Authenticating with Attributes

Abstract. An Attribute Based Authentication Scheme(AAS) is a new generation of group signatures. In such a scheme the characteristics of the signer are decided by the verifier and these requirements are sent to the group. Then any group member with enough credentials can sign on behalf of the whole group. AAS needs to be fully anonymous so that the verifier can not identify the signer. It should also be fully traceable so that a certain authority can revoke the anonymity in case of a dispute. Finally, the scheme should be attribute verifiable. This means when obtaining an attribute a group member can authenticate the attribute authority thus making it impossible for malicious authorities to attack the system. In this paper we define the algorithms and protocols used by the scheme as well as introducing the required security notions. We finally give a secure construction of an AAS scheme.

1 Introduction

An Attribute based Authentication Scheme is a new paradigm in cryptography and is motivated by the need to authenticate person's credentials. The following is a scenario were such a scheme is needed:
Bob has a company and Alice wants a signature from any employee as long as that employee proves to be a senior manager in department A or a manager (senior/junior) in department B. We would refer to such a scheme as the Attribute Authentication Scheme(AAS) and we would like it to include the following properties:

- No previous knowledge assumption: We can not assume that Alice and the employee know each other from before.
- Unforgeable : No employee can forge a proof of possession of an attribute. A none employee must not be able to forge a signature.
- Anonymous: Alice can not identify the employee from the signature.
- Unlinkable: Alice is unable to determine if two signatures originate from the same employee.
- Traceable: In case of a dispute Alice contacts the manager (Bob) and asks him to trace a signature. Bob should be able to revoke the anonymity of the signer and confirm to Alice that the signature is valid.
- Privacy: Alice does not need to know how the policy has been satisfied. In particular she does not need to know the department or level of management of the signer.
- Coalition Resistant: If an employee is in department A but is not a senior manager and another employee is in department C and is a senior manager, the two employees can not sign the message jointly.

– Separability: Each department in the company gives out attributes under its control independently of all other departments.

A possible solution to this problem is to use identity based signatures (IBS) which are digital signatures that use the identity of the person in the verification process rather than a public key [12]. Such a scheme can be considered as the first attempt to create Attribute Based Signatures. Assume we concatenate identities in IBS schemes to some attribute templates. Then this automatically ensures the scheme is coalition resistant. However the verifier will need the identity of the signer in order to create his verification request. Consequently the signer is forced to attach their identity to the signature which leads to breaking the anonymity and linkability of the scheme.
To overcome the problem of anonymity group signatures can be used [7]. Group signatures are digital signatures which allow any member of a group to sign anonymously on behalf of the whole group. We can insist the members of any group all possess a certain attribute, for example all members of a given group may belong to department A. Even though group signatures solve this problem, it requires the number of verifications to be equal to the number of attributes needed from the verifier. In other words an employee in Bob's company has to enclose all certificates Alice needs which might break anonymity. It is also vulnerable to coalition resistant among different groups.
Using anonymous credential systems (also known as pseudonym system) is one suggestion to rectify this predicament. Such systems were first proposed by Chaum in [6]. The scenario introduced then was having multiple users who anonymously can transfer credentials from one organization to another. In [5] a practical system was proposed which allowed users to obtain credentials from trusted organization(s). They defined a protocol that proves possession of a credential while ensuring the signer remains anonymous. Additionally they provide optional anonymity revocation depending on the transaction. Anonymous credential systems appear to cover a lot of the properties we require in our Attribute Authentication Scheme. However, they are missing some core properties. Anonymous Credential Scheme involves a "one credential proof" rather then a "multiple credential proof". This implies that the verifier needs to run the verification algorithm as many as the number of credentials needed. It also means that the privacy of the signer is breached because the attributes are enclosed explicitly. Proving multiple credentials also means we require a way to guarantee that it is coalition resistant, and even though some anonymous credential systems have anonymity revocation, this is dependent on the transaction. We need a system that is always traceable.
The shortcoming of all previous suggestions makes AAS a new cryptographic problem that requires the creation of a new scheme. An Attribute Authentication Scheme consists of four entities; a central authority (Bob), an attribute authority (department), a signer (employee) and a verifier (Alice). The Central authority will publish one public key base for the group (Step 1 Figure 1) and this will be used in verifying a signature. Central authority creates many pairs of private key bases and registration keys to be used in signing messages. Each

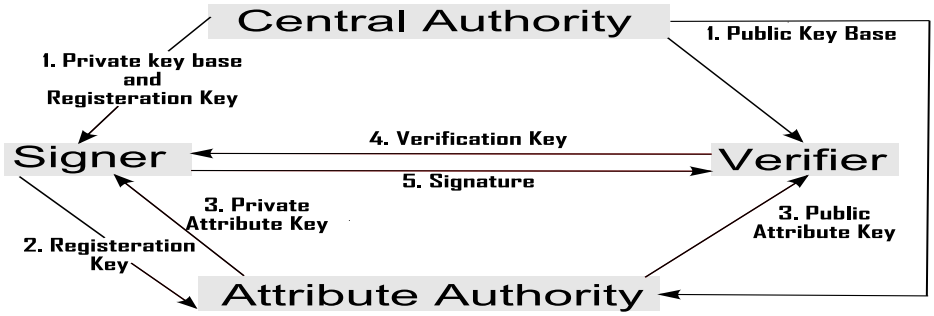

Fig. 1. Attribute Authentication Scheme

member (possible signer) of the group will be given their unique pair (i.e. Bob gives each employee a pair as shown in Step 1 Figure 1). Each member will be using the registration key to register with an attribute authority (Step 2 in Figure 1). Registering with an authority implies getting a copy of a private attribute key (Step 3 in Figure 1). A set of private attribute keys will then be used in signing a message. The public key base created earlier will be used by the attribute authority to create an attribute public key (Step 3 in Figure 1). Verifier (Alice) will create a verification key using the public key base and as much public attribute keys as needed (Step 4 in Figure 1). The verification key describes what attributes the verifier would like the signer to possess. It will also be used in both signing and verifying processes. An employee with such qualification (i.e has enough private keys) can then sign a message (Step 5 in Figure 1). The system can have many attribute authorities but has to have one central authority. The central authority can be an attribute authority itself.
In this paper we define an AAS scheme (Section 2) and the security notions required. We then construct an example in section 5. In section 6 we explain how attribute keys can be verified so that no malicious attribute authority can break the security of the system. Finally we conclude our result in section 7.

2 Definitions

An Attribute Authentication Scheme (AAS) contains a suite of algorithms and protocols that are executed by the entities (central authority, attribute authority, signer and verifier). To describe the scheme we define the following algorithms and protocols:

- **Setup**(k): This algorithm is run by the central authority, it takes a security parameter k as an input and outputs two sets of parameters, S_{pri} and S_{pub}. The system parameters S_{pri} is kept to the authority, while S_{pub} is published for all to see and use.
- **M.KeyGen**$(\mathbf{S_{pri}}, \mathbf{S_{pub}}, \mathbf{n})$: This algorithm is run by the central authority. It takes the inputs S_{pri} and n, where n is the maximum bound of members

in the group and generates n private key bases $bsk[i]$ and registration keys A_i. Then S_{pub} and S_{pri} are used to generate a public key base w, which is distributed to the members of the group. The public key base w is public to all, the $bsk[i]$ is kept private to the user, while the A_i is given to trusted third parties, as shown in $A.KeyGen_{pri}$.

