table. If φ is an empty set then revoke the user i by adding A_i to the revoked users column.

5 Security Notions of the Scheme

In this section we prove the scheme to be correct, anonymous and traceable, using the definitions in section[3].

5.1 Correctness of the ABGS Scheme

Theorem 3. *The ABGS scheme is correct.*

Proof. To prove the scheme is correct we need to show that $R_1 = \bar{R}_1$, $R_2 = \bar{R}_2$, and $R_3 = \bar{R}_3$. If all three equalities hold then $H(gpk, M, r, C_1, C_2, R_1, R_2, R_3) = H(gpk, M, r, C_1, C_2, \bar{R}_1, \bar{R}_2, \bar{R}_3)$ and signature should be correctly verified unless the user is revoked or one of his attributes are. We start our proof with showing that the three equations hold:

$$\bar{R}_1 = u^{s_\alpha}/C_1^c = u^{r_\alpha + c\alpha}/u^{c\alpha} = u^{r_\alpha} = R_1$$

$$\bar{R}_3 = C_1^{s_x}.u^{-s_\delta} = (u^\alpha)^{r_x + cx_i}.u^{-(r_\delta + c\delta)} = u^{\alpha r_x + \alpha c x_i - r_{delta} - c\delta} = C_1^{r_x}.u^{-r_\delta} = R_3$$

$$\bar{R}_2 = \hat{e}(C_2, g_2)^{s_x}\hat{e}(v, w)^{-s_\alpha}\hat{e}(v, g_2)^{-s_\delta}.(\frac{F_{root}}{\hat{e}(g_1, g_2)})^c$$

$$= \hat{e}(C_2, g_2)^{s_x}\hat{e}(v, w)^{-s_\alpha}\hat{e}(v, g_2)^{-s_\delta}.(\frac{\hat{e}(C_2, w)}{\hat{e}(g_1, g_2)})^c$$

$$= (\hat{e}(C_2, g_2)^{r_x}.\hat{e}(v, w)^{-r_\alpha}.\hat{e}(v, g_2)^{-r_\delta}).(\hat{e}(C_2, g_2)^{x_i}.\hat{e}(v, w)^{-\alpha}.\hat{e}(v, g_2)^{-\alpha x_i}.\frac{\hat{e}(C_2, w)}{\hat{e}(g_1, g_2)})^c$$

$$= R_2((\hat{e}(C_2 v^{-\alpha}, w g_2^{x_i}))/\hat{e}(g_1, g_2))^c = R_2(\hat{e}(A_i, w g_2^{x_i})/\hat{e}(g_1, g_2))^c = R_2$$

To Prove $F_{root} = \hat{e}(C_2, w)$ we need to prove,

$$\hat{e}(\beta_{(0,i,j)}, D_{(0,j)}D_{(1,j)})/\beta_{(1,i,j)} = \hat{e}(A_i v^\alpha, g_2)^{q_{leaf_j}(0)}$$

This should be enough proof because when you calculate F_{root}, you are actually calculating the value of $q_{root}(0)$ using Lagrange interpolation. So if the tree is satisfied $q_{root}(0) = \gamma$, therefore $F_{root} = \hat{e}(A_i v^\alpha, g_2)^{q_{root}(0)} = \hat{e}(C_2, w)$. So we continue the proof as follows:

$$\hat{e}(\beta_{(0,i,j)}, D_{(0,j)}D_{(1,j)})/\beta_{(1,i,j)} =$$

$$\hat{e}(T_{i,j} v^\alpha, g_2^{q_{leaf_j}(0)/t_j + q_{leaf_j(0)}})/(\hat{e}(A_i^{t_j}, D_{(1,j)}).\hat{e}(v^\alpha, D_{(0,j)})) =$$

$$(\hat{e}(A_i^{t_j} v^\alpha, g_2)^{q_{leaf_j}(0)/t_j}\hat{e}(T_{i,j} v^\alpha, g_2)^{q_{leaf_j(0)}}) \div (\hat{e}(A_i^{t_j}, g_2^{q_{leaf_j}(0)})\hat{e}(v^\alpha, g_2^{q_{leaf_j}(0)/t_j}))$$

