**University of Bath**

**PHD**

# Attribute based authentication schemes

Khader, Dalia

*Award date:*
2009

*Awarding institution:*
University of Bath

Link to publication

# Attribute Based Authentication Schemes

submitted by

Dalia Daoud Khader

for the degree of Doctor of Philosophy

of the

University of Bath

2009



Signature of Author .....

Dalia Daoud Khader

# Attribute Based Authentication Schemes

Dalia Daoud Khader

# SUMMARY

This thesis' major contribution is to propose an attribute based authentication scheme (AAS). An AAS scheme is a new cryptosystem that extends the field of public key cryptography and more precisely digital signatures. An AAS scheme allows a verifier to decide on the set of attributes (s)he would like the signer to possess. The verifier sends the request to a group of possible signers as a monotone boolean expression. Any member with sufficient attributes can sign. The scheme maintains certain properties as follows (see scenario in Chapter 1):

- No previous knowledge assumption: The signer and verifier may or may not have met before; therefore we can not rely on any kind of previous knowledge.
- Unforgeable : It is hard to forge signatures and/or the proof of possession of attributes.
- Anonymous Identities: Given the signature it is hard to identify the signer.
- Unlinkable: Given two signatures it is hard to know whether the signer is the same or not.
- Traceable: Each group of potential signers have a group manager and he is the only one capable of revoking anonymity and discovering the signer's identity. This property is meant to ensure signers do not misuse anonymity.
- Anonymous Attributes: The attribute disclosure should be to the minimum.
- Coalition Resistant: If a verifier requires more than one attribute from the signers, the signers should not be able to get together their individual attributes and sign as one entity.
- Separability: The tasks of different authorities should be separable and each entity should be capable of performing its task independently from others.

Different attribute oriented authentication schemes exist in literature, however each of them is designed to serve a certain application. However the properties we listed above never co-existed in one scheme. The proposed scheme was constructed in three phases each covers more properties than the one before as described in details in Chapter 5. We then propose a general construction that helps creating an AAS scheme using a group signature (Chapter 4) and attribute tree (Section 5.2) as building blocks. We prove that the security of the new AAS scheme created by such construction is based on the security of the group signature scheme.

# ACKNOWLEDGMENTS

Thank you God for giving me the opportunity to start my PhD and the capability to finish it. I shall cherish this gift forever.

Mama and Baba, to whom I dedicate this dissertation, you have been my source of strength through out the entire stages of my studies. The words "Thank you" are not enough to express my heart-felt gratitude and appreciation. Love you.

None of this would have been possible without the love and support of all members of my family in particular my parents, my grandparents, my sister Samah, and my brother Suleiman. I will always be grateful to you all.

I would like to express my utmost gratitude to all my friends who have helped me immensely not only to complete this thesis but also amass many unforgettable memories; the endless coffee breaks, the amazing lunch breaks, the boardgame nights, the picnics, the long discussions we have enjoyed together, etc.

Thanks for Dr. Russell Bradford for accepting me as one of his students.

Thanks to my examiners Professor James Davenport and Dr. Liqun Chen for their valuable feedback.

# Contents

# List of Figures

# Chapter 1

# Introduction

Everyone has a secret they would like to hide from the rest of the world. The best way to hide it is to keep it to oneself. However, sometimes it is required to share the secret with others. In order to transfer confidential information, some form of protection needs to take place. A protection can take the form of a locked box, where two communicators Alice and Bob have a copy of the key and no one else does. Alice writes a secret message, then puts it in the box knowing that no one can open it unless they own the key. If Alice writes a letter, locks it in the box, hides it somewhere around London and then challenges others to reveal her secret, she is not proving the box to be secure. To prove that the box is secure she will have to give the box together with the design specification and with hundreds of similar boxes plus their keys and then ask anyone to break into it[1].

Confidentiality is the major purpose of the science of cryptography. The science itself goes way back to the 4000 B.C. to the hieroglyphs carved into monuments from Egypt's Old Kingdom [78]. Encryption schemes are a set of algorithms that help in coding a message to some ciphertext that is unreadable by people who do not own a specific key. The key is certain information that helps changing plaintext to ciphertext and vice versa. The process that changes ciphers to understandable text using the key is called decryption.

In 1976 Diffie and Hellman proposed an idea that changed cryptography drastically [56]. The idea was having cryptosystems that rely on a pair of keys. If you have a locked box, how would you distribute copies of the key such that it does not get exposed? Diffie and Hellman suggested having one key to encrypt and a different one to decrypt data. The decryption key is private whereas the encryption key is public to all. There is no need to distribute keys in secret in this case since knowing the public key does not reveal anything in the message. A cryptosystem that uses one key is called symmetric while the cryptosystem that uses two different keys is called asymmetric.

[1] Example inspired from Bruce Schneier's book [121]

The invention of asymmetric cryptosystems was the start of a new field in cryptography, called authentication. If people can tell who they are dealing with they may not need to maintain confidentiality. Digital signature schemes are a set of algorithms that simulate handwritten signatures in an electronic form [56]. They are used for authentication purposes. They enable signing digital documents, such as emails. A private key owned by the signer is used in creating signatures and a public key used by any verifier is used in checking validity of the signature.

In real life we tend to need documents that are signed by someone who has a specific role. For example, a pharmacy needs a prescription signed by a doctor. In other words, we need a document to be certified. One way of doing this is to have a document signed by Dr Smith as an example. That means the pharmacy has to have a list of certified doctors and every time it wants to verify a prescription it goes through the list one by one, searching for Dr. Smith.

Another way of doing it is using group signatures [41]. Group signatures are digital signatures that allow any member of a group to sign anonymously on behalf of the group and in case of a dispute, a trusted group manager can revoke that anonymity. Suppose the Ministry of Health is the group manager. Any doctor that is registered in the group can sign a prescription. The pharmacy does not need to have a list of doctors anymore because it verifies the signature on a higher level. In other words, verification is done with the question, "does whoever signed belong to the group of certified doctors?"

The motivation of this thesis is to have attribute verification within a group. Assume Bob has a company where he is the most senior employee and is referred to as the group manager. 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 in advance.
- Unforgeable : An employee cannot forge a proof of possession of an attribute. A non-employee can not pretend to be an employee.
- Anonymity of Identity: Alice cannot tell from the signature which employee signed.In other words, neither can a verifier nor an eavesdropper derive the identity of the signer from the signature.
- Unlinkable: Alice cannot figure out whether or not two signatures are created by the same signer. If Alice can link signatures, she would be able to analyze the attributes and possibly break the anonymity.

- 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 this is a valid signature
- Anonymity of Attributes: If the verifier gives two or more alternatives of sets of attributes he requires, the signer should be capable of hiding which set he has chosen to sign with. For instance, if Alice needs the signer to be an employee from either department $A$ or $B$. As long as the rest of the policy is met, she does not need to know how the policy has been satisfied. By policy we mean the set of attributes the verifier requests and it is represented as monotone boolean expression.
- Coalition Resistant: Assuming an employee is in department $A$ but he is not a senior manager and another employee is in department $C$ and is a senior manager. The two employees cannot sign the message jointly.
- Separability: Each department in the company gives out attributes (such as manager, senior, junior) under its control independent of other departments.

Many authentication schemes include one or two of the above properties. Our proposed scheme in this thesis captures all of the properties mentioned.
The outline of the thesis will be as follows:

- **Chapter 2 Cryptographic Preliminaries:** In this chapter we provide a set of preliminaries that will make the thesis self contained.
- **Chapter 3 Provable Security:** Methods and tools used in proving cryptographic schemes secure are given in this chapter.
- **Chapter 4 Group Signature Scheme:** Group Signature is the main building block of our construction since it has desirable properties. Therefore we dedicate a whole chapter to describe two types of group signature schemes (i.e. Static and Dynamic) in the literature and we support the chapter by giving examples of such schemes.
- **Chapter 5 Attribute Authentication Scheme:** This chapter explains our main contribution to the world of cryptography. We have built an attribute authentication scheme in three stages. We explain these phases and support them with an example construction of each.
- **Chapter 6 General Construction:** In this chapter we propose a general construction that converts group signatures into attribute authentication schemes. Such a conversion algorithm is needed since group signatures have been studied extensively. It saves the effort in doing the same research for the attribute authentication scheme.

- **Chapter 7 Concluding Remarks and Future work:** We end the thesis with a summary of the thesis, some remarks and possible future plans.
- **Appendix A** In the appendix we include detailed proofs for the security of the constructions in Chapter 5.

Generally, the main contributions of this thesis are creating an attribute based authentication scheme in three phases as explained in Chapter 5 and the general construction of the attribute authentication schemes of Chapter 6.

# Chapter 2

# Cryptographic Preliminaries

Every cryptosystem needs to be proven secure. The common practice of proving schemes secure is to show that breaking it implies solving a mathematical problem that is known or believed to be hard. The mathematical problems are referred to as complexity assumptions. In Section 2.2 we list down some of the assumptions required for this thesis. We then explain some cryptosystems that exist in literature such as encryption schemes (Section 2.3), digital signatures (Section 2.4) and different cryptographic protocols (Section 2.5). Each of these sections includes a general definition of the scheme and is supported with examples relevant to the thesis. We explain the procedure of proving a scheme secure in the following chapter.

## 2.1 Introduction

This chapter makes the thesis self contained by providing the definitions of required cryptographic preliminaries. It is sensible to start this chapter with some notations and conventions used throughout the thesis. The definitions in the following section are taken from [97, 12, 105, 119]

### 2.1.1 Notations and Conventions

A finite string of bits is notated as $\{0,1\}^*$. If $a$ and $b$ are finite strings of bits then $|a|$ is the length of $a$ and $a||b$ is their concatenation. Let $\mathbb{Z}$ denote integers and gcd represent the greatest common divisor then:

**Definition 2.1.1.** *The integers modulo $n$, denoted $\mathbb{Z}_n$, is the set of (equivalence classes of) integers $\{0, 1, 2, ..., n-1\}$. Addition, subtraction, and multiplication in $\mathbb{Z}_n$ are performed modulo $n$.*

**Definition 2.1.2.** (Groups) *A group $G$ is a set with an operation (for instance in multiplicative groups the operation is often written ".") which [128]*

- *Is closed.* $\forall a, b \in G$ *then* $a.b \in G$
- *Has an identity* $I$. $\forall a \in G$, $a.I = a$
- *Is associative.* $\forall a, b, c \in G$, $(a.b).c = a.(b.c)$
- *Every element has an inverse.* $\forall a \in G$, *there exist an* $a^{-1}$ *where* $a.a^{-1} = I$.

A group which is commutative is often called abelian. Commutative implies that $\forall a, b \in G$, $a.b = b.a$. The majority of groups that are cryptographically interesting are abelian. A finite group is a group which has finite number of elements. Later chapters, 4, 5 and 6, will use the term group to represent a group of people.

**Definition 2.1.3.** (Fields) *A field is an additive abelian group* $\mathbb{F}$ *with identity* $0$, *such that* $F \setminus \{0\}$ *also forms an abelian group with respect to another operation (which is usually written multiplicatively). The two operations, addition and multiplication, are linked via the distributive law [128]:* $a.(b + c) = a.b + a.c = (b + c).a$

**Definition 2.1.4.** *The multiplicative group of* $\mathbb{Z}_n$ *is* $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n | \gcd(a, n) = 1\}$. *In particular if* $n$ *is a prime, then* $\mathbb{Z}_n^* = \{a | 1 \leq a \leq n - 1\}$

**Definition 2.1.5.** *The order of* $\mathbb{Z}_n^*$ *is the number of elements in the group, namely* $|\mathbb{Z}_n^*|$

Let $G$ be an arbitrary group. Throughout the thesis we will be making use of multiplicative groups (i.e. the binary operation of a group is ".").

**Definition 2.1.6.** *The order of a finite group* $G$ *is the number of elements in the group, namely* $|G|$.

**Definition 2.1.7.** *A group* $G$ *is cyclic if there is an element* $g \in G$ *such that for each* $\bar{g} \in G$ *there is an integer* $a$ *with* $\bar{g} = g^a$. *Such an element* $g$ *is called a generator of* $G$.

Groups of prime order have useful properties and are widely used in cryptography. All groups of prime order are cyclic. An element $h$ of a group $G$ is called non-trivial if it is not equal to the identity element of the group. Suppose $G$ is a group of order $p$ where $p$ is a prime, and let $h$ be a non-trivial member of $G$, then $h$ is a generator of $G$. There are more properties for such groups; however it is beyond our thesis scope to go through all of them.
A group isomorphism is a structure preserving map between two groups that sets up a one-to-one correspondence between the elements of the groups in a way that respects the given group operations.

**Definition 2.1.8.** *A computable isomorphism* $\psi$ *from* $G_2$ *to* $G_1$ *exists, if given an element in* $G_2$ *it is possible to map it to an element in* $G_1$. *In other words* $g_1 = \psi(g_2)$ *where* $g_1 \in G_1$ *and* $g_2 \in G_2$.

We should point out that having a computable isomorphism from $G_1$ to $G_2$ does not imply that the inverse isomorphism (i.e. from $G_2$ to $G_1$) is computable. A pair $(a, b) \in G^2$ implies that elements $a$ and $b$ in the pair belong to group $G$. The set $\{a_1, ..., a_u\} \in \mathbb{Z}^u$ implies all $u$ elements in the set belong to $\mathbb{Z}$. The notation $a \in_R X$ means choosing the element $a$ randomly from $X$ where $X$ can be a group $G$, a set $S$, a set of integers $\mathbb{Z}_p$ etc.

After covering some group theory terminologies we should define different types of algorithms that are relevant to our thesis and widely used in cryptography and complexity theory. We shall explain the difference between probabilistic and deterministic algorithms and define the terms "Polynomial Time Algorithms" and "Probabilistic Polynomial Time".

Probabilistic algorithms are important in cryptography since it is often that the algorithms of encryption and signature schemes are randomized and furthermore in studying the security of schemes the adversaries are also modeled as probabilistic (see Section 3). Therefore we clarify what is meant by probabilistic and deterministic algorithms [53].

The output of a deterministic algorithm, say $y$, is completely determined by its input, for example $x$. In other words, a sequence of predefined steps are used in order to calculate $y$ from $x$. A probabilistic algorithm, on the other hand is partly effected by a random event. The following is a definition of probabilistic algorithm[1].

**Definition 2.1.9.** (Probabilistic Algorithms [53]): *Given an input $x$, a probabilistic algorithm $A$ may toss a coin a finite number of times during its computation of the output $y$. The outcome of tossing the coin affects the next step in the calculation and affects the number of times the coin is tossed where the maximum bound is determined from the input $x$. The coin tosses are independent and fair (i.e. each side appears with probability of $1/2$ ).*

Following the definition of probabilistic algorithms we define a polynomial time algorithm and a probabilistic polynomial time algorithm as follows:

**Definition 2.1.10.** (Polynomial Time Algorithm) *An algorithm is called polynomial time if its worst-case running time function is polynomial in the input size. Any algorithm whose running time cannot be bounded by a polynomial is called super polynomial time.*

An algorithm is probabilistic polynomial time (PPT) if it uses randomness (e.g. flipping coins) and its worst case running time is polynomial in input size. The following is a formal definition[2]:

[1] Definition 5.1 page 112 in [53]
[2] Definition 5.2 page 115 in [53]

**Definition 2.1.11.** (Probabilistic Polynomial Time Algorithm) *A probabilistic algorithm $A$ is a probabilistic polynomial time algorithm if the running time of $A(x)$ is bounded by $P(|x|)$ where $P$ is a polynomial. The running time is measured by the number of steps in the model algorithm (i.e. The number of steps in a probabilistic Turing machine). Tossing a coin is one step in this model.*

In any cryptographic scheme we have some parameters used in the setup of the system that determines the length of keys, messages and running times of honest parties and attackers; everything is typically polynomially bounded by such a parameter. Such a parameter is referred to as the security parameter and can take arbitrary large values.
The notion of negligible functions, as Bellare defines it in his work [8], is used in theoretical cryptography to formalize the notion of a function asymptotically "Too Small to Matter". A formal definition of a negligible function helps in saying that a cryptographic primitive or scheme have a certain level of security and that is by providing a robust notion of rareness. A rare event should occur rarely even when repeating an experiment for feasible number of times. In this case the experiment involves an adversary trying to break a scheme. A function is called negligible if it vanishes faster than the reciprocal of any polynomial. A more formal definition is given below:

**Definition 2.1.12.** (Negligible Functions) *A function $f(a)$ is said to be negligible if $\forall c$, $\exists a_c$, where $f(a) \leq a^{-c}$, for every $a \geq a_c$.*

The advantage of an adversary is the measure of how successful it is in attacking the scheme. To assume a complexity assumption is hard or a cryptosystem is secure, the advantage of an attacker succeeding should be $Adv(k) \leq \varepsilon$ where $\varepsilon$ is negligible and $k$ is a security parameter.
In building our scheme we have also used Lagrange Interpolation. Loosely speaking, the interpolation of Lagrange comes from the fact that given $d+1$ points on a polynomial of degree $d$ we could uniquely identify that polynomial. The following is the formal definition:

**Definition 2.1.13.** (Lagrange Interpolation [129]):
*Let the points be $(x_0, y_0), ..., (x_d, y_d)$. We define the Lagrange polynomial to be:*
$q(x) = \sum_{j=0}^{d} y_j l_j(x)$, *where* $l_j(x) = \prod_{i=0, i\neq j}^{i=d} \frac{x - x_i}{x_j - x_i}$.

The following section explains different complexity assumptions used in this thesis.

## 2.2 Complexity Assumptions

Loosely speaking, to prove that a scheme is secure one shows that breaking it can be no easier than solving some mathematical problem that is assumed to be intractable.

These kind of assumptions are referred to as "Complexity Assumptions". In Chapter 3 such security proofs are explained in detail. In this section some common "Complexity Assumptions" are given.
An important class of problems, used massively in cryptography and in this thesis, are based on the Discrete Logarithm Problem (DLP). Let $G$ be a finite multiplicative group, we shall define the DLP first and then give some complexity assumptions that are strongly related to it.