- $\mathbf{A.KeyGen_{pub}}(\mathbf{S_{pub}}, \mathbf{w})$: This algorithm is run by the attribute authorities. Each authority is responsible for one or more attributes and for attribute j, the authority to which it belongs to contains a master key t_j. It is this key which is used to create the attribute-related keys. Using the master keys, the authority creates public keys bpk_j representing the attributes it supports. Only the attribute authorities can produce such keys since it requires the knowledge of t_j.
- $\mathbf{A.KeyGen_{pri}}(\mathbf{A_i}, \mathbf{j}, \mathbf{RL})$: This algorithm is run by the attribute authorities. In this algorithm member i registers with his key A_i to obtain a special private key $T_{i,j}$. Before this happens the attribute authority checks the user against a list RL, where RL contains all registration keys A_i of all users which have been revoked. If user has not been revoked, the private attribute key is calculated using the attribute authority's master key t_j. Member i will be then using his private key $usk[i] = \langle bsk[i], A_i, T_{i,1}, ..., T_{i,\mu} \rangle$ to sign. We should point out that not all $T_{i,j}$ have to be generated by the same attribute authority and that each signature will have a different set of $T_{i,j}$ depending on the verifier's request.
- $\mathbf{Verifign}(\mathcal{U}_\mathbf{i}(\mathbf{usk[i]}, \mathbf{M}), \mathcal{V}(\mathbf{M}, \bar{\mathbf{B}}, \mathbf{w}))$: This stage is a protocol between the verifier $\mathcal{V}$ and the signer $\mathcal{U}_i$. The verifier takes as an input $\bar{B} = (bpk_1, ..., bpk_m)$ and a message M to create a verification key $\bar{D}$ to be sent to the group. The signer uses his private key $usk[i]$, the message M and $\bar{D}$ as inputs to an algorithm **Sign**, where $Sign$ creates a signature σ to verify the use of $\bar{D}$. $\mathcal{U}_i$ sends σ to $\mathcal{V}$ and the verifier runs an algorithm **Verify** using σ, M, $\bar{D}$ and w as inputs. $Verify$ checks the validity of the signature and whether or not the signer is revoked.
- $\mathbf{Revoke}(\mathbf{A_i}, \mathbf{RL})$: This algorithm is run by the central authority. It adds a registration key A_i to the revocation list RL.
- $\mathbf{ChkRvk}(\sigma, \mathbf{RL})$: This algorithm takes a signature σ and a revocation list RL. It returns i if user is revoked and on the list otherwise it returns -1. Number i in this case is just an index in the revocation list and does not refer to any member.
- $\mathbf{Open}(\sigma)$: This algorithm is meant for tracing a signer in case of a dispute and revoking his anonymity. The central authority stores the value A_i for all members of the groups and adds all members (while maintaining indexes) to a fake revocation list (FRL). It runs $ChkRvk(\sigma, FRL)$ which returns an index i. Since all members belong to FRL the value of i must reveal the identity of the signer. $Open$ is not a totally new algorithm because it uses $ChkRvk$ on a fake list FRL.

AAS Security Notions: Like any other cryptographic scheme before we prove security of our AAS scheme we have to ensure correctness.

Definition 1. (Correctness of an AAS scheme): *For all keys* $usk[i]$ *generated correctly, every signature generated by a member* i *verifies as valid unless the user has been revoked or the member did not have enough attributes. In other words,* $Verify(\bar{D}_v, w, M, RL, Sign(M, \bar{D}_s, usk[i])) = valid;$
where $ChkRvk(Sign(M, \bar{D}, usk[i]), RL) = -1$ *and* $\bar{D}_s = \bar{D}_v;$
Otherwise $Verify(...) = not\ valid;$

The next stage is to define adversarial models that would include security notions mentioned in section 1. In the adversarial model we assume we have an adversary that interacts with a hypothetical challenger. We refer to the adversary as *Adam* and the challenger as *Charles*. We assume that the attribute authority is corrupted and we give the adversary the ability to choose the master keys he would like to be challenged on. Two adversarial models are enough to cover all security notions. We refer to them as the full anonymity model and full traceability model, the two models allow *Adam* to query certain oracles run by *Charles*. Before we define the adversarial models list the oracles:

- **USK Oracle**: To query this oracle *Adam* sends *Charles* an index i in order to find the user's private key base $bsk[i]$ and registration key A_i. *Charles* will send the pair $(bsk[i], A_i)$ to *Adam*. Note that since *Adam* is given the ability to choose the attribute master keys he would like to be challenged on, he can create private attribute keys from whatever private key bases he has.
- **Signature Oracle**: To query this oracle *Adam* sends *Charles* a verification key he created $\bar{D}$, a message M and an index i. *Adam* requires the signature of member i on message M using $\bar{D}$. *Charles* sends σ
- **Revoke Oracle**: To query this oracle *Adam* sends *Charles* and index i. *Charles* adds A_i to the list RL. The list RL is public and *Adam* can access it.

Adam initializes both game models by deciding the universal set of attributes U in which he would like to be challenged upon. Assume U is of size m and contains a list of master keys t_j that will be used. Both *Charles* and *Adam* have access to U.
(AAS Full Anonymity): We say that an AAS Scheme is fully anonymous under a specific set of attributes, if no polynomially bounded adversary *Adam* has a non-negligible advantage against the challenger *Charles* in the following game:

- **Setup:** *Charles* will play the role of the central authority. He will run the algorithms *Setup*, and $M.KeyGen$ to produce the systems S_{pub}, and S_{pri}. *Charles* also will generate the n private key bases $bsk[i]$ and n registration keys A_i. He will send to *Adam* S_{pub}. Finally, both *Adam* and *Charles* will generate m public attribute keys $\langle bpk_1,...,bpk_m \rangle$.
- **Phase(1):** *Charles* will run three oracles, USK, a signature oracle, and a revoke oracle.
- **Challenge:** *Adam* asks to be challenged on a message M, two indexes i_0,i_1, and verification key $\bar{D}$. *Charles* responds back with a signature σ_b, where

$b \in \{0,1\}$. The signer can be either i_0 or i_1. Both i_0, i_1 should not have been queried in the revoke oracle or USK oracle, however it can be queried on the sign oracle.
- **Phase(2):** This stage is similar to Phase 1. Except that (i_0, i_1) should not be sent to the revoke nor the USK oracles.
- **Output:** *Adam* outputs a guess $\bar{b} \in \{0,1\}$. If $\bar{b} = b$, *Adam* wins the game.

The adversary has been given strong attack capabilities by getting access to oracles such as USK, revoke and sign. He is not allowed to query both i_0, and i_1 in the revoke oracle or the USK oracle because that will give away the identities. If one of them is revoked or if he has the private key base then, using $ChkRvk$ in the first case, *Adam* can distinguish the signature and identify the signer. However *Adam* can query signatures of (i_0, i_1) and still his advantage in guessing the signer should be negligible. This implies that the signatures can not be linked and the advantage of wining the game is $Adv_A(n,k) = |Pr[b = \bar{b}] - 1/2|$, where n is the maximum bound of members and k is the security parameter used for setting up the system.