$$(\hat{e}(A_i, g_2)^{q_{leaf_j}(0)}\hat{e}(v^\alpha, g_2)^{q_{leaf_j}(0)/t_j}\hat{e}(T_{i,j}, g_2)^{q_{leaf_j(0)}}\hat{e}(v^\alpha, g_2)^{q_{leaf_j}(0)}) \div$$

$$(\hat{e}(A_i^{t_j}, g_2^{q_{leaf_j}(0)})\hat{e}(v^\alpha, g_2^{q_{leaf_j}(0)/t_j})) =$$

$$\hat{e}(A_i, g_2)^{q_{leaf_j}(0)}\hat{e}(v^\alpha, g_2)^{q_{leaf_j}(0)} = \hat{e}(A_i v^\alpha, g_2)^{q_{leaf_j}(0)}$$

In the verifying algorithm we check revoked users and revoked attributes before accepting a signature. In the signature we have $C_1 = \psi(\hat{u})^\alpha$ and $C_2 = A_i\psi(\hat{v})^\alpha$

for some random α. We reject a signature when either $(\hat{u}, \hat{v}, C_1, C_2/A_{revoked})$ or $(\hat{u}, \hat{v}, C_1, \beta_{(0,user,revoked)}/T_{user,revoked})$ is a co-Diffie Hellman tuple.

5.2 Traceability

Theorem 4. *If SDH is hard on groups (G_1, G_2) then the selective model of the Attribute Based Group Signature Scheme is said to be traceable under the random oracle with $Adv_{UTA} \leq (\varepsilon - 1/p)^2/16q_H$.*

To prove traceability, we construct a security model similar to the one in section 3.2. We use an input of $(q-1)$ SDH pairs as private key bases in the Setup of our model. We show how with the security model and the forking lemma(Theorem 2) we could find the q_{th} SDH pair. Therefore, we break Lemma 1 and solve the SDH problem (see Theorem 2). In the security model, in Appendix A, we added to the oracles queried, a hash oracle that represents H and H_0. If the adversary manages to create a forged signature once, then rewinding the game with changing the hash oracle responces as shown in Appendix A will imply that *Adam* with high probability could forge a new signature again.
A signature will be represented as $\langle M, \sigma_0, c, \sigma_1, \sigma_2 \rangle$. M is the signed message. $\sigma_0 = \langle r, gpk, C_1, C_2, R_1, R_2, R_3 \rangle$. c is the value derived from hashing σ_0. $\sigma_1 = \langle s_\alpha, s_x, s_\delta \rangle$ 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.
Using the two forged signatures from $\langle \sigma_0, c, \sigma_1, \sigma_2 \rangle$ and $\langle \sigma_0, \grave{c}, \grave{\sigma_1}, \sigma_2 \rangle$, we could extract a new SDH tuple. Let $\Delta c = c - \grave{c}$, $\Delta s_\alpha = s_\alpha - \grave{s}_\alpha$, and similarly for Δs_x, and Δs_δ.
Divide two instances of the equations used previously(See Theorem[5.1] proof) where one instance is with $\grave{c}$ and the other is with c to get the following:

- Dividing $C_1^c/C_1^{\grave{c}} = u^{s_\alpha}/u^{\grave{s}_\alpha}$ we get
  $u^{\tilde{\alpha}} = C_1$; where $\tilde{\alpha} = \Delta s_\alpha/\Delta c$

- Dividing $C_1^{s_x}/C_1^{\grave{s}_x} = u^{s_\delta}/u^{\grave{s}_\delta}$ will lead to
  $\Delta s_\delta = \tilde{\alpha}\Delta s_x$

- Dividing $(\hat{e}(g_1, g_2)/F_{root})^{\Delta c}$ will lead to
  $\hat{e}(C_2, g_2)^{\Delta s_x}\hat{e}(v, w)^{-\Delta s_\alpha}\hat{e}(v, g_2)^{-\tilde{\alpha}\Delta s_x} = (\hat{e}(g_1, g_2)/\hat{e}(C_2, w))^{\Delta c}$