**Definition 2.2.1.** (Discrete Logarithm Problem (DLP) [128])
*Given $h, g \in G$, find an $x \in \mathbb{Z}_p^*$ if it exists where $h = g^x$.*

We would like to mention that one of the most widely used groups in cryptography is the elliptic curve group. The DLP is believed to be hard in such suitably chosen elliptic curve groups. The best algorithm known to solve the discrete logarithm problem in literature is Pollard's rho method [112] which is of order $O(\sqrt{|E(\mathbb{F}_q)|})$ where $E(\mathbb{F}_q)$ refers to an elliptic curve over the field of integers modulo $q$. Smart has showed in his work in [127] how solving the DLP in elliptic curve groups using Pollard Rho's method can take up to 71 years on a single sparc-10 network when $q = 2^{19}$. For more details about elliptic curves the reader is referred to [15, 16].
Throughout the thesis we refer to complexity assumptions that are hard in a group of prime order $p$. When implementing we can use elliptic curve groups for such groups.
Diffie–Hellman problem is one example and it has been shown that it is computationally equivalent to the DLP in [18, 95]. The following is its definition.

**Definition 2.2.2.** (Diffie–Hellman Problem (DH) [128])
*Given $g, A, B \in G$ where $A = g^a$ and $B = g^b$ for $a, b \in \mathbb{Z}_p^*$. Find a $C$ where $C = g^{ab}$.*

The DH is a hard problem but no stronger than the DLP. The values $a$ and $b$ are not given and if the DLP can be solved $a$ and $b$ can be computed, therefore $C$ can be calculated too.
Another example of a DLP related complexity assumption is the Decisional Diffie Hellman (DDH). It was first mentioned in Brands' work in [29]. Even though at first glance it appears that DDH assumption is computationally equivalent to DH, it is not the case as proven in [96]. It has been shown that such an assumption is weaker. Nevertheless, it is still a valid assumption underlying the security of many cryptosystems in the literature.

**Definition 2.2.3.** (Decisional Diffie –Hellman Problem (DDH) [128])
*Given $g, A, B$ and $C \in G$ where $A = g^a$, $B = g^b$ and $C = g^c$ for $a, b, c \in \mathbb{Z}_p^*$ and $A, B, C \in G$. Determine whether or not $c = ab$*

Similar to the DH problem the DDH is no stronger than the DLP or the DH. The values of $a, b$ and $c$ are not given but can be computed if DLP is easy. If the DH is

easy the value $\bar{C} = g^{ab}$ is computed and compared with $C$.
In the same work of Brands' in [29] another complexity assumption was proposed, based on the Decisional Diffie Hellman, the Representation Problem. In order to state such complexity assumption we opt to discuss the definition of a representation and then state the problem itself.
Let $k \geq 2$ be a constant, $q$ be a prime number and $G_q$ a group of order $q$. A generator-tuple of length $k$ is a $k$-tuple $(g_1, ..., g_k)$ with, for all $i, j \in \{1, ..., k\}$, $g_i \in G_q \backslash \{1\}$ and $g_i \neq g_j$, if $i \neq j$. An index tuple of length $k$ is a $k-$tuple $(a_1, ..., a_k)$ with $a_i \in \mathbb{Z}_q^*$ for all $i \in \{1, ..., k\}$.

**Definition 2.2.4.** (Representation [29])*:*
*For any $h \in G_q$, a Representation of $h$ with respect to a generator-tuple $(g_1, ..., g_k)$ is an index-tuple $(a_1, ..., a_k)$ such that $h = \prod_{i=1}^{k} g_i^{a_i}$*

**Definition 2.2.5.** (The Representation Problem (RP) [29]):
*Given a group $G$ with elements $g_1, ..., g_k \in G$ , and given $h \in G$. Find whether there is a representation of $h$ with respect to $(g_1, ..., g_k)$.*

The hardness of the representation problem relies on the DLP. In 1997, Camenisch and Stadler were the first to use the Representation Problem. They have used it in their construction together with two new DLP related complexity assumptions referred to as "Double Discrete Logarithm Problem" and "e-th Root of Double Discrete Logarithm Problem". The following are their definitions:

**Definition 2.2.6.** (Double Discrete Logarithm Problem (DDLP) [36])
*Given $h, g \in G$ where $G$ is of prime order $p$ and $a \in \mathbb{Z}_p^*$. Find an $x \in \mathbb{Z}_p^*$ if it exists where $h = g^{a^x}$.*

**Definition 2.2.7.** (e-th Root of Double Discrete Logarithm Problem (RDDLP) [36])
*Given $h, g \in G$ where $G$ is of prime order $p$ and $e \in \mathbb{Z}_p^*$. Find an $x \in \mathbb{Z}_p^*$ if it exists where $h = g^{x^e}$.*

Once again these problems are not stronger than DLP in the group $G$.
Bilinear Maps have been a tool used in breaking schemes based on Diffie–Hellman problems. However, they have been used in a constructive manner for the first time by Joux work in [77]. In this thesis we will be using it to build schemes rather than break them. The following is a definition of Bilinear Maps, the Bilinear Diffie Hellman problem and the Decisional Bilinear Diffie Hellman.

**Definition 2.2.8.** (Bilinear Maps [49]):
*Let $G_1$,$G_2$ and $G_3$ be three groups of prime order $p$. 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 generators $g_1 \in G_1$, $g_2 \in G_2$ 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 2.2.9.** (Bilinear Diffie Hellman Problem (BDH) [21]):
*Let $e : G_1 \times G_2 \rightarrow G_3$ be an admissible bilinear map. Given generators $g_1 \in G_1$, $g_2 \in G_2$, $A$, $B$ and $C$ where $A = g_2^a$, $B = g_2^b$, $C = g_2^c$ and $a, b, c \in \mathbb{Z}_p^*$. Compute $e(g_1, g_2)^{abc}$.*

**Definition 2.2.10.** (Decisional Bilinear Diffie Hellman Problem (DBDH)):
*Let $e : G_1 \times G_2 \rightarrow G_3$ be an admissible bilinear map. Given generators $g_1 \in G_1$, $g_2 \in G_2$, $A$, $B$, $C$ and $Z$ where $A = g_2^a$, $B = g_2^b$, $C = g_2^c$, $Z = e(g_1, g_2)^z$ and $a, b, c, z \in \mathbb{Z}_p^*$. Decide whether $z$ is random or $z = abc$ .*

We should point out that in this thesis we refer to a security parameter $k$ that is used in deciding the prime ordering of the groups in the bilinear map. The way $k$ is used is beyond the scope of this thesis [63, 60]. However, we need to refer to $k$ rather than $p$ in order to be more accurate with what determines the security of the scheme when a bilinear map is used. The reader is referred to papers on implementing bilinear maps using Weil or Tate pairing [63, 60] for more details. Once again, the pairings we mentioned earlier can be implemented in elliptic curve groups. The complexity of BDH and DBDH have been studied widely [64, 44] and are known to be hard if the right elliptic curve group is chosen. We treat bilinear maps as black boxes throughout the thesis [65].
In EuroCrypt'04 Boneh and Boyen used a more advanced complexity assumption referred to as $q$-Strong Diffie–Hellman Problem [19], where $q$ is a size of a tuple given as input to the problem (see Definition 2.2.11). Their work is a major building block used in schemes constructed in this thesis. Therefore we define the complexity assumption they have proposed together with the equivalence theorem introduced in their work [19]. Assume $e : G_1 \times G_2 \rightarrow G_3$ has been created using parameter $k$. Let $G_1, G_2$ and $G_3$ be cyclic groups of prime order $p$, with a computable isomorphism $\psi$ from $G_2$ to $G_1$ or possibly $G_1 = G_2$. Assuming the generators $g_1 \in G_1$, and $g_2 \in G_2$.

**Definition 2.2.11.** ($q$-Strong Diffie–Hellman Problem ($q$-SDH) in $G_1$ and $G_2$ [19]):
*Given a $(q + 2)$ tuple $(g_1, g_2, g_2^{\gamma}, g_2^{\gamma^2}, ..., g_2^{\gamma^q})$ as an input where $\gamma \in \mathbb{Z}_p^*$, output what is called a SDH pair $(g_1^{1/(\gamma+x)}, x)$ for an $x \in Z_p^*$.*

An algorithm $A$ has a negligible advantage in successfully solving $q$-SDH in $(G_1, G_2)$ if:

$$Adv(p) = |Pr[A(g_1, g_2, g_2^{\gamma}, g_2^{\gamma^2}, ..., g_2^{\gamma^q}) = (g_1^{1/(\gamma+x)}, x)] - 1/|G|| \leq \varepsilon$$

where the probability is over a random choice of a generator $g_2$ (with $g_1 = \psi(g_2)$), and of random bits of $A$.

The advantage $Adv(p)$ is bounded to the prime number $p$ and is equivalent to the probability of the algorithm $A$ calculating the SDH pair from the $(q+2)$ tuple. This problem is believed hard to solve in polynomial time and $\varepsilon$ should be negligible [19]. Throughout this thesis we will be using q-SDH where q is roughly an upper bound on the number of users in our proposed cryptosystem (See Chapter 5).

In [44], Cheon gave an interesting analysis of the q-SDH problem. He investigated some known elliptic curve parameters and found that either $p-1$ or $p+1$ has many small divisors for the largest prime divisor $p$ of its order for each elliptic curve studied. Cheon gave an example of a broadcast encryption[3] [24]. If the elliptic curve $E^+$ was chosen over $GF(3^{155})$ in such an encryption, the secret key can be computed in $O(2^{59})$ exponentiations (resp. $O(2^{42})$) when number of users is $2^{32}$ (resp. $2^{64}$), rather than $O(2^{76})$ group operations. The scenarios in this thesis (Chapter 5) are similar. The number of users required whether $2^{32}$ or $2^{64}$ is ludicrous. Implications for the security parameters will exist, but they are relatively modest.

In the same paper [44], Cheon summarized relationships between problems by denoting $A \geq B$ when problem $B$ can be solved in polynomial time with polynomially many queries to the oracle to solve the problem $A$. The following is part of his summary: $DLP \geq DH \geq DDH \geq q-SDH$.

**Assumption 2.2.12.** *For $q$ relatively small (say no more that the number of living human beings), it is possible to choose reasonable security parameters $k$ such that the q-SDH problem is unfeasibly hard in appropriate groups of size $k$.*

**Theorem 2.2.13.** (Boneh–Boyen SDH Equivalence [19])
*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 [19] 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.*

Later the same year in Crypto'04 Boneh, Boyen and Shacham proposed another complexity assumption referred to as the Decision Linear Problem(DeLP) [20]. The DeLP is considered a hard problem [20] and was used in proving their proposed schemes secure. Using that assumption they proposed an encryption scheme and a signature scheme. The definition of that problem is shown below:

**Definition 2.2.14.** (Decision Linear Problem (DeLP) [20]):
*Let $G$ be a group of prime order $p$ and $u, v, h \in G$ be generators in that group. Given $u^a, v^b, h^c \in G$ as an input where $a, b, c \in \mathbb{Z}_p^*$, decide whether or not $a + b = c$.*

[3] An encryption scheme that has one public key for encrypting and $n$ decrypting keys. The system has $n$ users capable of decrypting the same ciphertext. Each user has a different private key and all $n$ private keys correspond to the one public key.

In 2005, Ateniese *et al.* proposed an interactive version of the $q$-SDH assumption [4]. By interactive we mean giving an adversary that is trying to solve the complexity assumption access to some oracle. This new interactive complexity assumption is called the Extended Diffie Hellman (EDH). Assume $A$ represents an adversary trying to solve the EDH problem for some $x \in \mathbb{Z}_p^*$, $A$ sends $c_i \in \mathbb{Z}_p^*$ to the oracle and receives the response $(g_1, g_2^{1/(x+v_i)}, g_2^{1/(c_i+v_i)})$, for a $v_i \in_R \mathbb{Z}_p^*$. Given access to such an oracle and given the tuple $(g_1, g_1^x, g_2, g_2^x)$ the advantage of adversary $A$ in solving the EDH problem is equivalent to the probability of finding a tuple $(c, a, a^x, a^v, g_2^{1/(x+v)}, g_2^{1/(v+c)})$ for some $a \in G_1$. A formal definition is shown below:

**Definition 2.2.15.** (Extended Diffie–Hellman (EDH) [4]): *Given a bilinear map $e : G_1 \times G_2 \to G_3$. Let $x \in \mathbb{Z}_p^*$. Let oracle $O_x(.)$ take input $c_i \in \mathbb{Z}_p^*$ and produce output $(g_1, g_2^{1/(x+v_i)}, g_2^{1/(c_i+v_i)})$, for a random $v_i \in \mathbb{Z}_p^*$. Find a tuple $(c, a, a^x, a^v, g_2^{1/(x+v)}, g_2^{1/(v+c)})$ where $c$ has not been queried in $O_x(.)$*

The EDH problem is hard if for all probabilistic polynomial-time adversaries $A$, all $v, c \in \mathbb{Z}_p^*$ and all $a \in G_1$ such that $a \neq 1$,
$Adv(p) = Pr[x \in_R \mathbb{Z}_p^* : A^{O_x}(g_1, g_1^x, g_2, g_2^x) = (c, a, a^x, a^v, g_2^{1/(x+v)}, g_2^{1/(v+c)}) \wedge c \notin Q] < \varepsilon$.
where $Q$ is the set of queries $A$ makes to oracle $O_x(.)$, and $\varepsilon$ is negligible.
The EDH is not the only interactive complexity assumption proposed in Ateniese *et al.*'s paper. They had two other complexity assumptions that are interactive and they are based on the Symmetric External Diffie–Hellman (SXDH) and the LRSW complexity assumption[4]. We shall define the SXDH then define the interactive versions (Referred to as Strong LRSW and Strong SXDH). These complexity assumptions will be used in the group signature in Section 4.5.2 which is an example of dynamic group signatures and was proposed by Ateniese *et al.*'s [4]. To define the SXDH assume a bilinear map $e : G_1 \times G_2 \to G_3$ where $G_1, G_2$ and $G_3$ are of prime order and $k$ is the security parameter that determines the order of such groups.

**Assumption 2.2.16.** (Symmetric External Diffie–Hellman (SXDH) [4]):
*The Decisional Diffie–Hellman problem DDH is hard in both $G_1$ and $G_2$. Assume that there do not exist efficiently computable isomorphisms $\psi : G_1 \to G_2$ or $\bar{\psi} : G_2 \to G_1$.*

The SXDH assumption has been used in this thesis to explain the security of a particular group signature scheme that we have chosen from literature as an example of dynamic groups (See Section 4.5.2). We do not use this assumption in any of our main results in Chapters 5 and 6.
In the Strong SXDH an adversary is given access to two oracles. Let $x, y \in \mathbb{Z}_p^*$. Both oracles take an input of form $m \in \mathbb{Z}_p^*$ however the response is different. The first oracle outputs $(g_1, g_2^{1/(x+v)}, g_2^{1/(v+m)})$ for a random $v \in \mathbb{Z}_p^*$ while the second oracle outputs $(g_1^r,$

[4]Complexity assumption proposed by Lysyanskaya, Rivest, Sahai,and Wolf in [91] and therefore the abbreviation.

$g_1^{ry}$, $g_1^{rv}$, $g_2^{1/(y+v)}$, $g_2^{1/(v+m)}$) for a random $r, v \in \mathbb{Z}_p^*$. The adversary $A$ has an advantage in breaking the complexity assumption, Strong SXDH, if and only if given the triple $(g_1, g_1^x, g_2)$ it can not distinguish whether the second oracle's responses where based on $x$ or $y$ the main reason for not being able to differentiate is because $v$ changes every time the oracles are queried. A formal definition is given below:

**Definition 2.2.17.** (Strong Symmetric External Diffie Hellman (sSXDH) [4]): *Let $g_1 \in G_1$, $g_2 \in G_2$ and $x, y \in \mathbb{Z}_p^*$. Let $O_x(.)$ be an oracle that takes as input $m \in \mathbb{Z}_p^*$ and outputs $g_1, g_2^{1/(x+v)}, g_2^{1/(v+m)}$ for a random $v \in \mathbb{Z}_p^*$. Let $Q_y(.)$ be an oracle that takes the same input type and outputs $(g_1^r, g_1^{ry}, g_1^{rv}, g_2^{1/(y+v)}, g_2^{1/(v+m)})$ for a random $r, v \in \mathbb{Z}_p^*$. Decide using a given triple $(g_1, g_1^x, g_2)$ whether the second oracle responses are based on $x$ or $y$.*

The sSXDH is considered hard if for all probabilistic polynomial-time adversaries $A^{(.)}$, and for randomly chosen $g_1 \in G_1$, $g_2 \in G_2$, and $x, y \in \mathbb{Z}_p^*$,

$$Adv(k) = |Pr[A^{O_x,Q_x}(g_1, g_1^x, g_2) = 1] - Pr[A^{O_x,Q_y}(g_1, g_1^x, g_2) = 1]| < \varepsilon,$$

where $k$ is a security parameter and $\varepsilon$ is negligible.
The last interactive complexity assumption we will be using in this thesis is the Strong LRSW. An adversary is given $X$, $Y$, $g_1$, and $g_2$ where $X = g_2^x$ and $Y = g_2^y$ for $x, y \in \mathbb{Z}_p^*$. It is also given access to an oracle that outputs the tuple $(a, a^x, a^{y+yxm}, a^m, a^{mx})$ for a random $a \in G_1$ when queried for value $m \in \mathbb{Z}_p^*$. The adversary has a negligible advantage in finding a tuple $(a, a^x, a^{y+yxm}, a^m, a^{mx})$ for an $m$ that has never been queried before. The formal definition follows:

**Definition 2.2.18.** (Strong LRSW [4]):
*Let $X, Y \in G_2$, $X = g_2^x$, $Y = g_2^y$. Let $O_{X,Y}(.)$ be an oracle that takes as an input a value $m \in \mathbb{Z}_p^*$ and outputs what is called a LRSW-tuple $(a, a^x, a^{y+yxm}, a^m, a^{mx})$ for a random $a \in G_1$. Find a tuple $(a, a^x, a^{y+yxm}, a^m, a^{mx})$ for an $m$ that has never been queried before.*

The Strong LRSW is a hard problem if for all probabilistic polynomial-time adversaries $A^{(.)}$ and all $m \in \mathbb{Z}_p^*$.

$$Adv(k) = Pr[x \in_R \mathbb{Z}_p^*, y \in_R \mathbb{Z}_p^*, X = g_2^x, Y = g_2^y, (a_1, a_2, a_3, a_4, a_5) \leftarrow$$
$$A^{O_{X,Y}}(g_1, g_2, X, Y) : m \notin Q \wedge a_1 \in G_1 \wedge a_2 = a_1^x \wedge a_3 = a_1^{y+yxm} \wedge a_4 = a_1^m \wedge a_5 = a_1^{mx}] < \varepsilon,$$

where $Q$ is the set of queries $A$ makes to $O_{X,Y}$, $k$ is a security parameter and $\varepsilon$ is negligible.
Note that while querying the oracles the last two elements $a^m$ and $a^{mx}$ are computable from the first two elements $a$ and $a^x$, therefore no need to have them as part of the output. We have included them in order to be consistent with the challenge.