Definition 2. Full Anonymity*: A scheme is fully anonymous if for all polynomial time adversary Adam, $Adv_A(n,k) < \varepsilon$ and ε is negligible.*

(AAS Full Traceability): We say that our AAS scheme is traceable if no polynomially bounded adversary *Adam* has a non-negligible advantage against the challenger *Charles* in the following game:

- **Setup:** *Charles* will play the role of the central authority as done in the setup stage of the anonymity game model mentioned earlier. He will run the algorithms *Setup*, and *M.KeyGen* to produce S_{pub}, and S_{pri}. *Charles* also will generate n private key bases $bsk[i]$ and n registration keys A_i. S_{pub} will be sent to *Adam* and *Adam* is given all registration keys A_i. Finally, both *Adam* and *Charles* will generate m public attribute keys $\langle bpk_1,...,bpk_m \rangle$.
- **Queries:** *Charles* runs two oracles: USK oracle, and signature oracle. *Adam* queries them as described earlier.
- **Output:** *Adam* asks to be challenged on a M which he sends to *Charles*. *Charles* calculates a random $\bar{D}$ and sends it back to *Adam*. *Adam* replies with a signature σ. *Charles* verifies the signature. If it is not valid return 0. If it turns up to be a valid signature *Charles* tries tracing it to a signer. If it traces to a signer in which *Adam* did not query before or if it traces to a nonmember then *Adam* wins the game. *Charles* returns 1 if *Adam* wins else 0 is returned.

Note that *Adam* does not have to query the revoke oracle. He has a more powerful capability and that is the list of all registration keys. *Adam* can run the revoke algorithm himself. Recall full traceability includes unforgeability. One can reduce the challenge to produce a valid pair of message and signature, where the message was not queried in phase 1. The adversarial model with such reduction is the definition of unforgeability. Therefore full traceability implicitly proves unforgeability. Additionally a strong formalization of coalition resistance is obtained in

this game since this can be defined with a similar game where the adversary is not given the registration keys. If we refer to the returned value as Exp then the advantage of winning the game is notated as $Adv_T(n,k) = Pr[Exp = 1]$.

Definition 3. Full Traceability *A scheme is fully traceable if for all polynomial time adversary Adam,* $Adv_T(n,k) < \varepsilon$ *and* ε *is negligible.*

Full traceability and full anonymity are introduced with assumption that the attribute authority is dishonest and they are meant to protect the signer and the verifier from each other. However in real life we would also require a way of protecting the signer and verifier from malicious attribute authorities. To achieve this we would add a third security notion which we discuss in section 6.

3 Attribute Tree

An attribute tree is the structure used to present the verifier's request and will be used in creating the verification key $\bar{D}$. An attribute tree was used in [8] to implement an attribute based encryption scheme. It is a tree in which each interior node is a threshold gate and the leaves are linked to attributes. A threshold gate represents m of n children branching from the current node which need to be satisfied for the parent to be considered satisfied. Satisfaction of a leaf is achieved by owning an attribute. For further explanation, consider the example in Figure 2 that demonstrates the scenario in section 1. Γ will be used as a

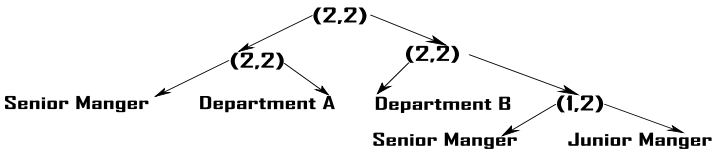

Fig. 2. Attribute Tree

description to our attribute tree. For example, to represent the tree in Figure 2, Γ = {(2, 2),(2, 2), (2, 2), Senior Manager, Dept. A, Dept. B, (1, 2), Senior Manager, Junior Manager }, where (m, n) represents a threshold gate m of n. The description Γ is representing the tree in a Top-Down-Left-Right manner. Let κ be the number of leaves in Γ and Υ_i be the set describing the private keys owned by a member. For example, if Smith is a senior manager and works in Department A, Υ_{Smith} = {Senior, Dept. A }. The size of Υ_i is represented by μ. Consider a bilinear map(Appendix 7) $e : G_1 \times G_2 \rightarrow G_3$ defined on groups of prime order p with generators $A_i \in G_1$ and $w \in G_2$.
Recall the master key of an attribute j is t_j. Then every public key of an attribute equals $bpk_j = w^{t_j}$, where w represents the public key base and every

private key of the attribute equals $T_{i,j} = A_i^{1/t_j}$, where A_i also represents the registration key of member i (i.e. employee in Bob company).
Alice builds the verification key. She starts with deciding the structure of the tree as shown in Figure 2. She randomly assigns all nodes (other than the root an index) and adds the indexes to Γ. We use the notation $Index(Node)$ to represent the index of a node. Alice chooses randomly an element $\alpha \in \mathbb{Z}_p$. Starting from the root she chooses polynomials q_{node} over $\mathbb{Z}_p$ where all polynomials are of degree $d_{node} = k_{node} - 1$, and k_{node} is the threshold gate of a node. Assign $q_{root}(0) = \alpha$ while the rest of the nodes $q_{node}(0) = q_{parent}(Index(Node))$. Notice that given enough q_{node} and the indexes, one can calculate α with Lagrange interpolation(see appendix 9). However, α will be Alice's one time secret that will help her verify the tree is satisfied. Alice will calculate $D_j = bpk_j^{q_j(0)}$ for all leaves and she will send $\langle \Gamma, D_1, ..., D_\kappa \rangle$ to Bob's company. An employee then decides from Γ and Υ_i which private keys he can use. He chooses a random element $\beta \in \mathbb{Z}_p$. Given he has enough attributes he can run a recursive function $Sign_{Node}$ as follows; If the node we are currently on is a leaf in the tree the algorithm returns the following:

$$Sign_{Node}(leaf) = \begin{cases} \text{If } (j \in \Gamma); \text{ return } e(T_{i,j}^{\beta}, D_j) \\ \text{Otherwise return } \perp \end{cases}$$

For a node ρ which is not a leaf the algorithm proceeds as follows: For all children z of the node ρ it calls $Sign_{Node}$ 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 set exist return $\perp$. Otherwise let

$$\Delta_{\hat{S}_\rho, index(z)} = \prod_{\iota \in \{index(z): z \in \hat{S}_\rho - index(z)\}} (-\iota/(index(z) - \iota))$$

and compute

$$F_\rho = \prod_{z \in \hat{S}_\rho} F_z^{\Delta_{\hat{S}_\rho, index(z)}} = \prod_{z \in \hat{S}_\rho} e(A_i^\beta, w)^{q_z(0) \cdot \Delta_{\hat{S}_\rho, index(z)}} = e(A_i^\beta, w)^{q_\rho(0)}$$

Calculate F_{root} then if the tree is satisfied $F_{root} = e(A_i^\beta, w)^\alpha$. An employee in Bob's company will then send $\langle F_{root}, A_i^\beta \rangle$ to Alice. Given that Alice knows α, A_i^β and w, she can verify that the employee satisfies the tree by verifying $F_{root}^{1/\alpha} = e(A_i^\beta, w)$. Notice forging the pair $\langle F_{root}, A_i^\beta \rangle$ is as hard as the BDH problem (See appendix A.1). The solution is not complete since we prove satisfaction of the tree only. Note we have not proven they belong to a valid signer within the group.
We will use the notation $\bar{D} = TCreate(\Gamma, \alpha, \bar{B})$ to represent that verification key $\bar{D}$ was created by running the previous algorithm on a set of public attribute keys $\bar{B} = (bpk_1,...,bpk_\kappa)$ where the $q_{root}(0) = \alpha$ and the tree structure is described in Γ. In this case $\bar{D} = (D_1,...,D_\kappa)$. We will use the notation $F_{root} = TVerify(\bar{D}, \Upsilon_i)$ to imply that F_{root} is calculated using enough private attribute keys from Υ_i that verify with $\bar{D}$.
In the following section we will introduce the preliminaries needed to construct the scheme. In section 5 we will construct the first attribute authentication

scheme. We will also explain further how the algorithm $TCreate$ is used together with group signature to create an AAS scheme.