Letting $\grave{x} = \Delta s_x/\Delta c$ we get $\hat{e}(g_1, g_2)/\hat{e}(C_2, w) = \hat{e}(C_2, g_2)^{\grave{x}}\hat{e}(v, w)^{-\grave{\alpha}}\hat{e}(v, g_2)^{-\grave{x}\grave{\alpha}}$ this could be rearranged as $\hat{e}(g_1, g_2) = \hat{e}(C_2 v^{-\grave{\alpha}}, w g_2^{\grave{x}})$. Let $\grave{A} = C_2 v^{-\grave{\alpha}}$ we get $\hat{e}(\grave{A}, w g_2^{\grave{x}}) = \hat{e}(g_1, g_2)$. Hence we obtain a new SDH pair $(\grave{A}, \grave{x})$ breaking Boneh and Boyens Lemma(See Section[1]). For more details of the proof see Appendix A.

5.3 ABGS Scheme Anonymity

Theorem 5. *If the decision linear assumption holds in group G_2 then the Attribute Based Group Signature Scheme is said to be anonymous under the random oracle with $Adv_{SAA} \leq \varepsilon$.*

For proving the previous theorem we construct a model similar to the one in section 3.2. We show in Appendix B how the existence of an adversary *Adam* that attacks the anonymity of our ABGS scheme, implies the existence of an adversary *Eve* that interacts with *Adam* in order to break the decision linear assumption in Definition 3. When it comes to the Challenge point in our security model *Eve* gives *Adam* a signature created using the inputs of the decision linear assumption $\langle u_0, u_1, u_2, u_0^a, u_1^b, Z\rangle$. *Adam* should guess an index of user at that point. The probability of guessing the user correctly depends on whether $Z = u_2^{a+b}$ or Z is random. So depending on *Adam*'s guess, *Eve* could solve the decision linear assumption. Details on how this is done are in the Appendix B.

6 Conclusion

In this paper we proposed an Attribute Based Group Signature Scheme, where verifying involves authenticating a person that belongs to a certain group and owns particular attributes. Our work is an extension of the scheme in [12] where a revocation algorithm has been added to attribute based group signatures. A revocation table is used to enable the verifiers to identify if a signer has been revoked or if any of his attributes have been revoked before accepting the signature as a valid one.
The scheme we propose is said to be anonymous and traceable under complex assumptions like the SDH and decision linear. One aspect of our proofs which has been left for future work is the fact that they rely on the random oracle model. Other modifications could be done to the scheme like increasing the anonymity level to include attribute anonymity. For example the verifier should be able to tell whether the signer satisfies the tree or not without knowing any of the attributes he owns. Efficiency always has scope for improvement and we will be working towards that in the future. We hope to achieve a scheme that has constant size keys and signatures, where the current ones have the size dependent on the number of attributes.
Our ABGS scheme allows us to include an extra feature, which is Revocation. This makes the scheme more practical because it imitates reality more closely. Afterall, in no practical scheme will the number of members remain unchanged in course of time.

References

1. G. Ateniese and G. Tsudik. Group signatures a la carte. In *ACM Symposium on discrete Algorithms*, pages 848–849, 1999.

2. G. Ateniese and G. Tsudik. Some open issues and new directions in group signatures. In *Financial Crypto'99*, volume 1648, pages 196–211, 1999.
3. G. Ateniese and G. Tsudik. Quasi-efficient revocation of group signatures, 2001.
4. 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.
5. 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.
6. 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.
7. D. Boneh and H. Shacham. Group signatures with verifier-local revocation. In *In proceedings of the 11'th ACM conference on Computer and Communications Security*, pages 168–177, 2004.
8. 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.
9. 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.
10. E.Bresson and J.Stern. Efficient revocation in group signature. In *In pro. PKC'01*, volume 1992, pages 190–206, 2001.
11. 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.
12. D. Khader. Attribute based group signature scheme. Cryptology ePrint Archive, Report 2007/159, 2007. `http://eprint.iacr.org/`.
13. A. Kiayias and M. Yung. Group signatures with efficient concurrent join. In *Proceedings of Eurocrypt 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 198–214. Springer-Verlag, 2005.
14. 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.
15. 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 223–242, 1998.
16. H. Petersen. How to convert any digital signature scheme into a group signature scheme. In *Proceedings of Security Protocols Workshop*, pages 177–190, 1997.
17. 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.
18. M. Trolin and D. Wikstrom. Hierarchical group signatures. In *Automata, Languages and Programming*, volume 3580, pages 446–458, 2005.