## 2.3 Encryption Schemes

Alice wants to send a confidential message to Bob. She converts the message to an unreadable text and sends it to Bob. The conversion is referred to as encryption. The message is called plaintext and the encrypted message is named ciphertext. Bob needs to derive the plaintext from the ciphertext. To do so he will have extra secret information referred to as the key. The procedure of changing a ciphertext back to a plaintext is referred to as decryption. Only people with the key can perform the encryption and decryption. If the key for encrypting is the same as the key for decrypting then the scheme is said to be symmetric otherwise the encryption scheme is said to be asymmetric. In this thesis the latter is more relevant to the constructions contributed in Chapter 5.
In this section a formal definition of an encryption scheme based on asymmetric cryptography is given together with two examples of constructions. Examples are chosen depending on relevance to the thesis.

**Encryption Scheme:** An asymmetric encryption scheme is a set of algorithms. To define the scheme we explain the algorithms.

- $KeyGen(k)$ : This algorithm creates two keys a public key, $pk$, known to all and a secret key, $sk$, given to the decryptor only. The input is a security parameter $k$.
- $Encrypt(M, pk)$ : This algorithm is run by the sender (Alice). She encrypts the message $M$ using the public key $pk$ and outputs a ciphertext $C$.
- $Decrypt(C, sk)$ : This algorithm is run by the receiver (Bob). He decrypts the ciphertext $C$ using the secret key $sk$ and derives the message $M$.

Sometimes for the encryption algorithm an extra input which is referred to as the randomizing coin is added to make the encryption more secure. In literature most definitions of encryption do not have that element as an explicit input. Many encryption schemes are in literature and it is hard to cover them all. We picked two which are used in the construction of the attribute based authentication scheme in Chapter 5.

### 2.3.1 Elgamal Encryption Scheme

In 1984, Elgamal [61] came up with a public key encryption scheme that is still used today. The security of Elgamal is dependent on the "Decisional Diffie Hellman". The prefix $EG$ is used to differentiate between the algorithms in this section and other algorithms. The algorithms of Elgamal are defined below:

- $EG.KeyGen(p)$ : A public key is $h, g \in G$ for $G$ is a multiplicative cyclic group of order $p$. The private key is the exponent $x \in \mathbb{Z}_p^*$ such that $h = g^x$.

- $EG.Encrypt(M, sk)$ : To encrypt a message $M$ choose a random element $\alpha \in \mathbb{Z}_p^*$ and output the pair $C = \langle C_1, C_2 \rangle$ where $C_1 = g^\alpha$ and $C_2 = M.h^\alpha$.
- $EG.Decrypt(C, pk)$ : To decrypt compute $M = C_2/C_1^x$.

The next encryption scheme defined is the Linear Encryption Scheme.

### 2.3.2 A Linear Encryption Scheme

In [20] the authors proposed an encryption scheme named the "Linear Encryption Scheme" since its security depends on the difficulty of the "Decision Linear Diffie–Hellman Assumption". The scheme was a building block in their cryptosystem. The notation $LE$ is used to distinguish its algorithms from others in this thesis. The algorithms are described below:

- $LE.KeyGen(p)$ : In a linear encryption scheme a user's public key is $u, v, h \in G_1$. The private key is the exponents $\xi_1, \xi_2 \in \mathbb{Z}_p^*$ such that $u^{\xi_1} = v^{\xi_2} = h$. Note that $p$ is a prime number, $G_1$ is a group of prime order $p$.
- $LE.Encrypt(M, sk)$ : To encrypt a message $M$ choose random elements $\zeta, \beta \in \mathbb{Z}_p^*$ and output the triple $\langle C_1, C_2, C_3 \rangle = \langle u^\zeta, v^\beta, Mh^{\zeta+\beta} \rangle$.
- $LE.Decrypt(C, pk)$ : To decrypt compute $C_3/(C_1^{\xi_1} C_2^{\xi_2})$.

The following section introduces digital signature schemes as a concept with examples of constructions that are relevant to the constructions in Chapter 5.

## 2.4 Digital Signatures

Alice wants to send a signed document to Bob. Alice has a public key that identifies her and is known to everyone and she has a secret key that no one knows but her. Using the secret key and a message Alice can create a signature and send it to Bob. Bob can verify the signature using the message and the public key. The signature contains several elements that look random to Bob. Among these elements is one which is referred to as a "Fingerprint". Bob knows the verifying procedure which will enable him to make use of the public key and the signature in order to recalculate the "Fingerprint". If what he calculated is equivalent to what he got from Alice then accept signature otherwise reject it.

**Digital Signature Scheme:** A Digital Signature Scheme is a set of algorithms. To define the scheme we explain the algorithms.

- $Setup(k)$ : This algorithm creates two keys using a security parameter $k$. The first is the public key $pk$ known to all and the second is a secret key $sk$ given to the signer only.

- $Sign(M, sk)$ : This algorithm is run by the signer (Alice). She signs the message $M$ using a secret key $sk$ and outputs a signature $\sigma$.

- $Verify(\sigma, pk, M)$ : This algorithm is run by the verifier (Bob). He uses $\sigma$ and the public key $pk$ to run the verification algorithm that will output either accept or reject.

### 2.4.1 RSA

Rivest, Shamir, and Adleman came up with a signature scheme that is still used today [118]. Factoring is the underlying, presumably hard problem which the security of RSA is based on. The algorithms of the signature scheme are described below:

- $RSA.Setup(k)$: Two distinct large random prime numbers are chosen $p$, and $q$. The size of them is represented by the security parameter $k$. Let $n = pq$. $\phi(n) = (p-1)(q-1)$. $e$ is chosen such that $1 < e < \phi(n)$ and $e$ and $\phi(n)$ are coprime. $d$ is computed to satisfy $e.d \equiv 1(mod\phi(n))$. The public key is $pk = (n, e)$ and the private key is $sk = (d, p, q)$.

- $RSA.Sign(M, sk)$: The signer represents the message $M$ as a number between $0 < M < n$. Signer calculates $c = M^d(modn)$ and sends $c$ along with the message as their signature.

- $RSA.Verify(\sigma, pk, M)$: Verifier calculates $c^e(modn)$ and compares it to the message. If they are equal then accept the signature otherwise reject it.

### 2.4.2 Schnorr

Schnorr came up with a signature that has been proven secure under the discrete logarithmic problem assumption[5] [122]. The algorithms are as follows:

- $Sc.Setup(k)$: Given the security parameter $k$, choose a random secret key $sk$ where $0 < sk < k$. The public key will be $pk = g^{sk}$, where $g \in G$ and $G$ is a group such that DLP is hard. Finally, a hash function $H$ is chosen (See Section 3.3.1). So $(g, pk, H)$ are public and $sk$ is private.

- $Sc.Sign(M, sk)$: The signer chooses a random $0 < x < k$. Calculate $r = g^x$ and $e = H(M||r)$, where $M$ is the message to be signed. Let $s = (x - e.sk)modk$. Note that both $e, s \in [0, k[$. They are sent to the verifier as the signature (i.e $\sigma = (e, s)$).

- $Sc.Verify(\sigma, pk, M)$: Verifier calculates $\bar{r} = g^s pk^e$. Then he calculates $\bar{e} = H(M||\bar{r})$. If $\bar{e} = e$ then accept the signature otherwise reject it.

[5] Secure under the Random Oracle assumption 3.3

### 2.4.3 Camenisch–Lysyanskaya

Camenisch–Lysyanskaya is a signature scheme secure under the LRSW assumption[6] [35]. In [4] they extended the scheme to be secure under the SXDH and the Strong LRSW. Note that in this scheme the message signed is unknown to verifier.

- $CL.Setup(k)$: Given a security parameter $k$, create a bilinear map $e : G_1 \times G_2 \rightarrow G_3$ where $G_1$, $G_2$, and $G_3$ are of prime order $p$ (chosen according to $k$). That is $S_{pub}^{CL} = (p, G_1, G_2, G_3, g_1, g_2)$, where $g_1 \in G_1$, and $g_2 \in G_2$. Let $s, t \in_R \mathbb{Z}_p^*$. The public key is $pk = (g_2^t, g_2^s)$ and the private key $sk = (s, t)$

- $CL.Sign(M, sk)$ : Inputs to the sign algorithm are a message and the secret key. Choose a random $a \in G_1$. Output $\sigma$=($a$,$a^t$,$a^{s+stM}$,$a^M$,$a^{Mt}$) as a signature .

- $CL.Verify(\sigma, pk)$ : To accept a signature $\sigma = (A, B, C, D, E)$ verify the equalities: (1) $e(B, g_2) = e(A, g_2^t)$. (2) $e(D, g_2^t) = e(E, g_2)$. (3) $e(C, g_2) = e(A, g_2^s).e(E, g_2^s)$

### 2.4.4 Boneh–Boyen

In [4] a modified version of Boneh and Boyen weak signature scheme [19] was used. The security of the $BB^+$ scheme relies on the EDH assumption[7]. Algorithms of the scheme are described as follows:

- $BB.Setup(k)$ : Given a security parameter $k$, the public system parameters are created by setting up a bilinear map $e : G_1 \times G_2 \rightarrow G_3$, such that $G_1$, $G_2$, and $G_3$ are of prime order $p$ (chosen according to $k$). The system parameters are $S_{pub}^{BB} = (p, G_1, G_2, G_3, g_1, g_2)$, where $g_1 \in G_1$, $g_2 \in G_2$. A random private key $sk$ is chosen from $\mathbb{Z}_p^*$. The public key is $pk = (g_1, g_1^{sk}, g_2)$.

- $BB.Sign(sk, M)$ : On input of a message $M$ and the secret key select a random $r \in \mathbb{Z}_p^*$ and output $\sigma = (g_1^r, g_2^{1/(sk+r)}, g_2^{1/(r+M)})$.

- $BB.Verify(pk, M, \sigma)$ : Input is the public key, a message and a signature of the form $(A, B, C)$. Accept signature if (1) $e(g_1^{sk}A, B) = e(g_1, g_2)$, and (2) $e(Ag_1^M, C) = e(g_1, g_2)$.

All four signatures introduced in this section have been used in constructing either group signatures in Chapter 4 or attribute authentication schemes in Chapter 5. Different protocols will also be used and are described in the next section.

[6] Secure without the Random Oracles 3.3.

[7] This Signature scheme is existentially unforgeable under adaptive chosen message attack

## 2.5 Cryptographic Protocols for Knowledge Proofs

Identification is the process of verifying that a person is who they claim they are. Let's assume that Alice wants to prove to Bob that she is Alice. They engage in some form of communication, at the end of which Bob should be convinced about Alice's identity. This implies that Alice will be giving some sort of information to Bob during the communication. However she is cautious about what information she gives because Bob or an eavesdropper can use it in impersonating Alice in other communications
Goldwasser, Micali and Rackoff proposed interactive proofs as a solution [68]. Their system is an abstract machine that models computation as the exchange of messages between two parties. Messages are sent between Alice and Bob until he is convinced it is Alice who he is dealing with. In their work they were looking for a system that guarantees no important information is leaked. This led to the discovery of Interactive Zero Knowledge proofs (IZK).
Blum, Feldman, and Micali showed that with an existence of a common random string shared between the prover and the verifier a computational zero-knowledge can be built without requiring interaction [17].
Extractable Zero Knowledge proofs are protocols followed to prove knowledge without leak of information, until in a later stage the prover publishes a piece of data that can extract the information from the elements of the zero knowledge.
In the following sections we give more definitions and examples of such proofs. The examples we give are used as building blocks in different cryptosystems including the ones proposed in this thesis.

### 2.5.1 Commitment Schemes

Making a commitment means that a player in a protocol is able to choose a value from some (finite) set and commit to his choice such that he can no longer change his mind. He does not have to reveal his choice - although he may choose to do so at some later time. It is the digital analogue for a locked box, that is sent by committer to a verifier. The key can be sent later to the verifier to reveal the committed value.
A commitment scheme is usually used in the construction of zero knowledge proofs, authentication schemes and in other cryptographic systems. An example of such a scheme is the one proposed by Pedersen in [109]. The scheme is defined as follows:

**Definition 2.5.1.** (Pedersen Commitment Scheme (PCOM) [109]):
*Let generators $h, g \in G$ where $G$ is of prime order $p$ and $\log_g h$ is unknown to the committer. The committer commits to $s \in \mathbb{Z}_p^*$ by choosing $t \in \mathbb{Z}_p^*$ randomly and sending $PCOM(s) = g^s h^t$. The commitment is revealed by publishing $s$ and $t$.*

A Commitment Scheme is said to be:

- Information Theoretically Binding if the sender can not change the value he committed to no matter how much computing power is available.
- Computationally Binding if the sender needs a huge amount of computation to change the value committed too.
- Information Theoretically Concealing if the receiver or an eavesdropper are unable to determine the value being committed until the revealing stage occurs and that is no matter what computational powers they have.
- Computationally Concealing if the receiver needs a huge amount of computation to retrieve the value committed before the revealing stage occurs.

In [109] Pedersen proved the commitment scheme to be computationally binding and concealing by comparing it to hardness of the DLP (Definition 2.2.1). The next section discusses zero knowledge proofs in more detail.

### 2.5.2 Zero Knowledge Protocols (ZKP)

There are two types of zero knowledge proofs: interactive and non-interactive.

#### Interactive Zero Knowledge Protocols (IZKP)

Alice and Bob (referred to as Peggy and Victor in lots of papers) run an interactive protocol between themselves. Bob has a polynomially bounded computational power. Alice proves to Bob that she knows an answer to an NP problem without disclosing that answer. Consider the following scenario [114]:

**Scenario 2.5.2.** *Alice wants to prove to Bob that she knows the password to a barrier in a cave as shown in Figure 2-1. She goes in the cave where Bob can not see her. He shouts out the direction he wants her to come out from. If she does not know the password she has a 50% chance that she is in the side that Bob requested and therefore she will come out from the right direction. To reduce the 50% chance of Alice being lucky the protocol is done $n$ times making the probability of Alice being lucky be* $1/2^n$

Knowledge of proofs should be:

- Complete: The ability of the prover (Alice) to convince the verifier (Bob) of the validity of any true assertion. In other words if the assertion is true Alice should be able to prove that with probability 1.
- Sound: No prover strategy can trick the verifier (Bob) to accept a false assertion. In other words if Alice does not know the thing she is proving then the probability of Bob accepting her proof is small.

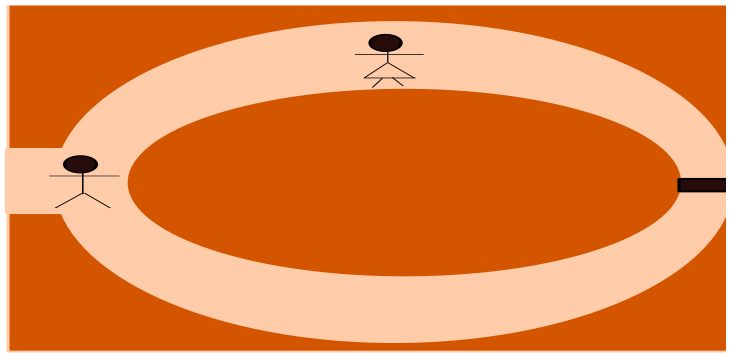

Figure 2-1: Knowledge of Proof

Assume Bob wants to prove to others that Alice knows the secret of the barrier in the cave. He can video tape the procedure and send it to anyone. The video tape is considered a view or transcript of the proof. Bob shows the video to a third person, Carol, to convince her that Alice knows the secret. Carol will not accept such a proof because the video can be faked, if Alice did not know the secret, by editing out the times Alice got it wrong. A simulator is a procedure that generates fake views of the proof (generated without the prover) that are indistinguishable from a genuine view (generated by the prover) of the proof [98]. So now we can formally define the ZKP as follows:

**Definition 2.5.3.** (Zero Knowledge Proof): *A proof of knowledge is considered to be zero knowledge if it has a simulator for the proof.*

We shall give an example of an IZKP for the Discrete Logarithm Problem. Alice knows the discrete logarithm of $A$ to the base $g$. Assume $A = g^a$ then Alice knows $a$ and wants to follow a IZKP with Bob to prove so. The protocol runs as follows:

- Alice chooses a random $r$ and sends Bob $R = g^r$.
- Bob chooses a random bit $b$ and sends it to Alice.
- Alice computes $s = r + ab$ and sends $s$ to Bob.
- Bob checks that $g^s = RA^b$,

The previous IZKP is sound, complete and maintains the zero knowledge property as Schnorr has proven in [122].
In the schemes in Chapters 5 and 4 different protocols are proposed to achieve the objectives of our thesis and these protocols use concepts from zero knowledge. The schemes do not just use interactive zero knowledge, they use the non interactive notion too.

**Non-Interactive Zero Knowledge (NIZK)**

A NIZK proof consist of three entities prover, verifier and a uniformly selected reference string $R$ (which can be selected by a trusted third party). The verifier and prover can access that string and each tosses additional coins. Several properties of non interactive zero knowledge proofs were proposed in literature for example:

- Non-Malleability is a security notion in NIZK. If whatever one can prove after seeing an NIZK proof, one can prove before seeing it (except for the ability to duplicate the proof), then the NIZK is said to be non-malleable. An adversary can not prove assertions it did not previously know using a NIZK proof.
- Adaptive NIZK is a security notion that is equivalent to proving the NIZK Non-Malleable even after giving the adversary access to an oracle that enables it to request proofs of theorems of its choice.
- The simulation-soundness requirement is a notion first proposed by Sahia in [120]. It says that a polynomially-bounded prover can not prove false theorems even after seeing simulated proofs of any statements (including false statements) of its choosing.

In the next section the term signature of knowledge is explained which is closely related to zero knowledge proofs.

### 2.5.3 Signature of Knowledge of Discrete Logarithmic problems

Signatures of knowledge as Camenish and Stadler named it in [36] is a method to prove the knowledge of a signature. Informally, it means the prover knows the secret used in signing a message. The interactive version of these proofs is a ZKP. The following signature of knowledge is inspired by Schnorr signature scheme (See Section 2.4.2). It also relies on the complexity assumption the Representation Problem defined in Definition 2.2.5.