4 Complexity Assumptions

The construction to be given in section 5 and its security rely on two main complexity assumptions: the q-Strong Diffie Hellman Problem and the Decision Linear Problem. In this section define them and begin with q-Strong Diffie Hellman Problem(qSDH).

Definition 4. (q-Strong Diffie-Hellman Problem [1])
Let G_1, G_2 be cyclic groups of prime order p, with a computable isomorphism ψ or possibly $G_1 = G_2$. Assuming the generators $g_1 \in G_1$, and $g_2 \in G_2$. 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 $(g_1^{1/(\gamma+x)}, x)$ where $x \in \mathbb{Z}_p^*$. 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 $\gamma \in \mathbb{Z}_p^*$ and of random bits of A.

This problem is considered hard to solve in polynomial time and ε should be negligible [1]. The q-Strong Diffie Hellman Problem is the complexity assumption underlying the full traceability security notion. Full anonymity is based on the Decision Linear Problem defined as follows:

Definition 5. (Decision Linear Problem in G_1 [2])
Let G_1 be a group of prime order p and u, v, h be generators in that group. Given $u, v, h, u^a, v^b, h^c \in G_1$ as an input, where $a, b, c \in \mathbb{Z}_p$, it is hard to decide whether or not $a + b = c$.

We have defined the main complexity assumptions we require. In the following section we will construct our AAS scheme.

5 Construction of our AAS Scheme

In this section we will construct our AAS scheme based on work by Boneh and Shacham [4]. We now go through the algorithms defined in section 2 and show how we can build them.

- **Setup**(k): Consider a bilinear pair $e : G_1 \times G_2 \rightarrow G_3$ where (G_1, G_2) have a computable isomorphism ψ and all three groups G_1, G_2 and G_3 are multiplicative and of prime order p. Suppose that the SDH assumption holds on (G_1, G_2) and the decision linear assumption holds on G_2. Select a hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_p$. Select a hash function H_0 with respected range G_2^2. Choose a generator $g_2 \in G_2$ at random and then set $g_1 \leftarrow \psi(g_2)$. Pick a random γ from $\mathbb{Z}_p$. Then let $S_{pub} = \langle G_1, G_2, G_3, e, H, H_0, g_1, g_2 \rangle$ and $S_{pri} = \gamma$.

– **M.KeyGen**($\mathbf{S_{pri}}, \mathbf{n}$) : Using γ generate for each user $i, 1 \leq i \leq n$, a private key base $bsk[i] = \langle A_i, x_i \rangle$. All $bsk[i]$ should be SDH pairs, where x_i is chosen randomly from $\mathbb{Z}_p^*$ and $A_i = g_1^{1/(\gamma + x_i)} \in G_1$. Let A_i be the registration key and compute the public key base as $w = g_2^{\gamma}$.

– **A.KeyGen$_{pub}$**($\mathbf{S_{pub}}$): The public key for attribute j is $bpk_j = w^{t_j} = g_2^{\gamma t_j}$, where t_j is chosen randomly from $\mathbb{Z}_p$

– **A.KeyGen$_{pri}$**($\mathbf{A_i}, \mathbf{j}, \mathbf{RL}$): User i wants to register attribute j. It contacts the attribute authority in charge, which in turn checks the revocation list RL. If the member i is not on the list the authority calculates $T_{i,j} = A_i^{1/t_j}$ and gives this information to user i. The private key for a user i will then be the tuple $usk[i] = \langle A_i, x_i, T_{i,1},...,T_{i,\mu} \rangle$.

– **Verifign**($\mathcal{U}_\mathbf{i}(\mathbf{usk[i]}, \mathbf{M}), \mathcal{V}(\mathbf{M}, \bar{\mathbf{B}})$) : The protocol runs as follows:

1. $\mathcal{V}$ has collected all public key bases it needs $\bar{B} = (bpk_1, ..., bpk_\kappa)$. $\mathcal{V}$ then chooses a random element $\alpha \in \mathbb{Z}_p$ and decides a Γ. Finally, $\mathcal{V}$ calculates $\bar{D} = TCreate(\Gamma, \alpha, \bar{B}) = (D_1, ..., D_\kappa)$ (section 3) and sends to $\mathcal{U}_i$.
2. $\mathcal{U}_i$ calculates $F_{root} = TVerify(\bar{D}, \Upsilon_i)$ as shown in section 3 then randomly selects an $r \in \mathbb{Z}_p$ to obtain $(\bar{u}, \bar{v}) \leftarrow H_0(\bar{D}, M, r)$. Next $\mathcal{U}_i$ computes the images $u \leftarrow \psi(\bar{u})$ and $v \leftarrow \psi(\bar{v})$. and chooses ξ, β, r_ξ, r_x, and r_δ randomly from $\mathbb{Z}_p$.
   Following on from this $\mathcal{U}_i$ computes $C_1 = u^\xi$, $C_2 = A_i v^\xi$, $C_3 = e(v^\xi, w)^\beta$ and $C_4 = w^\beta$.
   Let $\delta = x_i \xi$, $s_\xi = r_\xi + c\xi$, $s_x = r_x + c x_i$ and $s_\delta = r_\delta + c\delta$.
   Then compute $R_1 = u^{r_\xi}$; $R_3 = C_1^{r_x} u^{-r_\delta}$.
   $R_2 = e(C_2, g_2)^{r_x} e(v, w)^{-r_\xi} e(v, g_2)^{-r_\delta}$
   Compute $c = H(M, r, C_1, C_2, C_3, C_4, R_1, R_2, R_3)$,
   Finally, $\mathcal{U}_i$ sends $\sigma = (r, C_1, C_2, C_3, C_4, c, s_\xi, s_x, s_\delta, F_{root})$ to $\mathcal{V}$.
3. $\mathcal{V}$ calculates $(\bar{u}, \bar{v}) \leftarrow H_0(\bar{D}, M, r)$ then $u \leftarrow \psi(\bar{u})$, and $v \leftarrow \psi(\bar{v})$. Verifier derives R_1,R_2 and R_3 by calculating
   $\bar{R}_1 = u^{s_\xi} / C_1^c$, $\bar{R}_3 = C_1^{s_x} u^{-s_\delta}$
   $\bar{R}_2 = e(C_2, g_2)^{s_x} e(v, w)^{-s_\xi} e(v, g_2)^{-s_\delta} . (\frac{e(C_2, w)}{e(g_1, g_2)})^c$.
   If $c \neq H(M, r, C_1, C_2, C_3, C_4, \bar{R}_1, \bar{R}_2, \bar{R}_3)$ then reject signature.
   $\mathcal{V}$ verifies the attributes by checking $F_{root}^{1/\alpha} . C_3 = e(C_2, C_4)$.
   $\mathcal{V}$ ensures $ChkRvk(\sigma, RL)$ returns -1.

The protocol ends with $\mathcal{V}$ verifying the signer is a member of the group that satisfies $\mathcal{V}$'s requirements and is not revoked.

– **ChkRvk**($\sigma, \mathbf{RL}$) : This algorithm takes a signature σ and a revocation list RL. For each registration key on the list A^* check that $e(C_2/A^*, \bar{u}) = e(C_1, \bar{v})$. If it is equal then return i indicating user is revoked. Otherwise if the equality does not hold for any of the elements in the list return -1 indicating member is not revoked.