A Traceability

Theorem 6. *If SDH is hard on groups (G_1, G_2) then the selective model of the Attribute Based Group Signature Scheme is said to be traceable under the random oracle.*

Proof. In order to prove that we need three steps. Defining a security model for proving 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 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 Γ in which it would like to be challenged upon.

- **Setup:** *Challenger* runs the setup algorithm as in section [4] with a bilinear pair (G_1, G_2) with respective generators g_1, and g_2. It chooses randomly a value t_1, ..., t_m and a value γ. It calculates $w = g_2^{\gamma}$. The *Challenger* has to come up with the private key bases $gsk[i]_{base} = \langle A_i, x_i \rangle$ for an $1 \leq i \leq n$. 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. The *Challenger* creates a public key for the same attribute tree.
  So *Adam* is given $gpk = \langle g_1, g_2, w, D_{leaf_1}, ..., D_{leaf_\kappa} \rangle$ where $D_{leaf_j} = \langle D_{0,j}, D_{1,j} \rangle$ $= \langle g_2^{q_{leaf_j}(0)/t_{leaf_j}}, g_2^{q_{leaf_j}(0)} \rangle$, κ is the number of leafs and $1 \leq j \leq \kappa$. Along with the public key it picks random number of revocation tokens
  $R = \langle A_{revoke_1}, A_{revoke_2}, ..., T_{user_1, revoke_1}, T_{user_2, revoke_1}, ..., T_{user_1, revoke_2}, ... \rangle$
  where the size of the list should be reasonable enough to run the oracles in the next steps.

- **Hash queries:** When the *Challenger* asks *Adam* for the hash of $(gpk, M, r, C_1, C_2, R_1, R_2, R_3)$, *Adam* responds with a random element in G_1 and saves the answer just incase the same query is requested again. That represents the hash function H. When the *Challenger* asks *Adam* for the hash of (gpk, M, r), *Adam* responds with two random elements in G_2 and saves the answer.

- **Signature Queries:** *Adam* asks for a signature on a message M by a key index i and a set of attributes ζ that satisfy the tree. If $x_i \neq \star$ the *Challenger* follows the same signing procedure done in section[4].

  If $x_i = \star$ *Challenger* simulates a signature. It picks a random r from Z_p. It then gets $(\hat{u}, \hat{v}) = H_0(gpk, M, r)$. Sets $u \leftarrow \psi(\hat{u})$ and $v \leftarrow \psi(\hat{v})$. Picks a random α from Z_p.

  Challenger then calculates $C_1 = u^{\alpha}$ and $C_2 = A_i v^{\alpha}$. It randomly picks

c, s_δ, s_x, and s_α from Z_p. Calculates $R_1 = u^{s_\alpha}/C_1^c$, $R_3 = C_1^{s_x} u^{-s_\delta}$ and $R_2 = \hat{e}(C_2, g_2)^{s_x} \hat{e}(v, w)^{-s_\delta} \hat{e}(v, g_2)^{-s_\delta} . (\hat{e}(C_2, w)/\hat{e}(g_1, g_2))^c$. *Challenger* adds c to the list of the hash oracle H incase $(gpk, M, r, C_1, C_2, R_1, R_2, R_3)$ is queried later on. *Challenger* could set the values $CT_j = \langle \beta_{0,i,j}, \beta_{1,i,j} \rangle$ to equal $\langle A_i^{t_j} v^\alpha, \hat{e}(A_i^{t_j}, D_{1,j}) . \hat{e}(v^\alpha, D_{0,j}) \rangle$ where $j \in \zeta$.

Challenger returns the signature $\sigma = (r, C_1, C_2, c, s_\alpha, s_x, s_\delta, CT_1, ..., CT_\mu)$ to *Adam*.