**Definition 2.5.4.** (Signature of Knowledge of the Discrete Logarithm:)
*Given a group $G$ of prime order and $y, g \in G$. A pair $(c, s) \in \{0,1\}^k \times \mathbb{Z}_n^*$ satisfying $c = H(m||y||g||g^s y^c)$ is a Signature of Knowledge of the discrete logarithm of $y$ to base $g$ on message $m$.*

This signature can be computed if the secret $x$ is known and it is done by choosing a random $r \in \mathbb{Z}_n^*$ and computing: $c = H(m||y||g||g^r)$ and $s = r - cx(modn)$. This could be extended to include knowledge of a representation (See Section 2.2.4) as follows:

**Definition 2.5.5.** (Signature of Knowledge of Representation):
*A signature of the knowledge of representation of $y_1, ..., y_w$ with respect to bases $g_1, ..., g_v$*

*on message $m$ is denoted as follows:*
$SKREP[(\alpha_1, ..., \alpha_u) : (y_1 = \prod_{j=1}^{l_1} g_{b_{1,j}}^{\alpha_{e_{1,j}}}) \wedge ... \wedge y_w = \prod_{j=1}^{l_w} g_{b_{w,j}}^{\alpha_{e_{w,j}}})]$,
*where indexes $e_{i,j} \in \{1, ..., u\}$ refer to $\alpha_1, ..., \alpha_u$ and indexes $b_{i,j} \in \{1, ..., v\}$ refer to $g_1, ..., g_v$. The signature refers to $(c, s_1, ..., s_u) \in \{0,1\}^k \times \mathbb{Z}_n^u$ satisfying*
$c = H(m||y_1||...||y_w||g_1||...||g_v||\{\{e_{i,j}, b_{i,j}\}_{j=1}^{l_i}\}_{i=1}^{w}||y_1^c \prod_{j=1}^{l_1} g_{b_{1,j}}^{s_{e_{1,j}}}, ..., y_w^c \prod_{j=1}^{l_w} g_{b_{w,j}}^{s_{e_{w,j}}})$ .

This could only be calculated by knowing $(\alpha_1, ..., \alpha_u)$ and that is done by choosing $u$ random elements $r_i \in \mathbb{Z}_n$. Then calculating :
$c = H(m||y_1||...||y_w||g_1||...||g_v||\{\{e_{i,j}, b_{i,j}\}_{j=1}^{l_i}\}_{i=1}^{w}|| \prod_{j=1}^{l_1} g_{b_{1,j}}^{r_{e_{1,j}}}, ..., \prod_{j=1}^{l_w} g_{b_{w,j}}^{r_{e_{w,j}}})$
Finally, calculate $s_i = r_i - c\alpha_i (mod n)$.
Note that $\prod_{j=1}^{l_i} g_{b_{i,j}}^{r_{e_{i,j}}} = \prod_{j=1}^{l_i} g_{b_{i,j}}^{s_{e_{i,j}} + c\alpha_{e_{w,j}}} = y_i^c \prod_{j=1}^{l_i} g_{b_{i,j}}^{s_{e_{i,j}}}$
The distribution and creation of elements $l_i$ were not discussed in [36]. In this thesis we can assume that $l_i \in \mathbb{Z}$ are chosen randomly by the creator of the signature and sent as part of the proof of knowledge.

**Definition 2.5.6.** (Signature of Knowledge of the Double Discrete Logarithm)
*Let $l \leq k$ where $l$ and $k$ are security parameters. An $(l+1)$ tuple $(c, s_1, ..., s_l) \in \{0,1\}^k \times \mathbb{Z}^l$ satisfying the equation:*
$c = H(m||y||g||a||b_1||...||b_l)$ *with*

$$b_i = \begin{cases} \text{If } c[i] = 0\text{; return } g^{(a^{s_i})} \\ \text{Otherwise return } y^{(a^{s_i})} \end{cases}$$

*is a signature of the knowledge of a double discrete logarithm of $y$ to the bases $g$ and $a$, and is denoted $SKLOGLOG[\alpha : y = g^{a^\alpha}](m)$*

$SKLOGLOG[\alpha : y = g^{a^\alpha}](m)$ can be computed only if the double discrete logarithm $x$ of the group element $y$ to the bases $g$ and $a$ is known (i.e. $y = g^{a^x}$). Assume there is an upper bound $\lambda$ on the length of $x$. Let $\epsilon > 1$ be a constant. Compute the values $b_i^* = g^{a^{r_i}}$, for $i = 1, ...l$ and $r_i \in_R \{0, ..., 2^{\epsilon\lambda} - 1\}$. $c = H(m||y||g||a||b_1^*||...||b_l^*)$. Finally,

$$s_i = \begin{cases} \text{If } c[i] = 0\text{; return } r_i \\ \text{Otherwise return } r_i - x \end{cases}$$

Note that if $c[i] = 0$ then $r_i = s_i$ therefore $b_i^* = g^{a^{r_i}} = g^{a^{s_i}} = b_i$; Otherwise $r_i - x$ implies $b_i^* = g^{a^{r_i}} = g^{a^{s_i + x}} = g^{a^{s_i} a^x} = y^{a^{s_i}} = b_i$

**Definition 2.5.7.** (Signature of Knowledge of the e-th Root of Double Discrete Logarithm):
*Let* $(l+1)$ *tuple* $(c, s_1, ..., s_l) \in \{0,1\}^k \times \mathbb{Z}_n^{*^l}$ *satisfy the equation:*
$c = H(m||y||g||e||b_1||...||b_l)$ *with*

$$b_i = \begin{cases} \textit{If } c[i] = 0\textit{; return } g^{(s_i{}^e)} \\ \textit{Otherwise return } y^{(s_i{}^e)} \end{cases}$$

*is a signature of the knowledge of an e-th root of the discrete logarithm of* $y$ *to the base* $g$*, and is denoted* $SKROOTLOG[\alpha : y = g^{\alpha^e}](m)$

$SKROOTLOG[\alpha : y = g^{\alpha^e}](m)$ can be computed only if the e-th root $x$ of the discrete logarithm $y$ to the base $g$ is known (i.e. $y = g^{x^e}$). Compute the values $b_i^* = g^{r_i{}^e}$, for $i = 1, ...l$ and $r_i$ chosen randomly from $Z_n^*$. Let $c = H(m||y||g||e||b_1^*||...||b_l^*)$. Finally,

$$s_i = \begin{cases} \text{If } c[i] = 0\text{; return } r_i \\ \text{Otherwise return } r_i/x(modn) \end{cases}$$

Note that if $c[i] = 0$ then $r_i = s_i$ therefore $b_i^* = g^{r_i{}^e} = g^{s_i{}^e} = b_i$; Otherwise $r_i/x$ implies $b_i^* = g^{r_i{}^e} = g^{(s_i x)^e} = g^{s_i^e x^e} = y^{s_i^e} = b_i$
The definitions of this section are used in the construction of dynamic group signature in Section 4.5.1.

## 2.6 Chapter Summary

Complexity assumptions are mathematical problems that are believed to be intractable and are used in cryptography as building blocks for constructing different schemes. In Section 2.3 we discussed asymmetric encryption. Examples of such encryption schemes are Elgamal and Linear Encryption schemes. In Section 2.4 digital signature is defined. Examples of digital signature are RSA, Schnorr, Camenisch–Lysyanskaya and Boneh–Boyen. All these examples will be used in Chapters 4 and 5 to construct more advanced cryptosystems. Finally, we discussed protocols such as zero-knowledge, signature of knowledge and commitment schemes in Section 2.5.1 which are used in the protocols of the constructions in Chapters 4 and 5.

# Chapter 3

# Provable Security

In this chapter we give two examples of game models, one for encryption and the other for digital signatures. We define the term "Random Oracle" used in provable security often and used in our proofs in later chapters of this thesis. We then define the term "Forking Lemma" a method used in proving security notions of digital signatures and which will be used in proving some of our proposed schemes secure.

## 3.1 Introduction

Previously, cryptosystems offered very little security guarantees. They were designed, in an ad hoc fashion where the system is proposed, attacked, broken, and repaired [54]. Cryptographers were not satisfied with such an approach. They needed the scheme to be proved mathematically secure before being used. In 1949 Shannon came up with the first significant attempt to prove a scheme secure [124]. He came up with the term "perfectly" secure cipher. Informally, we can say an encryption scheme is perfectly secure if a ciphertext does not reveal anything about the plaintext without the key.
The modern approach to proving schemes secure is referred to as "Provable Security". The basis of such a method is to prove that if there were an adversary capable of breaking a certain security concept then that adversary is able to solve a computationally intractable problem. By that we imply that such an adversary solves a complexity assumption that is believed to be hard (See Section 2.2).
Usually the cryptographic system has an adversarial model represented as some game with an adversarial goal. The goal should capture what it means to break a scheme and the game itself represents what capabilities are given to the adversary in order to achieve such a goal.
In such game models the assumption is there exist an adversary that can break the security condition of a scheme. The adversary in this case is a "black box simulation" where assuming that black box is given certain inputs, the output is returned and that output is the result of achieving the adversarial goal. The same simulation is then used

to refute the complexity assumption.
Due to the fact that we are designing new cryptographic paradigms, we need to design such security models that capture security notions in which the new systems require.

## 3.2 Provable Security Examples

In this section we will give examples of designing game models for security proofs. There are a lot of examples in literature. Game models consist of an adversary and the challenger. The adversary, referred to as *Adam* throughout this thesis, is the bad entity in the game model that is trying to break the security condition. The challenger is the good entity in the model who runs certain queries and challenges the adversary to break the condition. We will refer to the challenger as *Charles*.

**Provable Security in Encryption Schemes:** An example of a game model for proving encryption schemes secure is IND-CPA [94] security. IND-CPA stands for IN-Distinguishable Chosen Plaintext Attack. *Adam* and *Charles* agree on the encryption scheme, the plaintext message space $M$ and ciphertext message space $C$ they want to challenge. We say that encryption scheme is IND-CPA secure if and only if there exist no polynomial time adversary that can win the IND-CPA game. By polynomial time adversary we mean that *Adam* must complete the game and output a guess within a polynomial number of time steps. Assume the encryption algorithm is $E$. The IND-CPA game is described as follows:

- *Charles* sets up the system by creating a public key $pk$, and a secret key $sk$ . *Charles* gives $pk$ to *Adam* and retains $sk$.
- *Adam* chooses two messages $M_0$ and $M_1$. They should be the same size or the shorter message should be padded to equalize the size. *Adam* sends the two messages to *Charles*.
- *Charles* randomly chooses $b \in \{0, 1\}$ and encrypts a message $C_b = E(M_b)$. In other words one of the messages is chosen randomly and encrypted. $C_b$ is sent to *Adam* as the challenge.
- *Adam* uses the ciphertext $C_b$ and all his computational ability to choose a $\bar{b} \in \{0, 1\}$ that matches $b$. If $\bar{b} = b$ then *Adam* wins the game else *Adam* loses the game.

The advantage of winning the game is defined by how much better *Adam* can do than a simple random guess (which has a probability 1/2 of being right). In other words, $Adv_{CPA}(k) = |Pr[b = \bar{b}] - 1/2|$ where $k$ is the security parameter used in setting up the encryption scheme .

**Definition 3.2.1.** (Indistinguishable Chosen Plaintext Attack) *An encryption scheme is IND-CPA secure if and only if* $Adv_{CPA}(k) < \varepsilon$ *where* $\varepsilon$ *is negligible and* $k$ *is the security parameter used in setting up the system.*

In literature there are more powerful security notions for encryption schemes other than IND-CPA[1]. For example, we can give the adversary access to a decryption oracle that he queries a number of times before and after the challenge, as long as he does not use the challenge ciphertext for issuing a query. If *Adam* is given access to such an oracle then the encryption scheme is said to be IND-CCA secure. However we have chosen to explain the CPA scheme instead since it will be used as a building block for our provable security proofs in later chapters of this thesis. We have not used CCA since it was hard to integrate the oracles of IND-CCA model with the oracles required in our game models.

**Provable Security in Digital Signature Schemes:** We will give an example of unforgeability of signature schemes security proofs [94]. As in encryption many models exist. In this section we have chosen one example of such security models. We say a digital signature scheme is secure under an adaptive chosen message attack if there exist no polynomial time adversary *Adam* capable of winning the following game:

- *Charles* sets up the system by creating a public key $pk$, and a secret key $sk$ . *Charles* gives $pk$ to *Adam* and retains $sk$.
- *Adam* can query a signature oracle that *Charles* controls. To query such an oracle *Adam* sends a message $M$ and *Charles* replies with a signature $\sigma$.
- *Adam* outputs a pair $(M, \bar{\sigma})$ of message and signature as his forgery, if the message has not been queried before.
- *Charles* verifies the signature $\bar{\sigma}$. If it is valid *Adam* wins and outputs 1 to indicate a successful experiment. Otherwise 0 is the output of the experiment implying *Adam* has failed.

If we refer to the game model as experiment $Exp$ then the advantage of winning the game is represented as $Adv_{CMA}(k) = Pr[Exp = 1]$ where $k$ is the security parameter used in setting up the system.

**Definition 3.2.2.** (Chosen Message Attack) *A digital signature is secure against the chosen message attack if and only if* $Adv_{CMA}(k) < \varepsilon$ *where* $\varepsilon$ *is negligible and* $k$ *is the security parameter used in setting up the system.*

[1] IND-CCA and IND-CCA2 are two other examples [128]

A large amount of research has been done on different game models for the various cryptosystems. In this thesis we will be using similar provable security techniques to prove our schemes secure. However, the two examples given in this section are sufficient to demonstrate the use of such techniques in our proposed proofs.

## 3.3 Random Oracle

Random Oracles are a cryptographic theoretic tool used in proving the security of a scheme. They are frequently used with provable security models. In this section we will be defining the term in more detail. Before we start we need to go through a prerequisite term and that is "Hash Function".

### 3.3.1 Hash functions

Hash functions are deterministic functions which map a bit-string of an arbitrary length to a hashed value which is another bit-strings of a fixed size. The following is a definition of hash functions:

**Definition 3.3.1.** (Hash Functions [128]):
*A hash function $H$ is a function with the following properties below:*

- *The function $H$ takes a message $M$ of finite length represented as a bit-string and maps it to a bit-strings of fixed length. The result is the hash-value or simply the hash of $M$.*
- *Given $H$ and $M \in \{0,1\}^*$, it is easy to compute $H(M)$.*

The properties we require in the hash functions used are defined below:

**Definition 3.3.2.** *Let $H$ be a hash function [128].*

- *Given $y$ is from the range of $H$, the hash function is preimage resistant if it is hard to calculate any $M$ such that $H(M) = y$.*
- *Given $M_0$, the hash function is second-preimage resistant if it is hard to find any $M_1 \neq M_0$ such that $H(M_0) = H(M_1)$.*
- *The hash function is collision resistant if it is hard to find a pair $M_0$ and $M_1$ such that $H(M_0) = H(M_1)$ and $M_0 \neq M_1$.*

Hash functions are used frequently as one of the building blocks of cryptosystems and they affect highly the security of the scheme. For example, in digital signatures, they are used to create "message fingerprints" (See Section 2.4). We will be seeing general usage of hash functions in Chapters 4 and 5. The security of these schemes depend crucially on the hash function.

### 3.3.2 Random Oracle Paradigm

In 1993, Bellare and Rogaway proposed the random oracle model that is used frequently with provable security models [11]. A random oracle is an oracle that when queried responds with a random reply, subject to the condition that the reply is different for various queries but the same when the same input is queried again. Such an oracle has the desired properties such as preimage resistance and collision resistance.
Cryptographers assume that all entities in the game model have access to the random oracle. Then the scheme is proved secure under that random oracle assumption. Finally when it comes to actually implementing the scheme the random oracle is replaced with a strong hash function. Researchers believe that proving a cryptosystem secure under the random oracle is equivalent to proving the security of the scheme dependent on exploiting the hash function used. Even though, Canetti *et al.* [37] proved that this is not necessarily true, cryptographers still consider such an artificial construction acceptable but not preferable.
Any security proof that does not rely on the random oracle model is referred to as a proof under the standard model. Proofs under standard models are considered stronger since no assumption is made. However, the random oracle paradigm yields cryptosystems much more efficient than standard ones while retaining many of the advantages of provable security methods. Therefore random oracle proofs are still acceptable. In order to guarantee efficiency and to be capable of using hash functions, our proposed schemes in Chapter 5 rely on the random oracle assumption.

## 3.4 Forking Lemma

Pointcheval and Stern [111] developed the Forking Lemma as a technique to prove certain security notions of a digital signature scheme under the random oracle assumption. Assume a signature scheme produces the triple $\langle \sigma_0, c, \sigma_1 \rangle$ where $\sigma_0$ takes its values randomly from a set, $c$ is the result of hashing the message $M$ with $\sigma_0$, and $\sigma_1$ depends only on $(\sigma_0, c, M)$. The Forking Lemma is as follows [111]:

**Theorem 3.4.1.** (The Forking Lemma)
*Let $A$ be a Probabilistic Polynomial Time Turing machine, given only public data as input. If $A$ can find, with non-negligible probability, a valid signature $(M, \sigma_0, c, \sigma_1)$ then, with non-negligible probability, a replay of this machine, with the same random tape but a different oracle, outputs new valid signatures $(M, \sigma_0, c, \sigma_1)$ and $(M, \sigma_0, \tilde{c}, \tilde{\sigma_1})$ such that $c \neq \tilde{c}$.*

In later chapters we will be using the Forking Lemma in proving some of our schemes secure.

## 3.5 Chapter Summary

In this chapter the tools used for proving cryptosystems secure were explained. The modern way of such proofs is referred to as "Provable Security". Examples of proving an encryption scheme secure and proving a digital signature scheme secure were given. Later in the chapter we gave the definition of random oracles, hash functions, and forking lemma. These are some primitives used frequently in the "Provable Security" methods.

# Chapter 4

# Group Signature Schemes

In this chapter we will describe two types of group signature schemes that exist in the literature: static and dynamic. We will go through detailed descriptions of the schemes' algorithms, define the major security notions they require and give examples of constructions of such schemes.