Algorithms $\mathbf{Revoke}(\mathbf{A_i}, \mathbf{RL})$ and $\mathbf{Open}(\sigma)$ are exactly as described in section 2 and do not need to be explained further. The scheme is proven correct (See Appendix B), fully anonymous (See Appendix E) and fully traceable (See Appendix D) according to the definitions in section 2. A comprehensive proof is in the appendix. In this section we state the resulting theorems.

Theorem 1. (Full Anonymity): *If the decision linear assumption holds in group G_2 then the Attribute Authentication Scheme is said to be anonymous under the random oracle.*

Theorem 2. (Full Traceability): *If SDH is hard on groups (G_1, G_2) then the Attribute Based Authentication Scheme is said to be fully traceable under the random oracle.*

In the next section we reinforce two more protocols to our scheme. Previously, we assumed the attribute authority is dishonest and that is by giving *Adam* the privilege of creating the master keys instead of the attribute authorities. However, in real life, we need the signer to be able to verify the honesty of the attribute authority. In the next section we enable verification of an attribute authority and the keys it generates.

6 Attribute Key Generation

In this section we add two further protocols which will help a group member trust the attribute keys they are using were created by an honest attribute authority.

Definition 6. (Honestly Generated Attribute Keys): *We say the public attribute key is generated honestly if the attribute authority can not produce it or change it without the knowledge of the central authority. We say that the private attribute key is generated honestly if the member can verify its correctness with the public attribute key and the members registration key.*

To improve our scheme we add the following protocols:

Attribute Public Key Exchange Protocol (APK): This protocol is between the central authority $\mathcal{CA}$ and attribute authority $\mathcal{AA}$. It authenticates the attribute authority to the central authority. It guarantees that the attribute authorities generate the master key honestly. Therefore we replace the $A.KeyGen_{pub}$ algorithm with a interactive protocol. This protocol is a 5-move key generation protocol(suggested in [9]). The procedure is as follows:

1. $\mathcal{AA}$ picks a random $a, b \in \mathbb{Z}_p$ and $c \in \mathbb{Z}_p^*$. $\mathcal{AA}$ then sends $A = w^a$, $B = w^b$, and $C = w^c$ to $\mathcal{CA}$.
2. $\mathcal{CA}$ picks $d, e \in \mathbb{Z}_p$ and sends $DE = w^d.C^e$ to $\mathcal{AA}$
3. $\mathcal{AA}$ picks $f \in \mathbb{Z}_p$ and sends it to $\mathcal{CA}$.
4. $\mathcal{CA}$ sends e, d to user.
5. $\mathcal{AA}$ checks $DE = w^d C^e$. If the check passes calculate $t_j = a + d + f$. Then send $z = (d + f)a + b \mod p$ and c to $\mathcal{CA}$.

6. $\mathcal{CA}$ checks $C = w^c$ and $A^{d+f}B = w^z$ and output $bpk_j = Aw^{d+f}$.

Note that our scheme does not deal with the honesty of the $\mathcal{AA}$ but it guarantees honesty when generating the master key. We would assume some form of standard authentication occurred before the protocol started. Therefore throughout this protocol our $\mathcal{CA}$ is agreeing on the master key used by the $\mathcal{AA}$ but without revealing the key.

Attribute Private Key Exchange Protocol (ASK): This protocol is between the attribute authority $\mathcal{AA}$ and a member of the group $\mathcal{U}_i$. It uses it to authenticate the member before giving him a private attribute key. Earlier we described the procedure for obtaining attribute private keys by explaining the algorithm $A.KeyGen_{pri}$. This can be replaced with the ASK protocol that runs as follows:

1. Attribute authority may want to verify some information about the member before giving him an attribute.
2. The member $\mathcal{U}_i$ and the attribute authority $\mathcal{AA}$ run the protocol $Verifign(\mathcal{U}_i(usk[i], M), \mathcal{AA}(M, \bar{B}))$.
3. From the $Verifign$ protocol $\mathcal{AA}$ obtains the signature, $\sigma = (r,C_1,C_2,C_3,C_4,c,s_\xi,s_x,s_\delta)$.
4. $\mathcal{AA}$ attempts to verify the signature. If it is valid $\mathcal{AA}$ sends $E = C_2^{1/t_j}$ and $F = v^{1/t_j}$ back. Note that v is known to $\mathcal{AA}$ because r is known (See section 5).
5. $\mathcal{U}_i$ calculates his attribute private key $T_{i,j} = E/F^\xi = A_i^{1/t_j}$.
6. $\mathcal{U}_i$ verifies $T_{i,j}$ is correct by checking $e(T_{i,j}, bpk_j) = e(A_i, w)$.

Note that the attribute authority does not know the attribute private key of the user since it can not calculate it from C_2. It also does not know the registration key either because it is coded in C_2.
Protocols ASK and APK ensure that definition 6 holds. Further discussion is contained in the appendix C.

7 Conclusion

In this paper we have proposed an Attribute Based Authentication Scheme and have defined the algorithms and protocols underlying such a scheme. We explained the main two security notions of full traceability and full anonymity. We demonstrated how members of the group can protect themselves from malicious attribute authorities by verifying their honesty in generating keys. We have also provided a construction of an AAS scheme that is proven secure.
Our scheme is efficient since its key and signature are of a constant size. We can use our scheme to construct anonymous credential schemes with multiple attributes. We verify a signature once rather then having multiple certificates each time. Future improvements on the scheme lies in the encryption of the verification key so that the requirements are kept secret from those who do not satisfy it.

References

1. 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.
2. 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.
3. D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of Crypto 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer-Verlag, 2001.
4. 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.
5. J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Eurocrypt 01*, volume 2045 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.
6. D. Chaum. Security without identifcation: Transaction systems to make big brother obsolete. In *Communications of the ACM*, volume 28 of *ACM*, pages 1030–1044, 1985.
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. 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.
9. J. Groth. Fully anonymous group signatures without random oracles. In *Proceedings of Asiacrypt'07*, volume 4833 of *Lecture Notes in Computer Science*, pages 164–180. Springer-Verlag, 2007.
10. 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.
11. 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.
12. A. Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84*, volume 7 of *Lecture Notes in Computer Science*, pages 47–53. Springer-Verlag, 1984.
13. E. Suli and D. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, 2003.

A Extra Preliminaries

Some definitions and preliminaries that we use in the construction of our scheme.

A.1 Attribute Tree Construction

This section gives the preliminaries underlying the attribute tree construction in section 3.

Definition 7. (Bilinear Maps [3]): *Let G_1,G_2 and G_3 be three groups of order p for some large prime p. A function $e : G_1 \times G_2 \rightarrow G_3$ is said to be bilinear if $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ for any generator $g_1 \in G_1$, $g_2 \in G_2$ and any $a, b \in \mathbb{Z}_p$.*

A bilinear map is said to be admissible if it satisfies the following:

- Non-degenerate: The map does not send all pairs in $G_1 \times G_2$ to the identity in G_3.
- Computable: There is an efficient algorithm to compute $e(g_1, g_2)$ for any $g_1 \in G_1$ and $g_2 \in G_2$.

Definition 8. (Bilinear Diffie Hellman Problem [3]) *Let $e : G_1 \times G_2 \rightarrow G_3$ be an admissible bilinear map. Given generators g_1^a, g_1^b and g_2^c for generators $g_1 \in G_1$, $g_2 \in G_2$, and $a, b, c \in \mathbb{Z}_p$. An adversary has a negligible advantage in calculating $e(g_1, g_2)^{abc}$.*

Lagrange interpolation was used in verifying the attribute tree in section 3.