- **Private Key Queries:** *Adam* issues a query for a private key by sending *Challenger* the index i, and a set Υ. If $x_i \neq \star$, *Challenger* responds back with $gsk[i] =< A_i, x_i, A_i^{t_1}, ..., A_i^{t_\mu} >$ otherwise *Challenger* fails and terminates the game.

- **Output:** If *Adam* is successful, he will output a forged signature $\sigma = (r, C_1, C_2, c, s_\alpha, s_x, s_\delta, CT_1, ..., CT_\mu)$ on a message M. C_1 and C_2 should not have any of the revocation list elements $A_{revoked}$ encoded in them and same goes for all CT_j in the signature. Let A^* be the value used in signing the forged signature. For $i = 1, ..., n$ check whether $\hat{e}(C_2/A_i, \hat{u}) = \hat{e}(C_1, \hat{v})$. If the equality holds then that implies that $A_i = A^*$. In that case check if $s_{i^*} = \star$ to output σ or otherwise declare failure. If the for loop goes through all the (A_i)s and there was no equality output σ.

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 (n)SDH 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.

Now we need to prove that the existence of any of the two forgeries contradicts with SDH assumption. For that we use the Forking Lemma (See Theorem[2]).

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. $\sigma_0 = \langle r, gpk, C_1, C_2, R_1, R_2, R_3 \rangle$. c is the value derived from hashing σ_0. $\sigma_1 = \langle s_\alpha, s_x, s_\delta \rangle$ 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.
We require *Adam* to query H_0 before H to ensure that by rewinding the game we could change values of $H(M, r, ..)$, while values of $H_0(M, r)$ should remain the same. Therefore the arguments u, v used in H remain unchanged too.
One simulated run of the adversary is described by the randomness string ω(used by *Adam* and *Challenger*),by the vector ℓ_0 of responses made by H_0 and by the vector ℓ of responses made by H. Let S be the set of tuple (ω, ℓ_0, ℓ) where *Adam* successfully forges the signature $(M, \sigma_0, c, \sigma_1, \sigma_2)$ and he queried H on (M, σ_0). Let $Ind(\omega, \ell_0, \ell)$ be the index of ℓ at which *Adam* queried (M, σ_0). Let $\nu = Pr[S] = \grave{\varepsilon} - 1/p$ where $1/p$ term represents the possibility that *Adam* guessed the hash of (M, σ_0) without quering it. For each χ, $1 \leq \chi \leq q_H$, let S_χ be a set of the tuple $(\omega, \ell_0, \ell)$where $Ind(\omega, \ell_0, \ell) = \chi$. Let Φ be the set of indices χ where $Pr[S_\chi|S] \geq 1/2q_H$ causing $Pr[Ind(\omega, \ell_0, \ell) \in \Phi|S] \geq 1/2$.
Let $\ell|_a^b$ be the restriction of ℓ to its elements at indices $a, a+1, ..., b$. For each $\chi \in \Phi$ consider the heavy row lemma (See Section[1]) with a matrix with rows indexed with$(\omega, \ell_0, \ell|_1^{\chi-1})$ 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 Ω_χ be the rows 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_0, ℓ) such that the $Pr_{\grave{\ell}}[(\omega, \ell_0, \grave{\ell}) \in S_j | \grave{\ell}|_1^{j-1} = \ell|_1^{j-1}] \geq \nu/(4q_H)$. 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_x, and Δs_δ.
Divide two instances of the equations used previously(See Theorem[5.1] proof) where one instance is with $\grave{c}$ and the other is with c to get the following:

- Dividing $C_1^c / C_1^{\grave{c}} = u^{s_\alpha} / \grave{s}_\alpha$ we get
  $u^{\tilde{\alpha}} = C_1$; where $\tilde{\alpha} = \Delta s_\alpha / \Delta c$
- Dividing $C_1^{s_x} / C_1^{\grave{s}_x} = u^{s_\delta} / u^{\grave{s}_\delta}$will lead to
  $\Delta s_\delta = \tilde{\alpha} \Delta s_x$
- Dividing $(\hat{e}(g_1, g_2)/F_{root})^{\Delta c}$ will lead to
  $\hat{e}(C_2, g_2)^{\Delta s_x} \hat{e}(v, w)^{-\Delta s_\alpha} \hat{e}(v, g_2)^{-\tilde{\alpha} \Delta s_x} = (\hat{e}(g_1, g_2)/\hat{e}(C_2, w))^{\Delta c}$