## 4.1 Introduction

A group signature scheme is a cryptosystem that is used as a major building block in our proposed scheme (Chapter 5 and 6). The word "group" in the term "group signature" refers to the number of users involved in the system rather than the mathematical group defined in 2.1.2. The following scenario defines the system and its properties informally:

**Scenario 4.1.1.** *Bob manages a company with many employees all of whom form the group of trusted employees. He wants any of the employees to be able to sign on behalf of the group. Bob wants to set up a cryptosystem that enables that. Bob gives each member a private key and announces one public key that represents the group. Any employee can sign a message with the private key they have. Anyone can verify the message with the public key announced. Suppose Alice wants her document signed by the company. She sends it to the group of employees. Any member can sign anonymously on behalf of the group such that Alice can not identify the signer. In case of a dispute, Alice contacts a group manager in Bob's company (usually the role of the group manager is played by either Bob or one of the employees assigned by Bob) and gives him the signature. The group manager has the capability of tracing the signature to a signer and informing Alice that it is not a fraud.*

In 1991, Chaum and Heyst proposed the first group signature scheme in [41]. The scheme had interesting new features. They proposed a scheme that satisfies the following:

1. Only members of the group can sign a message.
2. The verifier can check that the signature is valid and belongs to the group yet she can not tell which member of the group is the signer. Anonymity is important in order to preserve privacy of the signer. For instance, group signatures are used in traffic control systems, where a car sends a message out to other cars saying there is a traffic jam in a specific location. The message is signed and sent out. However, to preserve the location privacy of the car no one can tell the signer other than the group manager yet messages being sent are definitely valid and can not be frauds.
3. In case of dispute the signature can be opened with or without the help of the members of the group and that reveals the identity of the signer. The schemes proposed at that time had a group manager who was responsible for revoking anonymity of signers when needed.

The Chaum–Heyst scheme lacked efficiency since the signature size and the public key size were linearly dependent on the number of members in the group. The consequence of that is the scheme was not suitable for large groups of signers where more memory space is needed to save the key and signature and more bandwidth capacity is required to send them out. Another drawback of Chaum–Heyst scheme is the need to decide the maximum number of members in the group in advance when setting up the system.
Research was done to try improve the scheme [42, 31, 110]. In 1997, Camenisch and Stadler introduced the first efficient scheme [36]. Their scheme had the public key and the signature independent of the number of members. This allowed large groups. They also managed to enable members to join a group after setting up the scheme. In other words, they did not need to recalculate the public key every time a member joins and they did not have to decide a maximum size for the group. Camenisch and Stadler were the first to introduce group signatures (see Section 4.4). The first time the terms "static" and "dynamic" appeared in the literature of group signature [36], the main difference between them was the ability to add members to the dynamic group at any point. Group signatures can be divided according to how a member is added to the group :

- The number of members in the group is pre-decided by an authority and all signing keys are calculated in the set-up stage.
- The authority setting up the system can add a member to the group at any point.
- A member joins the group through engaging in a protocol with the authority responsible for setting up the system.

The literature has always referred to the first type of group signatures as static group signatures and the last type as dynamic. The early literature would have considered the

second type to be dynamic too [36], however more recent papers [9, 13] have confused the definition of dynamic group signatures. They defined dynamic group signatures to be group signatures that allow users to join a group by engaging in a protocol with an authority and choosing part of their signing key and that definition excludes the second type of group signature.
The idea of enabling the user to engage in a protocol in order to join a group was proposed in order for the user to create part of his private key to prevent key escrow. This thesis divides the group signatures into static and dynamic to be consistent with the most recent literature, even though what we actually mean is issued-key rather than static and owned-key rather than dynamic. It is a consequence of a static scheme that the authority knows the keys of everyone and can therefore impersonate anyone in the group.

**Definition 4.1.2.** Static Group Signatures*: are group signatures that do not have a join protocol, therefore a user can not choose part of his private key and instead an authority dictates the whole signing key to the new members of the group.*

**Definition 4.1.3.** Dynamic Group Signatures*: are group signatures that allows a user to engage in a join protocol with the authority to become a member of the group. The join protocol must allow the user to choose part of his signing key, in secret, and prevents key escrow.*

Group signatures should not be confused with other group oriented signatures such as multi-signature schemes, aggregate signatures, threshold signatures...etc. The following subsection provides a short definition of each and a simple scenario to help visualize the difference.

### 4.1.1 Group Oriented Signatures

In this section we will briefly describe other group oriented signatures in the literature and compare each one of them with group signatures.

- **Multi-Signature Schemes:** A multi-signature scheme allows any subgroup within a group of signers to jointly sign a document. A verifier should be convinced that each member of the subgroup participated in signing. Comparing it to the scenario 4.1.1, a set of employees in Bob's company sign a document and send it to Alice rather than one employee. Alice knows the set of employees so they are not anonymous. Each member has their pair of private and public keys. The employees involved in creating a signature use interactive protocols amongst themselves to create a pair of a message and a signature. Alice can verify that the specific set of employees have signed the document using the public key of each employee in the set. Multi-signatures were introduced in [76] and have been the topic of much work [26, 104, 103, 10, 38].

- **Aggregate Signatures:** An aggregate signature scheme is a digital signature that supports aggregation: Given $n$ signatures on $n$ distinct messages from $n$ distinct users, it is possible to aggregate all these signatures into a single short signature. This single signature (and the $n$ original messages) will convince the verifier that the $n$ users did indeed sign the $n$ original messages. In aggregate signatures, Alice would want, for example, three documents to be signed by employees $A$, $B$ and $C$ in Bob's company. Unlike group signatures, each employee has a pair of a private and a public key. Each employee signs one of the documents. They aggregate all three signatures into one and send it to Alice. To verify, Alice uses all the public keys and messages. The verification should convince Alice that employee $A$ signed the first message, employee $B$ signed the second one and employee $C$ signed the third message. Aggregate signatures is relatively a new concept introduced by Boneh *et al.* in [23]. Note that signers are not anonymous in aggregate signatures.

- **Threshold Signatures:** A threshold signature is a protocol that allows a subset $n$ of $m$ users to generate a signature but disallows any less than $n$ users to produce a valid signature. The verification key and signing key are obtained by running certain protocols between the members. Each player gets a private key and one public key is generated. At least $n$ private keys are needed to calculate a secret key. The secret key is used in signing and corresponds to the public key used in verifying.
  Alice wants a signature on a document from at least $n$ employees. Alice gets a signature that proves that some subgroup of sufficient size signed the document, and the minimal size is a parameter of the scheme and should be known in advance. Employees signing are anonymous.
  Threshold Cryptosystems were proposed by Desmedt and Frankel in [55] and since then threshold signatures have been of an interest to cryptographers [108, 87, 115, 85, 126, 1, 3].

- **Ring Signatures:** A ring signature is similar in concept with group signatures but differs in three key ways; First of all, there is no way to revoke the anonymity of an individual signature (i.e. no one can tell the signer of a message not even the group manager). The second difference is any group of users can be used as a group without additional setup. The third difference is that every user has a public and private key.
  The way ring signatures work is by having a member choose any set of possible signers that includes himself, and he signs a message by using his secret key and the others' public keys, without getting their approval or assistance. It is used as a building block of many cryptosystems.

Ring signatures were invented by Rivest, Shamir, and Tauman who presented their work in [117]. Cryptographers have tried improving the scheme since then [30, 102, 130, 14].

After having described group signatures, and comparing it to other group oriented signatures, we ought to discuss the security notions of group signatures. The following subsection explains security requirements for a group signature scheme.

### 4.1.2 Security Notions of Group Signatures

Group Signature schemes have seen several improvements and diversifications in their features. In this section we shall give informal definitions of such security notions, we shall not go into details on how to prove the schemes to be secure under such notions until later on in this chapter (Section 4.2.2 and Section 4.4.2). The main notions are:

- **Unforgeability:** Only group members are allowed to sign messages on behalf of the group. Otherwise the signature should not verify correctly [41, 9]. Unforgeability is one of the notions that was inherited from digital signatures. It is computationally hard to forge a pair of message and signature.

- **Anonymity:** Anonymity is integrated with the aim of protecting the privacy of signers especially when their identity has no direct effect on the communication of data [41, 9]. Given a signature, it is computationally hard to identify the signer unless you are the group manager.

- **Unlinkability:** Unlinkability is established when it is computationally hard to decide if two different valid signatures were generated by the same signer [6]. Conversely, indistinguishablity occurs if it is computationally infeasible to deduce whether the signatures are from different signers [59]. In group signature schemes, the group manager and he alone should be able to tell if the same member signed two different signatures.

- **Exculpability:** This notion is mostly a measure against adversaries who reside as members of the group. Exculpability is enforced if no group member, not even the manager, can sign on behalf of any other member. This feature reduces the risks of framing attacks and repudiation because no true signature can be mis-attributed to an honest member, even if every member other than the innocent target have colluded [9, 36]. Related notions have also been called Non-Frameability.

- **Traceability:** In case of a dispute, a trusted party should be able to trace a signature to a signer [9, 36]. The group manager is given that privilege. Given a pair of message and signature, the group manager revokes the anonymity of the signer.

- **Coalition-Resistance:** Some members of the group may consider forming a coalition to penetrate the foundations of the otherwise secure scheme. A group signature scheme can be considered coalition resistant if no colluding set can produce signatures which fail to open or cannot be traced back to a member belonging to the set [9].

In Eurocrypt 2003, Bellare, Micciancio and Warinschi presented a paper that provides theoretical foundations for the group signature primitive [9]. They introduced strong definitions for the core requirements of anonymity and traceability. They proved their definitions imply all the rest of the security notions in the literature. What the authors did not do, is consider dynamic group schemes. Their proofs and definitions are suitable for static groups only. Two years later, Bellare, Shi, and Zhang introduced strong definitions of non-frameability, traceability and anonymity that include all other security notions [13]. Their definitions provided the foundation for dynamic group signatures.
In the following sections we will explain static and dynamic signatures. We define their algorithms and security notions.

## 4.2 Static Group Signature Schemes

In static group signatures we have four main class of entities, the key generator, a group manager, a signer and a verifier. In the literature the key generator and the group manager may be combined into one authority. The signer is a member of a group that is capable of signing on behalf of the group. The verifier is anyone who wants to check the validity of the signature. The group manager is an authority who is capable of breaking the anonymity of the signer when given a signature. The group manager plays that role to solve disputes and ensures anonymity is not misused by members of the group. The key generator sets up the system and issues the keys needed.
The key generator starts with setting up the system. It creates one public key, one tracing key and many private keys. It publishes the public key so anyone can use it. It gives every member of the group a unique private key (See Figure 4-1 [1], step 1b) and it sends the tracing key to the group manager (See Figure 4-1, step 1c). Any signer of the group can use the private key he obtained to sign a message and send it to the verifier (See Figure 4-1, step 2). The verifier can check the validity of the signature using the public key she has (See Figure 4-1, step 3). Finally, if the verifier is in doubt of the signature she sends it to the group manager (See Figure 4-1, step 4). The group

[1] Numbering of the image steps are done according to the algorithms. If an algorithm needs to be run before the other then a sequence is used 1,2,..etc. The letters are just to indicate different steps may occur at the same time.

manager breaks the anonymity of the member who signed it (See Figure 4-1, step 5) by using the tracing key and the signature. In the following section we describe the

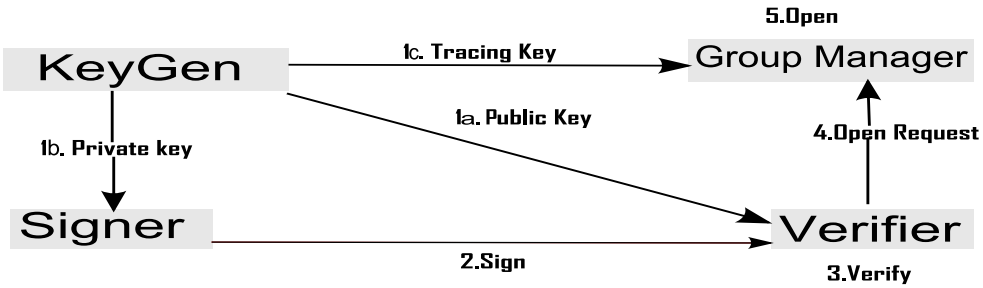


Figure 4-1: Group Signature Scheme

algorithm of the scheme in more details.

### 4.2.1 Definition

In this section we will define a static group signature scheme by describing its algorithms. To distinguish between the algorithms here and in other sections we will use the prefix *SGS* in front of each algorithm. A static group signature scheme is defined with five algorithms: *SGS.Setup*, *SGS.KeyGen*, *SGS.Sign*, *SGS.Verify* and finally *SGS.Open*. The algorithms perform the following operations:

- **SGS.Setup(k)** This algorithm is run by the authority. It takes a security parameter $k$ to generate two types of system parameters: private $S_{pri}$ and public $S_{pub}$.

- **SGS.KeyGen($\mathbf{S_{pub}}, \mathbf{S_{pri}}$)** : This algorithm is run by the authority to generate one public key *gpk* for the group, a tracing key *tk*, and a private key $bsk[i]$ for each member $i$ in the group.

- **SGS.Sign($\mathbf{M}, \mathbf{bsk[i]}, \mathbf{S_{pub}}$)** : Generate a signature $\sigma$ for message $M$ using the user's private key $bsk[i]$.

- **SGS.Verify($\mathbf{M}, \mathbf{gpk}, \sigma, \mathbf{S_{pub}}$)** : Verifies whether $\sigma$ is a valid signature on $M$ using the public key *gpk* and public parameter $S_{pub}$.

- **SGS.Open($\mathbf{M}, \sigma, \mathbf{tk}$)** : Traces a signature $\sigma$ to the signer $i$ using the tracing key *tk*.

We need to point out that the setup and key generation are usually represented as one algorithm. We separate them here since that will help us in explaining some issues in later chapters of this thesis.

### 4.2.2 Security Notions

Previously, in Section 4.1.2, we mentioned how various security notions were studied in the literature and how substantial effort was made to come up with formal definitions that include them all. In this section we go through the main definitions of full traceability and full anonymity as proposed by Bellare, Micciancio and Warinschi in [9]. Provable security (See Section 3) was used in defining these notions.
Two adversarial models are presented. The adversary attacks are modeled by providing it access to certain oracles. We start with explaining the oracles, then a model representing full anonymity attack is defined and a model representing full traceability is defined.
The oracles are listed below:

- **PriKey Oracle:** The Private-Key oracle allows an adversary to obtain a private key for a specific user. The adversary queries the oracle by sending it an index $i$. The oracle responds by sending the private key $bsk[i]$.

- **Signature Oracle:** This oracle allows the adversary to obtain a signature $\sigma$ on a message $M$ from user $i$. The adversary sends the pair $(M, i)$ as its query, implying that it requires a signature of user $i$ on message $M$. The oracle responds with $\sigma$.

- **Open Oracle:** This oracle is meant for tracing a particular signature to a signer. The adversary queries it by sending $(\sigma, M)$, requesting the user $i$ that signed $M$ and generated $\sigma$. The oracle sends back $i$.

**Full Anonymity** We say that a group signature scheme is *anonymous* if no polynomially bounded adversary *Adam* has a non-negligible advantage against the challenger *Charles* in the following Group Signature Anonymity $GSA$ game:

- **GSA.Setup:** *Charles* plays the role of the authority. He runs the algorithms *GS.Setup*, *GS.KeyGen*. He produces the system's parameters $S_{pub}$, and $S_{pri}$. *Charles* also generates $n$ private keys $bsk[i]$, a public key $gpk$ and a tracing key $tk$. The $S_{pub}$ and $gpk$ are sent to *Adam*.

- **GSA.Phase (1):** *Charles* runs three oracles in this phase: PriKey, Signature, and Open. He queries them as described earlier until he decides to start the challenge.

- **GSA.Challenge:** *Adam* asks to be challenged on a message $M$, and two indexes $i_0$,$i_1$. *Charles* responds with a signature $\sigma_b$, where $b \in \{0, 1\}$. The signer can be either $i_0$ or $i_1$.

- **GSA.Phase (2):** Phase (2) is similar to Phase (1). However, *Adam* should not query the Open oracle with what he has as a challenge.

- **GSA.Output:** *Adam* outputs a guess $\bar{b} \in \{0, 1\}$. If $\bar{b} = b$, *Adam* wins the game.

*Adam* is given the ability to corrupt users by querying the PriKey oracle. He can also obtain a number of signatures and trace them to signers using the Signature oracle and Open oracle. Access to such oracles have one condition only and that is in Phase 2 the user should not be able to query the signature $\sigma_b$ in the Open oracle. Such restriction is reasonable because if *Adam* does query it, he is told who the signer of the message is and therefore the challenge of guessing the signer is meaningless.
The idea behind this adversarial model is that even with all the privileges *Adam* has, and even with choosing two users in the challenge phase and the message, he can not distinguish whether the signature $\sigma_b$ is created by $i_0$ or $i_1$. Of course he can have a random guess and the possibility of getting it right is fifty percent but *Adam* should not have any advantage on guessing the right answer.
We represent the advantage of the adversary in winning the attack as $Adv_{GSA}(n, k) = Pr[b = \bar{b}] - 1/2$ where $n$ is the number of users, and $k$ is security parameter used in setting up the system.

**Definition 4.2.1.** Full Anonymity:
*A static group signature scheme is fully anonymous if for any polynomial time adversary, Adam, the advantage of winning the game is negligible. That is $Adv_{GSA}(n, k) < \varepsilon$ where $\varepsilon$ is negligible.*

**Full Traceability** We say that a group signature scheme is *traceable* if no polynomially bounded adversary *Adam* has a non-negligible advantage against the challenger *Charles* in the Group Signature Traceability GST game:

- **GST.Setup:** *Charles* plays the role of the authority. He runs the algorithms *GS.Setup*, and *GS.KeyGen*. He produces the systems $S_{pub}$, and $S_{pri}$. *Charles* also generates $n$ private keys $bsk[i]$, a public key $gpk$ and a tracing key $tk$. $S_{pub}$, $gpk$ and $tk$ are sent to the adversary.

- **GST.Oracles:** *Charles* runs the PriKey oracle and the Signature oracle. He queries them as shown before in the GSA game until he decides to challenge. Unlike the GSA game there is no need to run the Open oracle since *Adam* has the tracing key $tk$ and can run the algorithm himself.

- **GST.Challenge:** *Adam* asks to be challenged on a message $M$. That means *Adam* thinks he can forge a signature on a message $M$. *Adam* sends a forged signature $\sigma$ to *Charles*. *Charles* verifies the signature. If it turns out to be valid, he 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 and 1 is returned otherwise 0 is returned.