Definition 9. (Lagrange Interpolation [13]): *Given $d + 1$ points on a polynomial of degree d we can uniquely identify the polynomial. Let the points be $(x_0, y_0), ..., (x_d, y_d)$. The polynomial will be:*
$q(x) = \sum_{j=0}^{d} y_j l_j(x)$, *where* $l_j(x) = \prod_{i=0, i \neq j}^{i=d} \frac{x - x_i}{x_j - x_i}$.

A.2 Forking Lemma

Pointcheval and Stern [11] developed the Forking Lemma as a method to prove certain security notions of a digital signature scheme. We use it to prove our scheme is traceable (See Appendix D). Assume a 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, σ_2 depends only on (σ_1, h, M). The Forking Lemma is as follows [11]:

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

A.3 Heavy Row Lemma

In this section we define a Boolean Matrix. and Heavy Row in such matrices [10]. Definitions will be used in proving traceability (See Section D).

Definition 10. (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 the adversary fails, or $\top$ if the adversary manages to win the game.*

Definition 11. (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 the 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.*

A.4 The Strong Diffie-Hellman Assumption

In addition to the q-SDH problem mentioned earlier in section 4, we need the Boneh and Boyen SDH Equivalence theorem to prove traceability of the scheme.

Theorem 4. (Boneh-Boyen SDH Equivalence [1])
Given a q-SDH instance $(\tilde{g}_1, \tilde{g}_2, \tilde{g}_2^{\gamma}, \tilde{g}_2^{\gamma^2}, ..., \tilde{g}_2^{\gamma^q})$, by applying the Boneh and Boyen's Lemma found in [1] we 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.

A.5 Linear Encryption

We now define an encryption scheme which depends on the difficulty of the Decision Linear Diffie-Hellman Assumption. The IND-CPA security of the following encryption scheme will be used to prove our scheme is anonymous.

Definition 12. (A Linear Encryption Scheme [2])
In a Linear Encryption scheme a user's public key is $u, v, h \in G_1$. 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 message M choose random elements $\alpha, \beta \in Z_p$ and output the triple $\langle C_1, C_2, C_3 \rangle = \langle u^{\alpha}, v^{\beta}, Mh^{\alpha+\beta} \rangle$. To decrypt compute $C_3/(C_1^{\xi_1} C_2^{\xi_2})$.

LE has been proven to be IND-CPA secure under the Decision Linear Problem.

B Correctness of the Scheme

Theorem 5. *The AAS 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 $R_3 = \bar{R}_3$. That way we prove $H(M, r, C_1, C_2, C_3, C_4, R_1, R_2, R_3) = H(M, r, C_1, C_2, C_3, C_4, \bar{R}_1, \bar{R}_2, \bar{R}_3)$. The signature should be correctly verified unless the user is revoked. We start our proof with showing that the three equations hold:

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

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

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

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

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

In the verifying algorithm we check revoked users before accepting a signature. In the signature we have $C_1 = \psi(\bar{u})^\xi$ and $C_2 = A_i \psi(\bar{v})^\xi$ for some random ξ. We reject a signature when $(\bar{u}, \bar{v}, C_1, C_2/A_i^*)$ is a co-Diffie Hellman tuple. We also check whether signer satisfies the tree Γ by checking the equality of $F_{root}^{1/\alpha} . C_3 = e(C_2, C_4)$. The correctness of that could be proved by starting with the fact that $F_{root} = e(A_i^\beta, w)^\alpha$ when the tree is satisfied. This implies $F_{root}^{1/\alpha} . C_3 = e(A_i^\beta, w) . e(v^\xi, w)^\beta = e(A_i v^\xi, w^\beta) = e(C_2, C_4)$.

C Honesty in Generating Attribute Keys

In section 6 we gave two protocols and claimed the following:

Claim. The ASK and APK protocols ensure honesty in attribute key generation as defined in section 6.

We prove the claim in two steps. Step one is to prove the attribute public key is generated honestly and the second step is proving that attribute private key is generated honestly. To generate the attribute public key we used the APK protocol. This protocol has perfect correctness and assuming the discrete logarithm problem is hard it is possible to black box simulate both the $\mathcal{AA}$ and $\mathcal{CA}$. The reader is referred to [9] for the full proof.
This leaves us with proving the private attribute key is generated honestly. Creating this key was achieved using the ASK protocol between $\mathcal{U}_i$ and $\mathcal{AA}$. Assuming that the AAS scheme is fully traceable, and σ obtained in the third step of the protocol ($Verifign$) verifies, then the pair (C_2, v) must be created honestly. $\mathcal{AA}$ calculates $C_2^{1/t_j}, v^{1/t_j}$ which is impossible to do without the value of t_j. In the sixth step of the protocol $\mathcal{U}_i$ verifies that t_j used in calculating $T_{i,j}$ is the same as t_j used in calculating bpk_j. $\mathcal{U}_i$ can trust that bpk_j was created honestly using the APK protocol. Therefore he can trust that $T_{i,j}$ is honestly created too.

D Traceability

Theorem 6. *If SDH is hard on groups (G_1, G_2) then the Attribute Based Authentication Scheme is said to be fully traceable under the random oracle.*

Proof. We complete this proof in three steps we begin by defining a security model for proving traceability, then introducing two types of signature forger, and finally we show 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 $Charles$ and $Adam$ as follows:

– **Init:** *Adam* decides on the universal set of attributes $U = t_1,...,t_m$ from $\mathbb{Z}_p$, where m is the size of the set U. *Adam* sends this list to *Charles*.

– **Setup:** *Charles* is given a bilinear map $e : G_1 \times G_2 \rightarrow G_3$ with generators $g_1 \in G_1$, and $g_2 \in G_2$. He is also given a value $w = g_2^{\gamma}$ and n private key bases $bsk[i] = \langle A_i, x_i \rangle$ 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 are valid SDH pairs (Definition 4). *Charles* sends w, and $S_{pub} = \langle G_1, G_2, G_3, e, g_1, g_2 \rangle$ but keeps $S_{pri} = \gamma$. Hash functions H_0 and H are represented as random oracles. Both *Charles* and *Adam* can run the $A.KeyGen_{pub}$ to obtain $\langle bpk_1,...,bpk_m \rangle$ where $bpk_j = w^{1/t_j}$. *Adam* is also given all registration keys $A_1, .., A_n$.

– **Queries:** *Adam* queries the oracles USK, Signature and Hash as follows:
USK Oracle: *Adam* asks for a certain private key by sending *Charles* an index i. If *Adam* queries an index where $x_i = \star$ abort the game and declare failure otherwise respond with sending $\langle A_i, x_i \rangle$.
Hash Oracle: When *Adam* asks *Charles* for the hash of $(M, r, C_1, C_2, C_3, C_4, R_1, R_2, R_3)$, *Charles* responds with a random element in G_1 and saves the answer just in case the same query is requested again. This represents the hash function H. When *Adam* asks *Charles* for the hash of $(\bar{D}, M, r)$, *Charles* responds with two random elements in G_2 and saves the answer.
Signature Oracle: *Adam* runs the $\bar{D} = TCreate(\Gamma, \alpha, \bar{B})$ for a random $\alpha \in \mathbb{Z}_p$ and a set of attribute public keys $\bar{B}$. He then sends $\bar{D}$ to *Charles*. *Adam* requests a signature on a message M by the member i. If $x_i \neq \star$ then *Charles* follows the same signing procedure done in section 5.