Letting $\grave{x} = \Delta s_x / \Delta c$ we get $\hat{e}(g_1, g_2)/\hat{e}(C_2, w) = \hat{e}(C_2, g_2)^{\grave{x}} \hat{e}(v, w)^{-\grave{\alpha}} \hat{e}(v, g_2)^{-\grave{x}\grave{\alpha}}$ this could be rearranged as $\hat{e}(g_1, g_2) = \hat{e}(C_2 v^{-\grave{\alpha}}, w g_2^{\grave{x}})$. Let $\grave{A} = C_2 v^{-\grave{\alpha}}$ we get $\hat{e}(\grave{A}, w g_2^{\grave{x}}) = \hat{e}(g_1, g_2)$. Hence we obtain a new SDH pair $(\grave{A}, \grave{x})$ breaking Boneh and Boyens Lemma(See Section[1]). Now putting things together we get the following theorems:

Theorem 7. *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 8. *We could solve an instance of* n *SDH with a probability* $(\varepsilon/n - 1/p)^2/16q_H$ *using a Type-II forger Adam*

B ABGS Scheme Anonymity

Theorem 9. *If the decision linear assumption holds in group* G_2 *then the Attribute Based Group Signature Scheme is said to be anonymous under the random oracle.*

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

- **Init:** *Adam* decides the attribute tree Γ he would like to be challenged upon and gives it to *Eve*.

- **Setup:** *Challenger* gives *Eve* the tuple $\langle u_0, u_1, u_2, h_0 = u_0^a, h_1 = u_1^b, Z\rangle$ where $u_0, u_1, u_2 \in G_2$ and $a, b \in Z_p$. Z is either random or $Z = u_2^{a+b}$. *Eve* should decide which Z it was given. Recall that g_1, g_2 are in G_1 and G_2 respectively. *Eve* chooses a random γ from Z_p. *Eve* also chooses $t_1, ..., t_\kappa$ for attributes of the tree. *Eve* assigns $w = g_2^\gamma$. She creates the $n-2$ private key bases $gsk[i]_{base} = \langle A_i, x_i\rangle$ as in section[4]. She will then choose a random $W \in G_2$. The missing private key bases of user i_0 and i_1 will be defined as $A_{i_0} = ZW/u_2^a$ and $A_{i_1} = Wu_2^b$ for some x_{i_0},x_{i_1}. Notice that $A_{i_0} = A_{i_1}$ when $Z = u_2^{a+b}$. *Eve* does not know the values of either $gsk[i_0]_{base}$ or $gsk[i_1]_{base}$. We will show later in our security model how she could still interact with *Adam* pretending she does know them. *Eve* also create a public key for the tree structure $gpk = \langle g_1, g_2, w, D_{leaf_1}, ..., D_{leaf_\kappa}\rangle$.

- **Phase 1:** *Eve* runs four oracles a signature oracle, a private key oracle, revocation oracle and a hash oracle. If *Adam* queries the hash oracle *Eve* should keep a list of her responces to insure randomness and consistancy for both hash functions H and H_0. In the rest of the oracles *Eve*'s reaction will be divided into three depending whether *Adam* queried i_0,i_1 or neither.

  If *Adam* queries the signature oracle he should send an index i, a set of attributes ζ that satisfy the tree and a message M. If $(i \neq i_0, i_1)$; *Eve* will reply with a signature $\sigma = \langle r, C_1, C_2, c, s_\alpha, s_x, s_\delta, CT_1, ..., CT_\mu\rangle$ as done in section[4]. If $(i = i_0)$, *Eve* picks a random $s, t, l \in Z_p$ and makes the following assignments:

$C_1 = h_0 u_0^s$; $C_2 = ZW u_2^s h_0^t u_0^{st}$; $\hat{u} = u_0^l$; $\hat{v} = (u_2 u_0^t)^l$.
Let $\alpha = (a+s)/l \in Z_p$. Then $C_1 = \hat{u}^\alpha$ and $C_2 = A_{i_0}\hat{v}^\alpha$. If $(i = i_1)$, *Eve* picks a random $s, t, l \in Z_p$ and makes the following assignments:
$C_1 = h_1 u_1^s$; $C_2 = W h_1^t u_1^{st} / u_2^s$; $\hat{u} = u_1^l$; $\hat{v} = (u_1^t/u_2)^l$
Let $\alpha = (b+s)/l \in Z_p$. Then $C_1 = \hat{u}^\alpha$ and $C_2 = A_{i_1}\hat{v}^\alpha$.
So either case the values of $C_1 = \hat{u}^\alpha$ and $C_2 = A_i\hat{v}^\alpha$ for some α. *Eve* now assigns $\beta_{(0,i,j)} = C_1^{t_j}$, and $\beta_{(1,i,j)} = \hat{e}(\beta_{(0,i,j)}, D_{1,j})$. She chooses random values $r, c, s_\alpha, s_x, s_\delta$ from Z_p. *Eve* sets the values $R_1 = u^{s_\alpha}/\psi {C_1}^c$, $R_2 = \hat{e}(\psi C_2, g_2)^{s_x}\hat{e}(\psi(\hat{v}), w)^{-s_\alpha}\hat{e}(\psi(\hat{v}), g_2)^{-s_\delta}(\hat{e}(\psi C_2, w)/\hat{e}(g_1, g_2))^c$, and finally $R_3 = \psi(C_1)^{s_x}\psi(u)^{-s_\delta}$.
The probability that $H(gpk, M, \psi(C_1), \psi(C_2), R_1, R_2, R_3)$ or $H_0(gpk, M, r)$ have been queried before is at most q_H/p where q_H is the numbers of queries. If a collusion happens *Eve* reports a failure. Otherwise we add both to the list of the hash oracle such that $H(gpk, M, \psi(C_1), \psi(C_2), R_1, R_2, R_3) = c$ and $H_0(gpk, M, r) = (\hat{u}, \hat{v})$
Eve sends back the signature $\sigma = \langle r, \psi(C_1), \psi(C_2), c, s_\alpha, s_x, s_\delta\rangle$

When *Adam* issues a query on the private key oracle he needs to send *Eve* an attribute set Υ and an index i. *Eve* responds back with $\langle A_i, x_i, A_i^{t_1}, ..., A_i^{t_\mu}\rangle$. If *Adam* queries i_0, i_1, *Eve* reports failure.

Finally when querying the revocation oracle *Adam* either sends a users index i alone or sends it together with the attribute he wants to revoke. *Eve* replies with either A_i or $T_{i,j}$ for the revocation queries with maintaining the table of revocation R. If *Adam* queries i_0, i_1, *Eve* reports failure.

- **Challenge:** *Adam* asks to be challenged on message M, attribute set ζ and indexes i_0^* and i_1^*. If $\{i_0^*, i_1^*\} \neq \{i_0^*, i_1^*\}$ then *Eve* reports failure. Otherwise, *Eve* picks randomly $b \in \{0, 1\}$ and generates a signature the same way it would have done in the signature query. So *Eve* responces back with a signature σ_b.

- **Phase 2:** Is exactly like phase 1.

- **Output :** *Adam* outputs a guess $\grave{b} \in \{0, 1\}$. If $b = \grave{b}$ then Z is random, otherwise $Z = u_2^{a+b}$.

There are two ways this game could end. Case one is when *Eve* does not abort. If Z is random then $Pr[b = \grave{b}] > 1/2 + \varepsilon$ otherwise if $Z = u_2^{a+b}$ then both signatures should be identical and therefore challenge is independent of b hence $Pr[b = \grave{b}] = 1/2$. So the advantage of *Eve* solving the linear challenge is at least $\varepsilon/2$.
The second case is *Eve* aborts and fails. *Eve* could abort in the signature queries with probability $q_S q_H/p$ where q_S is the number of signature queries and q_H are hash queries. The probability that all queries in phase 1 and the challenge do not

cause Eve to abort is $1/n^2$. Concatenating both cases together the probability of Eve solving the linear challenge is $(\varepsilon/2)((1/n^2) - (q_S q_H)/p)$ as required.