In this case *Adam* is given access to the oracles and is given the tracing key $tk$. He can corrupt users by querying the PriKey and ask for signatures by querying the Signature oracle. Giving *Adam* the tracing key $tk$ strengthens his attack capabilities since it proves that even though *Adam* can know every signer of every signature, he should not be able to forge signatures. There is no need for the Open oracle since *Adam* has $tk$.
If the signature he creates in the Challenge is not valid then he obviously lost. If he claims to have created a signature and gives one that he has queried before then again it is not a successful forgery because the challenger gave him that information. Anything else would be a successful forgery and the scheme is said to be untraceable.
We represent the advantage of the adversary in winning the attack as $Adv_{GST}(n,k) = Pr[Exp = 1]$ where $n$ is the number of users, $k$ is security parameter used in setting up the system and $Exp = 1$ refers to the game returning 1.

**Definition 4.2.2.** Full Traceability:
*A static group signature scheme is fully traceable if for any polynomial time adversary, Adam, the advantage of winning the game is negligible. That is $Adv_{GST}(n,k) < \varepsilon$ where $\varepsilon$ is negligible.*

Full traceability and full anonymity cover all notions in section 4.1.2 as proved in [9]. The following is a brief explanation on why:

- Unforgeability is covered through the full traceability game. One can restrict the adversary so that he is not capable of querying private keys in PriKey Oracle and there is no need to give him the tracing key $tk$ and that would represent the unforgeability game model. Therefore we can conclude that if the scheme was forgeable then there must exist an adversary who can break the full traceability game.

- Unlinkability is covered through full anonymity. If the scheme was linkable the adversary of the full anonymity game can query a signature of a user $i$ and later in the challenge stage include $i$ among the users he wants to be challenged upon. There is no condition on the signature oracle in both phase 1 and 2 in the full anonymity game. The adversary does not to guess the signer of the signature he obtains in the challenge because he can just link it thus breaking full anonymity.

- Exculpability is covered through full traceability. If the group manager or the user[2] defeats exculpability then there exist an adversary that can break full traceability. In the former case the adversary simply produces a signature that is a forgery and can not be traced to the person who actually created it. In the latter case the adversary can ask for the private key for user $i$, assuming user $i$ is the one who defeats exculpability, and creates a signature that is not traced to the member $i$ thus breaking full traceability.

- Coalition Resistance is covered through full traceability. If the scheme was not coalition resistant the adversary can query the oracle PriKey to obtain a number of private keys that are enough to create a coalition. The coalition can create a signature that fails to open or does not trace to a member in the group and that also implies breaking full traceability.

In the next section we will give some examples of constructions of different group signatures.

## 4.3 Static Group Signatures-Constructions

In this section we give explicitly two constructions of group signature schemes; The first was proposed by Boneh, Boyen and Shacham (Section 4.3.1) and the other was proposed by Boneh and Shacham (Section 4.3.2). Recall from section 4.1 that we consider the following schemes as static since they do not have a join protocol unlike the schemes in Section 4.4. A better naming would have been "owned-key" group signatures since the authority in both constructions can add users later on and the signing keys are created by the authority without the involvement of the users.

### 4.3.1 Boneh, Boyen, and Shacham Scheme

In Crypto'04 Boneh, Boyen and Shacham proposed a group signature scheme that is secure under the assumption that the Strong Diffie–Hellman problem (Definition 2.2.11) is hard and the Decision Linear problem (Definition 2.2.14) [20] is hard. In this section we will go through the construction of their scheme. The algorithms are described below and the prefix BBS is used to distinguish them from other algorithms in this thesis:

- **BBS.Setup(k)** : This algorithm takes a security parameter $k$ that is used to generate a bilinear map $e : G_1 \times G_2 \rightarrow G_3$ where $G_1$, $G_2$ and $G_3$ are of prime order $p$ ($p$ is determined using the parameter $k$ as mention in section 2.2). Furthermore

[2]The trusted third party that plays the role of the key generator can defeat exculpability since it has all private keys of all users.

there exists a computable isomorphism $\psi$ from $G_2$ to $G_1$. Suppose that the Decision Linear Problem (Definition 2.2.14) is hard to solve in $G_1$ and the q-Strong Diffie–Hellman (see Definition 2.2.11) is hard to solve (as defined by security parameter $k$) in both $G_1$ and $G_2$. Choose a hash function $H : \{0,1\}^* \to \mathbb{Z}_p^*$. Select a generator $g_2 \in G_2$ uniformly at random and set $g_1 = \psi(g_2)$. Select $\gamma \in_R \mathbb{Z}_p^*$. $S_{pri} = \gamma$. $S_{pub} = (e, G_1, G_2, g_1, g_2, H)$.

- **BBS.KeyGen($\mathbf{S_{pri}}, \mathbf{S_{pub}}$)** : Select $h \in_R G_1$ and $\zeta_1, \zeta_2 \in_R \mathbb{Z}_p^*$. Set $u, v \in G_1$ such that $u^{\zeta_1} = v^{\zeta_2} = h$. This can be computed by assigning $u = h^{1/\zeta_1}$ and $v = h^{1/\zeta_2}$. Calculate $w = g_2^{\gamma}$. Generate for each user a private key $bsk[i] = (A_i, x_i)$, where $A_i = g_1^{1/(\gamma+x_i)} \in G_1$ and $x_i \in \mathbb{Z}_p^* \backslash \{-\gamma\}$. Then $gpk = (h, u, v, w)$ and the tracing key is $tk = (\zeta_1, \zeta_2)$.

- **BBS.Sign($\mathbf{S_{pub}}, \mathbf{gpk}, \mathbf{bsk[i]}, \mathbf{M}$)** : The signer chooses $\alpha, \beta, r_\alpha, r_\beta, r_x, r_{\delta_1}, r_{\delta_2} \in_R \mathbb{Z}_p^*$.

  He computes $T_1 = u^\alpha$,$T_2 = v^\beta$, and $T_3 = A_i h^{\alpha+\beta}$. He calculates $\delta_1 = x_i\alpha$ and $\delta_2 = x_i\beta$.

  He then computes
  $R_1 = u^{r_\alpha}$,
  $R_2 = v^{r_\beta}$,
  $R_3 = e(T_3, g_2)^{r_x} e(h, w)^{-r_\alpha - r_\beta} e(h, g_2)^{-r_{\delta_1} - r_{\delta_2}}$,
  $R_4 = T_1^{r_x} u^{-r_{\delta_1}}$,
  $R_5 = T_2^{r_x} v^{-r_{\delta_2}}$.

  The signer calculates $c = H(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5) \in \mathbb{Z}_p^*$.

  Then computes $s_\alpha = r_\alpha + c\alpha$, $s_\beta = r_\beta + c\beta$, $s_x = r_x + cx_i$, $s_{\delta_1} = r_{\delta_1} + c\delta_1$, and $s_{\delta_2} = r_{\delta_2} + c\delta_2$.

  The signature will be $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$.

- **BBS.Verify($\mathbf{gpk}, \mathbf{S_{pub}}, \mathbf{M}, \sigma$)** : The verifier re-derives the following:
  $\bar{R_1} = u^{s_\alpha} T_1^{-c}$,
  $\bar{R_2} = v^{s_\beta} T_2^{-c}$,
  $\bar{R_4} = T_1^{s_x} u^{-s_{\delta_1}}$,
  $\bar{R_5} = T_2^{s_x} v^{-s_{\delta_2}}$,

$\bar{R}_3 = e(T_3, g_2)^{s_x} e(h, w)^{-s_\alpha - s_\beta} e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} (\frac{e(T_3,w)}{e(g_1,g_2)})^c$.

To verify the signature check whether $c = H(M, T_1, T_2, T_3, \bar{R}_1, \bar{R}_2, \bar{R}_3, \bar{R}_4, \bar{R}_5)$ holds. If it is true then accept signature otherwise reject it.

- **BBS.Open**$(\mathbf{M}, \sigma, \mathbf{tk}, \mathbf{S_{pub}}, \mathbf{gpk})$ : To trace a signature $\sigma$ to a signer. The authority first verifies its validity. The second step is to recover $A_i = T_3/(T_1^{\zeta_1} T_2^{\zeta_2})$ and compare it to a list of $A_i$ of the members of the group.

This scheme has been proven secure under full anonymity and full traceability definitions (Section 4.2.2). It is efficient since the size of the signature is constant. The work done in that paper was a fundamental building block of our constructions in later chapters because of its security and efficiency.

### 4.3.2 Boneh and Shacham's Scheme

After the publication of Boneh *et al.*'s work in Crypto'04 (previous section), Boneh and Shacham decided to modify the scheme further so that it enables revocation [25]. The revocation is done by having a list that contains a token representing each revoked user. Using that list a verifier can test whether the signature belongs to a revoked user. Using the revocation algorithm the authority can identify the signer of any valid signature. The authority has all the tokens of members of the group. It creates a fake revocation list containing all members' tokens and runs the revocation algorithm on the fake list. It should trace to a member in that list. In other words the *Open* algorithm is replaced with *Revoke*. When a verifier is running *Revoke* he is checking whether signer is revoked or not. On the other hand, when the authority runs the *Revoke* algorithm on the fake list it is tracing the signature to a signer. The scheme proposed is secure under the assumption that the q-SDH problem (See Section 2.2.11) and the Decision Linear problem (See Section 2.2.14) are hard.
A description of their method follows and a prefix BS is used to distinguish the algorithms in this section from ones in other sections:

- **BS.Setup**$(\mathbf{k})$ : This algorithm takes a security parameter $k$ and generates a bilinear map $e : G_1 \times G_2 \rightarrow G_3$ where $G_1$, $G_2$ are of prime order and $G_1, G_2$ have a computable isomorphism $\psi$ from $G_2$ to $G_1$. Suppose further that the Decision Linear is hard to solve in $G_1$ and the SDH is hard to solve in both $(G_1, G_2)$. Choose the hash functions $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$ and $H_2$ with range $G_2^2$. A generator $g_2 \in G_2$ is randomly chosen and $g_1 = \psi(g_2)$ is computed. Select $\gamma \in_R \mathbb{Z}_p^*$. Let $S_{pub} = (e, G_1, G_2, G_3, g_1, g_2, H_1, H_2)$ and $S_{pri} = \gamma$.

- **BS.KeyGen**$(\mathbf{S_{pri}}, \mathbf{S_{pub}})$ : Set $w = g_2^{\gamma}$. For each user $i$ generate the private key $bsk[i] = (A_i, x_i)$, where $A_i = g_1^{1/(\gamma+x_i)}$. The public key is $gpk = (g_1, g_2, w)$. The revocation token is $A_i$.

- **BS.Sign**$(\mathbf{M}, \sigma, \mathbf{S_{pub}}, \mathbf{S_{pri}})$ : The signer picks $r \in_R \mathbb{Z}_p^*$ and obtains $(\bar{u}, \bar{v}) = H_2(gpk, M, r)$. The signer computes the images in $G_1$ so that $u = \psi(\bar{u})$ and $v = \psi(\bar{v})$.
  The signer selects an exponent $\alpha \in_R \mathbb{Z}_p^*$ then computes $T_1 = u^{\alpha}$ and $T_2 = A_i v^{\alpha}$.
  Let $\delta = x_i \alpha$. Let $r_\alpha$, $r_x$, $r_\delta \in_R \mathbb{Z}_p^*$.
  The signer computes
  $R_1 = u^{r_\alpha}$
  $R_3 = T_1^{r_x}.u^{-r_\delta}$
  $R_2 = e(T_2, g_2)^{r_x} e(v, w)^{-r_\delta} e(v, g_2)^{-r_\delta}$.
  Let $c = H_1(gpk, M, r, T_1, T_2, R_1, R_2, R_3)$.
  Then computes $s_\alpha = r_\alpha + c\alpha$, $s_x = r_x + cx_i$, and $s_\delta = r_\delta + c\delta$.
  Signature is $\sigma = (r, T_1, T_2, c, s_\alpha, s_x, s_\delta)$.

- **BS.Verify**$(\mathbf{M}, \sigma, \mathbf{S_{pub}}, \mathbf{gpk})$ : The verifier starts with recomputing $\bar{u}, \bar{v}, u$, and $v$, and then deriving
  $\bar{R}_1 = u^{s_\alpha}/T_1^c$
  $\bar{R}_3 = T_1^{s_x} u^{-s_\delta}$
  $\bar{R}_2 = e(T_2, g_2)^{s_x} e(v, w)^{-s_\alpha} e(v, g_2)^{-s_\delta} (\frac{e(T_2,w)}{e(g_1,g_2)})^c$.
  To accept a signature the equality $c = H_2(gpk, M, r, T_1, T_2, \bar{R}_1, \bar{R}_2, \bar{R}_3)$ must hold, else reject the signature.

- **BS.Revoke**$(\sigma, [..\mathbf{A}..])$ : For each element $A$ in the revocation list $[..A..]$ check if $e(T_2/A, \bar{u}) = e(T_1, \bar{v})$, if it is true then that user is revoked.

The constructions in Section 4.3.1 and 4.3.2 are strongly related with the main difference being that the bases of the exponents (i.e. $u$ and $v$) are randomized each time using the hash function $H_2$. This randomization makes it possible to remove an element $T_3$ and possible to have an element $T_2$ that can be tested against revoked users in algorithm *Revoke*. The scheme is still fully traceable and fully anonymous.
In this section we have explained static group signatures, their definition, their security notions, and we have given two examples for recent constructions. The following section we go through the same discussion for dynamic group signatures.

## 4.4 Dynamic Group Signature Schemes

In Crypto'97 Camenisch and Stadler came up with the first dynamic group signature scheme (See Section 4.5.1) in their work [36]. They introduced the concept of joining a group at any point of time regardless of the timing of the setup stage of the system. Much research has been carried out afterwards in order to increase dynamic group signatures efficiency.

In dynamic group signatures we have five main entities: the signer, the verifier, the opener, the issuer and the key generator. The opener is responsible for revoking the anonymity of the signer in case of a dispute while the issuer is responsible of giving out private keys to members that join the group.

In this case, the scheme starts with the key generator (KeyGen) setting up the system by generating the required keys and sending them to the right entities (See Step 1a,1b,1c and 1d Figure 4-2). A special key used for tracing signatures is sent securely to the opener $tk$. A key for generating private keys $ik$ is sent to the issuer. A general public key $gpk$ is sent to everyone.

The signer can contribute in creating his private key. The signer and the issuer run a protocol (See Step 2 Figure 4-2) and as a result a registration key $reg_i$ and a private key $bsk[i]$ are created (The protocol is referred to as $Join$ protocol). The registration key is added to a list which is accessible to the opener. That list is useful for the opener since it is used together with the $tk$ to break the anonymity of the signer (See Step 3 Figure 4-2). The private key is used to sign messages (See Step 4 Figure 4-2). The verifier will check the validity of a signature using the general public key (See Step 5 Figure 4-2). If she is in doubt of the signature she can consult the opener (See Step 6 Figure 4-2). The opener should be able to prove that it traced the signature to a valid signer (See Step 7 Figure 4-2).

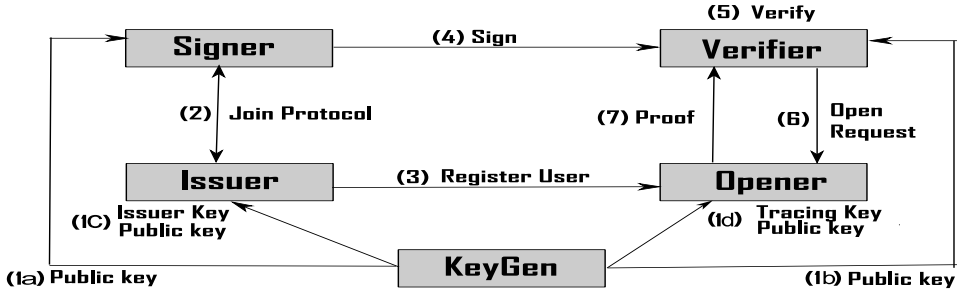


Figure 4-2: Dynamic Group Signature Scheme

In the following section we describe the algorithms of the dynamic group signature in more detail.

### 4.4.1 Definition

In this section we will define a dynamic group signature by defining its algorithms. To differentiate between algorithms in this section we will use the prefix $DGS$ in front of them. A dynamic group signature is defined in six main algorithms $DGS.KeyGen$, $DGS.U.KeyGen$, $DGS.Sign$, $DGS.Verify$, $DGS.Open$, $DGS.Judge$ and a protocol $DGS.Join$ between user $\mathcal{U}_i$ and issuer $Iss$. The algorithms and protocols are described as follows:

- $\mathbf{DGS.KeyGen(k_1)}$ : This algorithm is run by the key generator. It takes as input a security parameter $k_1$ and it outputs three types of keys. The first is the issuer key $isk$ which is given to the issuer as a secret key in order to produce private keys for new members of the group. The second key is the tracing key $tk$ which is a secret key given to the opener and enables it to revoke the anonymity of a signer when needed. The last key is a general public key $gpk$ used mainly in verifying signatures. These keys are used as inputs to other algorithms as shown later.

- $\mathbf{DGS.U.KeyGen(k_2)}$ : This is run by a user who is thinking of joining a group. The input of this algorithm depends on the implementation (it is usually another security parameter $k_2$). The output is a pair of public and private keys $(upk[i], usk[i])$. The public keys can be obtained by anyone.

- $\mathbf{DGS.Join(Iss(isk) : \mathcal{U}_i(usk[i], upk[i]))}$ : This is an interactive protocol between the issuer $Iss$ and the user $\mathcal{U}_i$. The idea behind the protocol is to enable the user to become a member of the group. To engage the protocol the user would be using the pair of keys $(upk[i], usk[i])$ as inputs while the issuer would be using $isk$. By the end of the protocol the user should have a basic secret key $bsk[i]$ that will enable him to sign on behalf of the group and the issuer should be able to calculate a registration key $reg_i$. The latter key will be added to the list of users and that list is accessible by the opener and is used for tracing a signature to a signer.

- $\mathbf{DGS.Sign(bsk[i], M)}$ : This algorithm is run by a member of the group where given a message $M$ and a secret key $bsk[i]$ he can produce a signature $\sigma$.

- $\mathbf{DGS.Verify(\sigma, gpk, M)}$ : This algorithm is run by a verifier who given the public key $gpk$, the message $M$, and the signature $\sigma$, she can decide whether the

signature is valid or not.

- **DGS.Open**$(\mathbf{M}, \sigma, \mathbf{tk}, [..\mathbf{reg_i}..])$ : This is an algorithm that is run by the opener. Given a signature $\sigma$, a message $M$, a list of registration keys $[..reg_i..]$ and a tracing key $tk$, the authority can output an index $i$ of the signer and $\tau$ which is a proof of this claim.