If $x_i = \star$, *Charles* simulates a signature by selecting a random r from $\mathbb{Z}_p$ to obtain $(\bar{u}, \bar{v}) \leftarrow H_0(\bar{D}, M, r)$ and sets $u \leftarrow \psi(\bar{u})$ and $v \leftarrow \psi(\bar{v})$. He then picks a random ξ and β from $\mathbb{Z}_p$.

Following this *Charles* calculates $C_1 = u^{\xi}$, $C_2 = A_i v^{\xi}$, $C_3 = e(v^{\xi}, w)^{\beta}$ and $C_4 = w^{\beta}$ and randomly picks c, s_{δ}, s_x, and s_{ξ} from $\mathbb{Z}_p$. He calculates $R_1 = u^{s_{\xi}}/C_1^c$, $R_3 = C_1^{s_x} u^{-s_{\delta}}$ and
$R_2 = e(C_2, g_2)^{s_x} e(v, w)^{-s_{\xi}} e(v, g_2)^{-s_{\delta}} . (e(C_2, w)/e(g_1, g_2))^c$.
Charles adds c to the list of the hash oracle H to maintain consistency.

Charles takes every D_j in $\bar{D}$ and calculates $e(A_i^{t_j}, D_j)^{\beta}$, this is in place of the recursive $Sign_{Node}$ function in section 3. *Charles* can now calculate F_{root} using the elements it calculated and Γ.

Charles returns the signature $\sigma = (r, C_1, C_2, C_3, C_4, c, s_{\xi}, s_x, s_{\delta}, F_{root})$ to *Adam*.

– **Output:** *Adam* asks to be challenged and sends *Charles* a message M. *Charles* responds with a $\bar{D}$ for a certain Γ. If *Adam* is successful he will

output a signature $\sigma = (r, C_1, C_2, C_3, C_4, c, s_\xi, s_x, s_\delta, F_{root})$ for a message M where C_1 and C_2 should not contain any of the revocation list elements A^* encoded in them. Let A_i^* be the value used in signing the forged signature. For $i = 1, ..., n$, *Charles* checks whether $e(C_2/A_i, \bar{u}) = e(C_1, \bar{v})$. If the equality holds then this implies that $A_i^* = A_i$. 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 no equality is identified output σ.

From this model of security there are two types of forgery. Type-I outputs a signature that can be traced to some identity which is not part of $\{A_1,...,A_n\}$. Type-II has $A_i^* = A_i$ where $1 \leq i \leq n$ but *Adam* did not submit a query of i to the USK oracle. We prove both forgeries are hard.

Type-I: If we consider Theorem 4 for a $(n + 1)$ SDH, we can obtain g_1,g_2 and w. We can also use the n pairs (A_i, x_i) to calculate the private key bases $\langle A_i, x_i \rangle$. These values are used when interacting with *Adam*. *Adam*'s success leads to forgery of Type-I and the probability is ε.

Type-II: Using Theorem 4 once again but for a (n) SDH, we can obtain g_1, g_2 and w. Then we use the $n - 1$ pairs (A_i, x_i) to calculate the private key bases $\langle A_i, x_i \rangle$. In a random index i^*, we choose the missing pair randomly where $A_{i^*} \in G_1$ and set $x_{i^*} = \star$. *Adam*, in the security model, will fail if he queries the USK oracle with index i^* and all other private key queries will succeed. In the signature oracle (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 phase the probability of tracing a forged signature which leads to index i^* is equal to ε/n.

We now prove that any of the two forgeries contradict the SDH assumption. For this we make use of the Forking Lemma (See Theorem 3). Let *Adam* be a forger of any type in which the security model succeeds with probability $\tilde{\varepsilon}$. A signature will be represented as $\langle M,\sigma_0,c,\sigma_1,\sigma_2 \rangle$, M is the signed message, $\sigma_0 = \langle r,C_1,C_2,C_3,C_4,R_1,R_2,R_3 \rangle$, c is the value derived from hashing σ_0, and $\sigma_1 = \langle s_\xi, s_x, s_\delta \rangle$ which are values used to calculate the missing inputs for the hash function. Finally, $\sigma_2 = F_{root}$ the value that depends 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 can change values of $H(M, r, ..)$, while the 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 *Charles*), 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 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 $\nu = Pr[S] = \tilde{\varepsilon} - 1/p$ where $1/p$ term represents the possibility that *Adam* guessed the hash of (M, σ_0) without querying it. For each χ, $1 \leq \chi \leq q_H$, let

S_χ be the tuple (ω, ℓ_0, ℓ) where $Ind(\omega, \ell_0, \ell) = \chi$. Let Φ be the set of indexes 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 indexes $a, a+1, ..., b$. For every $\chi \in \Phi$ consider the Heavy Row Lemma (See Appendix A.3) with a matrix with rows indexed by $(\omega, \ell_0, \ell|_1^{\chi-1})$ 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 $\forall (x, y) \in [\Omega_\chi : Pr_{\tilde{y}}[(x, \tilde{y}) \in S_\chi]] \geq \nu/(4q_H)$. 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 approximately $\nu/4$. 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 $Pr_{\tilde{\ell}}[(\omega, \ell_0, \tilde{\ell}) \in S_j|\tilde{\ell}|_1^{j-1} = \ell|_1^{j-1}] \geq \nu/(4q_H)$. In other words if we have another simulated run of the adversary with $\tilde{\ell}$ that differs from ℓ starting the jth query *Adam* will forge another signature $\langle M, \sigma_0, \tilde{c}, \tilde{\sigma_1}, \sigma_2 \rangle$ with the probability $\nu/(4q_H)$.
Finally we show how we can extract from $\langle \sigma_0, c, \sigma_1, \sigma_2 \rangle$ and $\langle \sigma_0, \tilde{c}, \tilde{\sigma_1}, \sigma_2 \rangle$ a new SDH tuple. Let $\Delta c = c - \tilde{c}$, and $\Delta s_\xi = s_\xi - \tilde{s}_\xi$, and similarly for Δs_x, and Δs_δ. Divide two instances of the equations used previously in proving correctness of the scheme (See Appendix B). One instance with $\tilde{c}$ and the other with c to obtain the following:

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

Letting $\tilde{x} = \Delta s_x / \Delta c$ we get $e(g_1, g_2)/e(C_2, w) = e(C_2, g_2)^{\tilde{x}} e(v, w)^{-\tilde{\xi}} e(v, g_2)^{-\tilde{x}\tilde{\xi}}$ which can be rearranged as $e(g_1, g_2) = e(C_2 v^{-\tilde{\xi}}, w g_2^{\tilde{x}})$. Let $\tilde{A} = C_2 v^{-\tilde{\xi}}$ and we get $e(\tilde{A}, w g_2^{\tilde{x}}) = e(g_1, g_2)$. Hence we obtain a new SDH pair $(\tilde{A}, \tilde{x})$ breaking Boneh and Boyens Lemma(See Theorem 4). Now putting things together we get the following theorems:

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

E AAS Scheme Anonymity

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

Assuming *Adam* is an adversary that breaks the anonymity of the AAS scheme, we will prove that there exist an adversary *Eve* that solves the decisional linear assumption using *Adam*'s capabilities. Note that *Eve* in this game plays a

challengers role when it comes to interacting with *Adam* and an adversary's role when she interacts with *Charles*. So the resulting game is described below:

- **Init:** *Adam* decides the universal set of attributes $U = \{t_1, ..., t_m\}$, in which he would like to be challenged upon and gives it to *Eve*.

- **Setup:** *Charles* 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 $a, b \in \mathbb{Z}_p$. Z is either random or $Z = u_2^{a+b}$. *Eve* should decide which Z she was given. Recall that g_1, g_2 are in G_1 and G_2 respectively. *Eve* chooses a random γ from $\mathbb{Z}_p$ and assigns $w = g_2^{\gamma}$. She creates the $n-2$ private key bases $bsk[i] = \langle A_i, x_i\rangle$ as in section 5. She will then choose a random $W \in G_2$. The missing private key bases of user i_0 and i_1 are 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 if $Z = u_2^{a+b}$ then $A_{i_0} = A_{i_1}$. *Eve* does not know the values of either $bsk[i_0]$ or $bsk[i_1]$. We will show later in our security model how she can still interact with *Adam* while pretending she does know them. *Eve* gives *Adam* w and $S_{pub} = \langle G_1,G_2, G_3, e, g_1, g_2\rangle$ but retains $S_{pri} = \gamma$. Both *Charles* and *Eve* can run the $A.KeyGen_{pub}$ algorithm to obtain $\langle bpk_1,...,bpk_m\rangle$.

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

  If *Adam* queries the signature oracle he should send an index i, a verifying key $\bar{D}$ for a certain attribute tree Γ, and a message M. If $(i \neq i_0, i_1)$, *Eve* will reply with a signature $\sigma = \langle r,C_1,C_2,C_3,C_4,c,s_\xi,s_x,s_\delta, F_{root}\rangle$ as done in section 5. If $(i = i_0)$, *Eve* picks a random $s, t, l, \beta \in \mathbb{Z}_p$ and makes the following assignments:
  $C_1 = h_0u_0^s$; $C_2 = ZWu_2^sh_0^tu_0^{st}$; $\bar{u} = u_0^l$; $\bar{v} = (u_2u_0^t)^l$.
  Let $\xi = (a+s)/l \in \mathbb{Z}_p$, then $C_1 = \bar{u}^{\xi}$ and $C_2 = A_{i_0}\bar{v}^{\xi}$.
  Eve assigns $C_3 = e(ZW, w)^{\beta}$ and $C_4 = w^{\beta}$. She calculates F_{root} by replacing the recursive algorithm $Sign_{Node}$ with $Fake - Sign_{Node}$, which is described below:

$$Fake - Sign_{Node}(leaf) = \begin{cases} \text{If } (j \in \Gamma)\text{; return } e((u_2^sh_0^tu_0^{st})^{t_j}, D_j)^{\beta} \\ \qquad\qquad = e(u_2^sh_0^tu_0^{st}, w)^{\beta q_j(0)} \\ \text{Otherwise return } \perp \end{cases}$$

  F_{root} in this case will equal $e(u_2^sh_0^tu_0^{st}, w)^{\beta\alpha}$. Notice that $F_{root}^{1/\alpha}.C_3 = e(C_2, w)^{\beta}$. If β_1, β_2 are random elements in $\mathbb{Z}_p$, then it is hard to distinguish between the following two triples:
  $\langle e(v^{\xi}, w)^{\beta_1}, w^{\beta_1}, e(A_i, w)^{\beta_1\alpha}\rangle$ and $\langle e(ZW, w)^{\beta_2}, w^{\beta_2}, e(u_2^sh_0^tu_0^{st}, w)^{\beta_2\alpha}\rangle$.

If $(i = i_1)$, *Eve* picks a random $s, t, l, \beta \in \mathbb{Z}_p$ and makes the following assignments:
$C_1 = h_1 u_1^s$; $C_2 = W h_1^t u_1^{st} / u_2^s$; $\bar{u} = u_1^l$; $\bar{v} = (u_1^t / u_2)^l$
Let $\xi = (b + s)/l \in \mathbb{Z}_p$. Then $C_1 = \bar{u}^{\xi}$ and $C_2 = A_{i_1} \bar{v}^{\xi}$.
Eve assigns $C_3 = e(W, w)^{\beta}$ and $C_4 = w^{\beta}$. She calculates F_{root} by replacing the recursive algorithm $Sign_{Node}$ with $Fake - Sign_{Node}$ which is described below:

$$Fake - Sign_{Node}(leaf) = \begin{cases} \text{If } (j \in \Gamma); \text{ return } e((h_1^t u_1^{st} / u_2^s)^{t_j}, D_j)^{\beta} \\ \qquad = e(h_1^t u_1^{st} / u_2^s, w)^{\beta q_j(0)} \\ \text{Otherwise return } \perp \end{cases}$$

F_{root} in this case will equal $e(h_1^t u_1^{st} / u_2^s, w)^{\beta\alpha}$. Notice that $F_{root}^{1/\alpha}.C_3 = e(C_2, w)^{\beta}$.
If β_1, β_2 are random elements in $\mathbb{Z}_p$, it is hard to distinguish between the triple :
$\langle e(v^{\xi}, w)^{\beta_1}, w^{\beta_1}, e(A_i, w)^{\beta_1\alpha} \rangle$ and $\langle e(W, w)^{\beta_2}, w^{\beta_2}, e(h_1^t u_1^{st} / u_2^s, w)^{\beta_2\alpha} \rangle$

Eve chooses random values $r, c, s_{\xi}, s_x, s_{\delta}$ from $\mathbb{Z}_p$. *Eve* sets the values
$R_1 = u^{s_{\xi}} / \psi(C_1)^c$, $R_3 = \psi(C_1)^{s_x} \psi(u)^{-s_{\delta}}$, and
$R_2 = e(\psi(C_2), g_2)^{s_x} e(\psi(\bar{v}), w)^{-s_{\xi}} e(\psi(\bar{v}), g_2)^{-s_{\delta}} (e(\psi(C_2), w) / e(g_1, g_2))^c$.
The probability that $H(M, \psi(C_1), \psi(C_2), \psi(C_3), \psi(C_4), R_1, R_2, R_3)$ or $H_0(\bar{D}, M, r)$ has 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 *Eve* adds
$H(M, \psi(C_1), \psi(C_2), \psi(C_3), \psi(C_4), R_1, R_2, R_3) = c$ and $H_0(\bar{D}, M, r) = (\bar{u}, \bar{v})$
to the hash oracles list.
Eve sends to *Adam* the signature $\sigma = \langle r, \psi(C_1), \psi(C_2), \psi(C_3), \psi(C_4), c, s_{\xi}, s_x, s_{\delta}, F_{root} \rangle$

Adam queries the revoke oracle, by sending a users index i to be revoked. *Eve* replies with adding A_i to RL. If *Adam* queries i_0, i_1, *Eve* reports failure. RL is accessible to both *Adam* and *Charles*.

- **Challenge:** *Adam* asks to be challenged on message M, verification key $\bar{D}$ for a certain Γ and indexes i_0^* plus 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 oracle. So *Eve* responds with signature σ_b.

- **Phase 2:** Is exactly like phase 1 as long as i_0^* and i_1^* are not queried in neither the revoke nor the USK oracles.
- **Output :** *Adam* outputs a guess $\bar{b} \in \{0, 1\}$. If $b = \bar{b}$ then Z is random, otherwise $Z = u_2^{a+b}$.

There are two ways this game can end. Case one is when *Eve* does not abort. If Z is random then $Pr[b = \bar{b}] > 1/2 + \varepsilon$ otherwise if $Z = u_2^{a+b}$ then both signatures should be identical and therefore challenge is independent of b. Hence

$Pr[b = \bar{b}] = 1/2$. So the advantage of Eve solving the linear challenge is at least $\varepsilon/2$.
The second case is Eve aborts and so fails. Eve can 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.