- **DGS.Judge**$(\mathbf{M}, \sigma, \tau)$ : This is an algorithm run by the verifier where given a proof $\tau$, a message $M$ and a signature $\sigma$, the verifier can either accept the claim of the opener that $i$ is the signer or reject it.

Recall that we consider the major difference between the dynamic group signature schemes and the static ones to be the fact that users generate part of their private keys and can join the group at any point. This is the reason behind having the algorithms $Join$ and $U.KeyGen$. The $Judge$ algorithm helps in assuring non-frameability in dynamic group signature schemes. The rest of the algorithms are similar to the ones in static group signatures.
In the following section we explain the core security requirements needed to prove a dynamic group signature secure. According to Bellare, Shi and Zhang in [13] it is enough to prove the scheme secure under the non-frameability, traceability and anonymity's strong definitions they provided. In the following section we explain these definitions.

### 4.4.2 Security Notions

Bellare, Shi, and Zhang used provable security in [13] (See Chapter 3) in order to define the security notions required in a dynamic group signature. We have three adversarial models each representing one of the security notions anonymity, traceability, and non-frameability. All three models use a set of oracles that the adversary can query. Some oracles require the $Join$ protocol to be thought of as many separate stages. Assume that in each stage in the $Join$ protocol one of the two parties takes an incoming message and a current state and returns an outgoing message with an updated state and a decision of accept, reject or continue the protocol. The form of these messages and states are implementation dependent. We can now describe the oracles as follows:

- **AddUser Oracle:** The Add-User Oracle is meant to add an honest user to a certain list. This oracle allows an adversary to add a user $i$ to the list of honest users $HU$. The oracle picks a pair $(usk[i], upk[i])$ on behalf of the user and then it executes the $Join$ protocol by playing the rule of both the issuer and user. At the end of this oracle an entry $reg_i$ is added to the registration database and an entry $usk[i]$ is added to a list of private keys. The public key $upk[i]$ is given to the adversary.

- **CrptUser Oracle:** The Corrupted-User oracle allows the adversary to corrupt a user $i$. It starts with the adversary choosing a $upk[i]$ key and then setting it to be the public key of $i$. At this stage the oracle initializes the issuer's state in anticipation for starting the $Join$ protocol which will take place in the following Snd2Iss oracle.

- **Snd2Iss Oracle:** The Send-to-Issuer oracle assumes a corrupted user performing the $Join$ algorithm with an honest issuer. The adversary provides the oracle with an index $i$ and a message $M$ to be sent to the issuer. The oracle having maintained the issuer state of the previous CrptUser Oracle creates the right response as in the $Join$ protocol, returns the message to the adversary and sets an entry $reg_i$ in the registration database if the protocol completes.

- **Snd2Usr Oracle:** The Send-to-User oracle is used to engage a $Join$ protocol with an issuer who has been corrupted by the adversary. The adversary sends a message $M$ and an index $i$ to be sent to the user. The oracle maintains the state of the user by creating a pair $(upk[i], usk[i])$ the first time $i$ is sent and saving it in a list for later use. The oracle computes a response as in the $Join$ algorithm.

- **USK Oracle:** The adversary can query the User-Secret-Key oracle by sending an index $i$ to expose the user secret keys $(usk[i], bsk[i])$.

- **RReg Oracle:** The Read-Registration oracle allows the adversary to query for a registration key for the user $i$ by sending the index to the oracle. The oracle responds by sending $reg_i$.

- **WReg Oracle:** The Write-Registration oracle allows the adversary to modify or write to the registration table by sending an index $i$ and the key $reg_i$.

- **Signature Oracle:** A signature oracle allows the the adversary to query a signature from user $i$ on message $M$ as long as $i$ is an honest user. The oracle replies with a valid signature.

- **Open Oracle:** The opening oracle allows the user to submit a message $M$, and a signature $\sigma$ to obtain the output of the original open algorithm that uses $tk$.

These oracles will be used in the three adversarial models defined below to give the adversary some reasonable privileges then study whether or not he has an advantage in breaking the scheme. These models are used to define the security notions of full anonymity, full traceability and non-frameability [13].

**Full Anonymity** We say that a dynamic group signature is fully *anonymous* if no polynomially bounded adversary $Adam$ has a non-negligible advantage against challenger $Charles$ in the following Dynamic Group signature Anonymity $DGA$ game:

- **DGA.Setup:** *Charles* will play the role of the key generator. He generates the general public key $gpk$, the tracing key $tk$, and the issuing key $isk$. The issuing key $isk$ is given to *Adam* along with the public key $gpk$.
- **DGA.Phase(1):** In this phase *Adam* queries the oracles Open, Snd2Iss, Snd2Usr, WReg, RReg, USK, Signature and CrptUser and *Charles* responds with an output that corresponds to the ones of the oracles described earlier.
- **DGA.Challenge:** *Adam* decides on a message $M$ and two indexes $(i_0, i_1)$ he would like to be challenged on. *Charles* replies with signature $\sigma_b$ of either member (i.e. $b \in \{0, 1\}$).
- **DGA.Phase(2):** This phase is similar to Phase(1) except that $\sigma_b$ can not be queried in the Open oracle.
- **DGA.Output:** *Adam* outputs a guess $\bar{b} \in \{0, 1\}$. If $\bar{b} = b$, *Adam* wins the game.

This adversarial game model is similar to the one defined in Section 4.2.2 for static group signatures. The main difference between them is the type of oracles *Charles* runs. In this game we show that the opener using the tracing key should be the only entity capable of revoking anonymity. Therefore the choice of oracles takes into consideration that the issuer can be totally corrupted. The issuer in this game is corrupted since *Adam* has $isk$ and access to the oracle Snd2Iss and WReg. This implies *Adam* can play the role of the issuer in *Join* protocol. *Adam* can also obtain honest users' secret keys. He can query for signatures and ask to trace them as in Section 4.2.2 using the Open and USK oracles. He can also corrupt a user and interact with the issuer on their behalf given the fact he has access to Snd2User and CrptUser. Finally, he can read and write to the registration list with WReg. *Adam* definitely can not corrupt the opener since that contradicts with the purpose of the game since the opener should be able to revoke anonymity of members and no one else.
We represent the advantage of the adversary in winning the attack as $Adv_{DGA}(n, k) = Pr[b = \bar{b}] - 1/2$ where $n$ is the number of users, and $k$ is the security parameter used in setting up the system.

**Definition 4.4.1.** Full Anonymity:
*A dynamic group signature scheme is fully anonymous if for any polynomial time adversary, Adam, the advantage of winning the game is negligible. That is $Adv_{DGA}(n, k) < \varepsilon$ where $\varepsilon$ is negligible.*

**Full Traceability** We say a group signature is *traceable* if no polynomially bounded adversary *Adam* has a non-negligible advantage against a challenger *Charles* in the Dynamic Group signature Traceability DGT game:

- **DGT.Setup:** *Charles* will play the role of the key generator. He generates the general public key $gpk$, the tracing key $tk$, and the issuing key $isk$. The tracing key $tk$ is given to *Adam* along with the public key $gpk$.
- **DGT.Oracles:** *Adam* queries the oracles Snd2Iss, RReg, USK, AddUser, Signature and CrptUser and *Charles* responds with an output that corresponds to the ones of the oracles described earlier.
- **DGT.Challenge:** *Adam* asks to be challenged on a message $M$. He sends a forged signature $\sigma$ on that message to *Charles*. *Charles* checks three things before declaring victory to *Adam*. First of all he checks that the signature verifies. If the signature verifies, then *Adam* has not lost the game yet. *Charles* checks whether an honest opener can identify the origin of the signature (i.e. get a valid $i$). If he can not identify it then *Adam* won the game and 1 is returned; Otherwise *Charles* checks whether or not the honest opener produces a correct proof of its claim where the *Judge* algorithm accepts that proof. If it fails then *Adam* has won the game and 1 is returned else he lost and 0 is returned.

The traceability model is similar to the one in Section 4.2.2 in many ways. The opener can be totally corrupted since forging a signature should not rely on whether you can revoke anonymity or not. On the other hand, the issuer can not be corrupted at all since he is the one who generates private keys and we are proving that only he is capable of that. The idea of the game is to ask the adversary to produce a signature where an honest opener can not identify the origin of it or if he does trace it the opener can not produce a correct proof $\tau$ for it. *Adam* is given strong attack capabilities. He has the tracing key $tk$ since the opener can be totally corrupted. He can obtain private keys for users by querying USK and he can create honest members by querying AddUser. He can corrupt users and read the registration list using oracle CrptUser and RReg. He can interact with the issuer on behalf of the corrupted users via Snd2Iss. In other words, *Adam* can play the role of a corrupted user in a *Join* protocol but not the role of a corrupted issuer. *Adam* does not need the Open oracle since he has the tracing key $tk$. We emphasize the fact that *Adam* can not corrupt the issuer and that is a reasonable assumption otherwise he can create fake members that do not trace. For the same reason he can not query WReg.
We represent the advantage of the adversary in winning the attack as $Adv_{DGT}(n,k) = Pr[Exp = 1]$ where $n$ is the number of users, $k$ is the security parameter used in setting up the system and $Exp = 1$ refers to the game returning 1.

**Definition 4.4.2.** Full Traceability:
*A dynamic group signature scheme is fully traceable if for any polynomial time adversary, Adam, the advantage of winning the game is negligible. That is $Adv_{DGT}(n,k) < \varepsilon$ where $\varepsilon$ is negligible.*

**Non-Frameability:** We say that a dynamic group signature is Non-Frameable if no polynomially bounded adversary *Adam* has a non-negligible advantage against challenger *Charles* in the following Dynamic Group Signature Frameability *DGF* game:

- **DGF.Setup:** *Charles* will play the role of the key generator. He generates the general public key *gpk*, the tracing key *tk*, and the issuing key *isk*. All three keys are given to *Adam*.
- **DGF.Oracles:** *Adam* queries the oracles Snd2Usr, WReg, USK, Signature and CrptUser and *Charles* responds as described earlier.
- **DGF.Challenge:** *Adam* asks to be challenged on a message $M$, signature $\sigma$, identity $i$ and a proof $\tau$. *Adam* wins and 1 is returned if all of the following statements are true: (1) $\sigma$ is a valid signature on $M$. (2) $i$ is an honest signer. (3) *Judge* accepts $\tau$ as proof that $i$ created the signature. (4) *Adam* has not queried $(i, M)$ in the signature oracle nor did he query $i$ in USK oracle.
  If any of the four statements fail *Adam* loses and 0 is returned.

*Adam* in this game model is trying to produce a valid proof that an honest user created a verifiable signature even though that signature was not created by that member. *Adam* is given powerful capabilities. He is given both the tracing key *tk* and the issuing key *isk* and access to the Snd2Usr oracle. This implies that the adversary can fully corrupt both issuer and opener. He gets private keys of members by querying USK, and he can corrupt users too by querying CrptUser.
We represent the advantage of the adversary in winning the attack as $Adv_{DGF}(n, k) = Pr[Exp = 1]$ where $n$ is the number of users, $k$ is the security parameter used in setting up the system and $Exp = 1$ refers to the game returning 1.

**Definition 4.4.3.** Non-Frameability:
*A dynamic group signature scheme is Non-Frameable if for any polynomial time adversary, Adam, the advantage of winning the game is negligible. That is* $Adv_{DGF}(n, k) < \varepsilon$ *where* $\varepsilon$ *is negligible.*

## 4.5 Dynamic Group Signatures-Constructions

In this section we give two different constructions of a dynamic group signature. In section 4.5.1 the first ever dynamic group signature, proposed by Camenisch and Stadler in Crypto'97 [36], is explained. The next subsection (Section 4.5.2) will go through the construction of a more recent and more practical group signature that has been proposed by Ateniese *et al.* in the work done in [4].

### 4.5.1 Camenisch and Stadler Scheme

Camenisch and Stadler were the first to come up with the concept of dynamic groups [36]. Even though the authors have suggested using ZKP in order to prove honesty of the opener, their scheme did not include the $Judge$ algorithm, and their suggestion was informal, since it was not included in the security proofs. Another difference in their scheme compared to other dynamic schemes is that the opener, issuer and key generator are one authority. This causes minor differences between the algorithms of Camenisch and Stadler scheme compared to the general definition we had in Section 4.4.2.
The security of their scheme was dependent on the DLP (Definition 2.2.1), and the security of both RSA (Section 2.4.1) and Schnorr (Section 2.4.2) signatures. The algorithms are based on the definitions of signature of knowledge of the double discrete logarithm (Definition 2.5.6) and the e-th root of the double discrete logarithm (Definition 2.5.7). The following is a description of the algorithms of the scheme.

- **CS.KeyGen($\mathbf{k_1}$)** : An RSA public key $(n, e)$ is generated with the security parameter $k_1$ (See Section 2.4.1) and the private key of the RSA represents the issuing key $isk = d$. The authority also picks an element $g \in G$ where $G$ is a cyclic group of order $n$ and the discrete logarithmic problem is hard in $G$. The authority picks an element $a \in \mathbb{Z}_n^*$. It finally picks an upper bound $\lambda$ on the length of the secret keys and a constant $\varepsilon > 1$ (needed for the $SKLOGLOG$ defined in 2.5.6). The public key is $gpk = (n, e, G, g, a, \lambda, \varepsilon)$.

- **CS.U.KeyGen** : A user chooses his secret key $usk[i] \in_R \{0, ..., 2^\lambda - 1\}$ and computes $y = a^{usk[i]}(\mod n)$ and $z = g^y$. The public key is $upk[i] = (y, z)$ where $z$ is computable by any party given $y$.

- **CS.Join(Iss(isk) : $\mathcal{U}_\mathbf{i}$(usk[i], upk[i]))** : The user commits himself to $y$ by signing it and sends $upk[i]$ to the authority. User has to prove that he knows the discrete logarithm of $y$ to base $a$ using signature of knowledge techniques (See Section 2.5.3). The authority if convinced with the proof returns the certificate $v$ to user
  $v \equiv (y + 1)^{1/e}(\mod n)$
  The user is now a member of the group and can sign using $bsk[i] = (usk[i], y, v)$. This triple is infeasible to calculate without the help of the authority. Calculating the e-th root of $y + 1$ is hard because the factorization of $n$ is unknown (This is equivalent to RSA which is believed to be as hard as the factorization problem [51]). If $y + 1$ is computed by calculating $w^e$ for some $w$ it will be infeasible to calculate the discrete logarithm of $w^e - 1$ to the base of $a$.

- **CS.Sign(bsk[i], M)**: To sign a message $M$ a member will need to calculate $\bar{g} = g^r$ for $r \in_R \mathbb{Z}_n^*$. The signer calculates $\bar{z} = \bar{g}^y$. Then he calculates $V_1 = SKLOGLOG[\alpha : \bar{z} = \bar{g}^{a^\alpha}](M)$. Finally, he calculates $V_2 = SKROOTLOG[\beta : \bar{z}\bar{g} = \bar{g}^{\beta^e}](M)$. $\sigma = (\bar{g}, \bar{z}, V_1, V_2)$.
  (See Section 2.5.3 for the definitions of $SKROOTLOG$ and $SKLOGLOG$).

- **CS.Verify($\sigma$, gpk, M)**: The verifier checks the correctness of the signature of knowledge of $V_1, V_2$. If both are correct then signature is valid (Recall the definitions 2.5.6 and 2.5.7). Correctness of $V_1$ implies that $\bar{z}\bar{g} = \bar{g}^{a^\alpha+1}$ for some $\alpha$ the signer knows. $V_2$ proves that the signer knows an e-th root of $(a^\alpha + 1)$. Therefore the correctness of both implies the signer knows a secret key and has a certificate.

- **CS.Open($\sigma$)** : The authority knows $tk = \log_{\bar{g}} \bar{z}$ for all members. It can use that knowledge in tracing the signature $(\bar{g}, \bar{z}, V_1, V_2)$ by checking $\bar{g}^{tk} = \bar{z}$

The scheme was proposed in 1997 that is eight years before the security notions of Bellare, Shi and Zhang were proposed. Therefore the security of the scheme was proven under the different notions mentioned in 4.1.2. It was the first dynamic group signature scheme ever proposed. The following section will give a more recent scheme that is proven secure under full anonymity, full traceability and non-frameability.

### 4.5.2 ACHM Scheme

Ateniese *et al.* came up with a scheme whose security depends on the complexity assumptions sLRSW, EDH, and sSXDH (See Definitions 2.2.18, 2.2.15, and 2.2.16). The scheme is proven fully anonymous, fully traceable and non-frameable. The reader is referred to the main paper for the proofs [4]. The authors of the paper used the Camenisch–Lysyanskaya signature scheme (Section 2.4.3) and the Boneh–Boyen signature scheme (Section 2.4.4) as building blocks for their group signatures. The signature of Camenisch–Lysyanskaya will be notated as $CL\sigma$ while Boneh–Boyen will be $BB\sigma$ leaving the notation $\sigma$ to present the main ACHM group signature.
We should highlight that the signature schemes introduced in Section 2.4 ("Camenisch–Lysyanskaya" and "Boneh–Boyen") were modified in this section to serve as building blocks. Consider the following:

- Camenisch–Lysyanskaya: This signature was changed so that it becomes a blind signature. Recall $CL.Sign(M, sk)$ in Section 2.4.3 takes as an input the message $M$ and the secret key $sk = (s, t)$. It chooses a random $a \in_R G_1$ and outputs $CL\sigma$=($a$,$a^t$,$a^{s+stM}$,$a^M$,$a^{Mt}$). In the ACHM we replace that notation with $CL\sigma = BCL.Sign_{sk}(a^M)$ such that the signing key is $sk$ and the message signed

is hidden since $a^M$ is given as an input rather than $M$ itself.

- Boneh–Boyen: This signature has changed because the public keys $g_1$ and $g_2$ of the scheme in Section 2.4.4 will keep changing and are not part of the public key of the ACHM group signature scheme. Recall $BB.Sign(sk, M)$ in Section 2.4.4 takes a message $M$ and the secret key $sk \in \mathbb{Z}_p^*$ as inputs. A random $r \in_R \mathbb{Z}_p^*$ is selected and the signature $BB\sigma = (g_1^r, g_2^{1/(sk+r)}, g_2^{1/(r+M)})$. However, $BB\sigma = BBB.Sign_{sk}(M, g_1, g_2)$ used in this section implies a signature $BB\sigma$ has been created on message $M$ using private key $sk$ and bases $(g_1, g_2)$.

Ateniese *et al.* have one authority in the scheme which they referred to as group manager $\mathcal{GM}$. The reason for that was the fact that there is no main tracing key for the system. Each member had a tracing element given to the group manager as an outcome for the joining protocol. This made it hard to separate the issuer from the opener. Details of the algorithm are shown below:

- **ACHM.Setup($\mathbf{k_1}$)**: Takes a security parameter $k_1$ to generate the public parameters $S_{pub} = (p, G_1, G_2, G_3, g_1, g_2)$. Run the $CL.Setup$ using the same pairing tuple in $S_{pub}$ to produce $gpk = (S_{pub}, \mathcal{S} = g_2^s, \mathcal{T} = g_2^t)$ which will be public and the private parameters $S_{pri} = (s, t)$.

- **ACHM.U.KeyGen($\mathbf{S_{pub}}$)** : Each user $i$ chooses $usk[i] \in_R \mathbb{Z}_p^*$ and $h \in_R G_1$. He establishes a public key $upk[i] = (h, e(h, g_2)^{usk[i]})$ used for joining a group.

- **ACHM.Join($\mathcal{U}_\mathbf{i}$($\mathbf{gpk}, \mathbf{usk[i]}$), $\mathcal{GM}$($\mathbf{upk[i]}, \mathbf{S_{pri}}$))** This is an interactive protocol between user $\mathcal{U}_i$ and the group manager $\mathcal{GM}$. The user takes inputs his secret key and the public key of the group. Group manager takes inputs the user's public key $upk[i]$ and the private parameters $S_{pri}$. The interaction is as follows:

  1. $\mathcal{U}_i$ gives the public key $upk[i] = (p_1, p_2)$, $g_1^{usk[i]}$ and tracing information $tk = g_2^{usk[i]}$ to the group manager $\mathcal{GM}$. If $e(p_1, tk) \neq p_2$ or $usk[i]$ was in the registration database $D$ then abort. Otherwise, $\mathcal{GM}$ adds $tk$ to $D$.
  2. $\mathcal{U}_i$ sends to the $\mathcal{GM}$ a commitment to his secret key $cmt = PCOM(usk[i])$. $\mathcal{GM}$ considers the public parameters of the $PCOM$ scheme (Definition 2.5.1) to be $g_1$, and $h$. $\mathcal{U}_i$ and $\mathcal{GM}$ run the signature of Camenisch–Lysyanskaya in (Section 2.4.3) on the committed value. $\mathcal{GM}$ picks $r \in_R \mathbb{Z}_p^*$ and sets $f_1 = g_1^r$. $\mathcal{GM}$ computes $(f_2, f_3) = BCL.Sign_{gsk}(f_1^{usk[i]})$. $\mathcal{U}_i$ verifies $CL\sigma$. If it does not verify, he aborts.

3. $\mathcal{U}_i$ uses a zero knowledge proof that the committed value $usk[i]$ in $cmt$ is consistent with the public key $upk[i]$. He also provides extractable zero knowledge proof of $usk[i]$ (See Section 2.5).
4. $\mathcal{GM}$ gives an extractable zero knowledge proof of $S_{pri} = (s, t)$.
5. $\mathcal{U}_i$ calculates $f_4 = f_1^{usk[i]}$ and $f_5 = f_2^{usk[i]}$
6. $\mathcal{U}_i$ has obtained the membership certificate $C_i = (f_1, f_2, f_3, f_4, f_5)$= $(a, a^t, a^{s+st(usk[i])}, a^{usk[i]}, a^{t(usk[i])})$.

- **ACHM.Sign**$(\mathbf{usk[i], C_i, m})$ : $\mathcal{U}_i$ uses his secret key, certificate and a message as inputs for the signature algorithm. He first re-randomizes $C_i$ using $r \in \mathbb{Z}_p^*$ so that he gets $(a_1, ..., a_5) = (f_1^r, ..., f_5^r)$.
  Compute $BB\sigma = BBB.Sign_{usk[i]}(m, a_5, g_2) = (a_6, a_7, a_8)$. $GS\sigma = (a_1, ..., a_8)$ is of the form $(b,\ b^t,\ b^{s+st(usk[i])},\ b^{usk[i]},\ b^{t(usk[i])},\ b^v,\ g_2^{1/(usk[i]+v)},\ g_2^{1/(v+m)})$

- **ACHM.Verify**$(\mathbf{gpk, m, GS}\sigma)$ : To verify $GS\sigma = (a_1, a_2, ..., a_8)$. Check if $(a_1, ..., a_5)$ is a valid $CL\sigma$. Check if $(a_6, a_7, a_8)$ is a valid $BB\sigma$. If both are valid accept $GS\sigma$ otherwise reject it.

- **ACHM.Open**$(\mathbf{gsk, m, GS}\sigma)$ : Check each record in database $D$ whether $e(a_4, g_2) = e(a_1, tk)$. If a match is found then output $i$ as the index of the signer.

- **ACHM.V.Open**$(\mathbf{gsk, m, GS}\sigma, \mathbf{upk[i], tk})$ : $\mathcal{GM}$ checks the signature $GS\sigma$ is valid. $\mathcal{GM}$ traces it to a user $\mathcal{U}_i$. Finally, $\mathcal{GM}$ gives a zero knowledge proof that he knows an $\alpha$ such that $e(p_1, tk) = p_2$ and $e(a_1, tk) = e(a_4, g_2)$.

The scheme proposed was proven secure under full anonymity, full traceability and non-frameability as defined by Bellare, Shi, and Zhang in [13]. It is practical since the size of the key and signature is constant.
The following section explains some features that have been added to group signatures in order to use them for certain applications.

## 4.6 Features Added to Group Signature Schemes

Many features have been added to group signatures to serve different applications. We describe briefly some of the features and research done in the field:

- **Flexible Sized Group Signatures:** As stated earlier adding new members to the system was limited to some predefined number and that problem was first solved by Camenisch and Stadler in [36]. Removing members was also difficult.

In practice, adding and revoking are necessities. Research was done in order to make the size of the number of group members changeable [42, 31, 82, 5, 34, 43, 25, 57, 100, 52]. At first the modifications were simply trying to add the special algorithms like Revoke and Join [42, 31]. Later on, research was more about improving such algorithms. In [5, 34] the authors tried to have revocation algorithms that : (I) Did not rely on time stamps as previous algorithms, or (II) did not require the signature size to be dependent on the number of revoked users. In [43] the proposed scheme made the signing and verifying algorithms independent from the number of current users or revoked ones. In [25] the authors introduced the concept of revocation lists to the operation of group signatures, which meant much less computational and communicational overhead compared to previous schemes. Research in revoking and adding users in the most efficient way continues until today.

- **Identity Based Group Signature:** Identity based cryptography was proposed by Shamir in 1984 in [123]. He suggested replacing public key schemes by identity based ones. In other words, he proposed replacing public keys with identities. In [107] the first identity based group signature was proposed, where the verifying key was replaced with an identity ID that represents the group of signers. Research in this line was to improve such schemes performance-wise [113], property-wise [135] or security-wise [7]. Performance implies improving keys size, signature size, computational and communicational overhead. Property-wise modification can be mixing other properties of group signatures with identity based group signatures [140, 47, 72, 7]. Security improvements might include trying to have stronger security notions and proofs. In [135] they suggested having a group signature where group manager, members and opener are all identity-based.

- **Blind Group Signatures:** Blind signatures were introduced for cases where a signer should not know the content of a message. Even if he sees the pair of message and signature later on, signer should not be able to recollect it. The idea of blind group signatures was first proposed to improve e-cash systems. Group blind signature was proposed in [90]. Cryptographers since then tried to modify it performance-wise, property-wise and security-wise [71, 140]

- **Multi-group and Sub-group Signatures :** In [6] Ateniese *et al.* introduced the concepts multi-groups and sub-groups. Multi-group means a user that belongs to two groups or more can sign on behalf of both with one signature. The word "multi" in this case refers to more than one group but there is one signature and one signer. Sub-group signatures is enabling the verifier to know if a signature belongs to a certain sub-group.

- **Convertible Group Signatures:** In the schemes proposed by Kim *et al.* [83],

in addition to the properties of group signatures, the release of a single bit string by the signer turns all of his group signatures into ordinary digital signatures. Both [89, 88] showed that Kim *et al.*'s work to be insecure under an attack they have defined. Finally [92] proposed a totally new convertible scheme and as far as we know that has not been proved insecure yet.

- **Hierarchical Group Signature:** Hierarchy in group signatures comes from the possibility that a group might have more than one manager and they are arranged in a hierarchy. That is a group manager can open signatures of members who are under his responsibility and his signature can be opened by another group manager higher in hierarchy. It was introduced in [131] and as far as we know no more papers took that research line further.

- **Group Signcryption:** In cryptography, signcryption is a public key cryptosystem that was designed to both digitally sign and encrypt a message simultaneously. Therefore, signcryption implementations are meant to be secure, but also more efficient than traditional system, which was sign and then encrypt. Signcryption was introduced in 1996 by Yuliang Zheng in [141]. In 2003, Kwak–Moon suggested having a group signcryption scheme. Any member of the group can signcrypt a message on behalf of the rest. In [133] cryptanalysis revealed that the Kwak–Moon scheme cannot satisfy any of the properties of unforgeability, coalition-resistance, and traceability. In [86] the authors extended Kwak–Moon in order to reduce the security flaws it had.

Following these examples of how different features were added to group signatures, we should explain our contribution to the set of features. In this thesis we introduce a new generation of group signature schemes. We propose attribute based authentication schemes (Chapter 5 and 6). The idea behind our scheme is to enable the verifier to choose a possible set of attributes he would like the member of the group to have before signing a message. Referring to the scenario 4.1.1 Alice may want the employee to be in a particular department. Group signature is a powerful cryptosystem that we will be using as a building block. The following two chapters have a comprehensive description of our scheme and a general construction that permits us to transfer any group signature scheme to an attribute authentication scheme.

## 4.7 Chapter Summary

Group signatures are digital signatures that allow any member within a group to sign on behalf of the group. Group signatures were introduced by Chaum and Heyst in [41]. It is one of many signature schemes allowing multiple alternative signatories existing in the literature such as multi-signatures, aggregate signatures, threshold signatures and

ring signatures. Group signature is divided into two main types static and dynamic. Formal definitions of their security was proposed in two different papers [9] and [13]. For a static group signature it is enough to prove a scheme secure by proving full anonymity and full traceability. In a dynamic group signature it is enough to prove a scheme secure by proving full anonymity, full traceability and non-frameability. Many features have been added to group signatures and our contribution will be to develop an attribute authentication system that relies on group signatures as a building block.

# Chapter 5

# Attribute Authentication Schemes

We start this chapter by explaining a structure used in the creation of our scheme. The structure is referred to as an attribute tree and is used to verify the trueness of a monotone boolean expression where inputs to that expression represent attributes (e.g. Junior Manager and in Department 'A'). Verification of that expression can not be used as a standalone verification scheme but will be used together with other cryptographic blocks to create our attribute authentication scheme.

In Sections 5.3, 5.4, and 5.5 we explain the three main phases of creating our scheme. In the first phase we modify group signatures in order to include the attribute tree, therefore we name the scheme attribute based group signature. We give the definition of the new scheme, the definition of the security notions required and an example construction. Then we analyze the scheme giving advantages and disadvantages of it.

Disadvantages of the attribute based group signatures led us to redesign a new system (Phase II in Section 5.4). The new design is referred to as the attribute authentication scheme (AAS). Section 5.4 covers definitions of the scheme, and its security notions. It gives an example construction. We finally analyze the attribute authentication scheme discussing its advantages and disadvantages.

Section 5.5 enhances the AAS. The major advance of such a scheme is the dynamic property where users enroll themselves in the group rather than being assigned to become members. In other words, there is a join protocol in the scheme. The DAAS scheme inherits all advantages of the AAS scheme and adds to it extra features. In Section 5.5 we define the new scheme and its security notions. We give an example of a construction and finish the section by analyzing the scheme's security and efficiency.

Section 5.4.6 and 5.5.6 introduce extra protocols to the schemes in Section 5.4 and 5.5. These protocols enhance the security of the scheme. It explains how to exchange information between the different entities in the system.

Finally, we finish the chapter with a conclusion of our result and a summary of the chapter.

## 5.1 Introduction

The aim of this chapter is to come up with an attribute authentication scheme that serves the following scenario:

**Scenario 5.1.1.** *Bob manages a company with many departments and many employees. Employees are divided according to positions, responsibilities, levels, departments ... etc. As in group signatures any member can sign on behalf of the whole company. Nevertheless, the verifier can set a policy on what kind of characteristics he would like the signer to have. For example, Alice as a verifier, wants the employee signing to prove to be a senior manager in department A or a manager (senior/junior) in department B. Similar to group signatures there is a private key for each employee and a public key for the company used for the purpose of signing and verifying. Additional to that each employee obtains from the company a registration key that will help him get attribute related keys from attribute authorities. Attribute authorities can be the different departments of the company. Alice sends her request to the whole company. The request format should make it clear that Alice needs the signer to be: (Senior manager and Department A) or ((Junior Manager and Department B) or (Senior Manager and Department B)). Employees with sufficient attribute-related keys can sign on behalf of the company. Anonymity under a set of attributes is a requirement. In case of a dispute only Bob as the manager can revoke anonymity.*

We will refer to such a scheme as the attribute authentication scheme (AAS) and it should include the following properties:

- No previous knowledge assumption: Previous cryptosystems relied on knowing the public key of the signer before hand or maybe their identity too. In this scheme we cannot assume that Alice has any information about the employees of the company. She may or may not know them from before.

- Unforgeable: This property is inherited from group signatures. It is hard for someone outside the group of possible signers to forge a signature. The signer has to be an employee in the company. In addition to that, proof of possession of attributes is hard to fake. Only employees with sufficient attributes can create valid signatures.

- Anonymity of Identity: Signatures can not be linked to a signer. The verifier or any eavesdropper can not reveal the identity of the signer from the signature. However, anonymity can only be subject to the required attributes. For example,

if there is only one person (say Joe) with a given attribute $X$, and Alice requests the attribute $X$, the signer cannot be anonymous.

- Unlinkable: The verifier cannot distinguish whether two signatures were created by the same signer or not. If signatures are linked, the identity of the signers may get exposed by time causing anonymity to break.
- Traceable: Traceability is required in order to prevent employees from misusing anonymity. Bob and only him as a manager can revoke the anonymity of the signers and trace signatures to members of the group even if they were part of a coalition.
- Anonymity of Attributes: The scheme should not enclose the attributes with the signature, it should just provide a proof of possession of sufficient set of attributes. For instance, in Alice's policy she does not need to know whether the employee signing is actually in department A or B. All she needs to know is that her policy is satisfied and it does not matter how.
- Coalition Resistant: If two valid employees have enough attributes, jointly, that satisfy the policy they should not be able to create a valid group signature as if they were one person. For instance, an employee in department A but not a senior manager and a senior manager in department C should not be able to create an acceptable signature to Alice.
- Separability: The different departments should be capable of providing attributes independently of the rest of the company.

In this chapter we describe the three main phases we went through while designing the cryptosystem. Phase one covers the properties above except for anonymity of attributes. Phase two improved phase one to include anonymity of attributes. The level of separability improved in every stage. The details on how it improved is described throughout this chapter.
The methodology explaining how policies have been created is out of the scope of the thesis. It is assumed that the verifier decides on a policy that includes a description of attributes he requires from the signer and represents it using the attribute tree structure explained in 5.2.
In the following section we shall discuss existing attribute oriented authentication schemes. We will point out their drawbacks in order to convince the reader that a new cryptographic scheme is needed for implementing scenario 5.1.1.

### 5.1.1 Attribute Oriented Authentication

Our proposed authentication scheme was not the first research to be done on authenticating with attributes. Cryptographers realized the importance of moving towards

attribute oriented authentication since early 80's. However, the solutions provided varied according to the applications given and none of the existing solutions cover all properties we require. There are two terms that will be used throughout this thesis: Attributes and Credentials Digital credentials are the computer analog to paper credentials such as Student ID, Driving license, passports etc. Attributes are descriptions of users in a digital form such as age, nationality, position etc. An attribute can be presented in more than one credential. For example, nationality can be proven with a passport or civilian ID. The terms have been used equally in literature depending on the application and the cryptosystem. We use the term "Attribute Oriented Authentication" to include authentication schemes that are similar to the proposed attribute authentication scheme in this thesis. Even though some of cryptosystems below use the term credential rather than attribute.
In this section we shall introduce some examples of attribute oriented schemes and analyze their properties.

- **Identity Based Cryptography:** Identity based cryptography is a special form of public key cryptography. Here, the public key has been replaced with the identity of either the signer or the decryptor. The idea was first proposed in Crypto'84 when Shamir presented his work in [123]. The following is an explanation on how identity based cryptography works:

  - Identity Based Signatures: There are three entities; a signer, a verifier and a key generator. The signer obtains a private key that corresponds to his identity from a key generator. He signs a message with that key. The verifier uses the identity of the signer to check validity of the signature.
  - Identity Based Encryption: There are three entities; an encryptor, decryptor, and a key generator. The decryptor sends his identity to the key generator and obtains a private key that corresponds to it. The encryptor will use that identity to encrypt the message. The ciphertext is decrypted with the private key.

  Much research has been done on identity based cryptography [21, 73, 48, 46] since Shamir's first proposal.
  If we consider an identity to be an attribute then this is the first step towards having an attribute based authentication scheme. In the work done in hidden credentials [74, 28] and the work in cryptographic workflow [2] the authors suggested a credential based cryptographic scheme. The general idea is to create an encryption scheme such that the receiver of a ciphertext can only decrypt if he satisfies a particular policy chosen by the sender at the time of encryption. The schemes were based on identity based encryption. These techniques can be also used for authentication purposes. Identities in these papers were created

using the ID of the person encrypting concatenated with attribute templates (i.e. $ID_{new}$=$ID_{org}||template_1||...||template_n$ where $ID_{new}$ is given to the key generator and he produces a private key for it, $ID_{org}$ is the actual identity of the decryptor, and finally $template_j$ is a publicly known template for attribute $j$). The following is a toy example: Alice encrypts a message to Bob with a certain policy. Bob decrypts the message if he satisfies the policy and sends it back encrypted with Alice's public key. Alice will be able to authenticate Bob by knowing he obtains sufficient credentials since he would have not been able to decrypt otherwise.

Another approach to achieve authentication with identity based signatures, rather than identity based encryption, is by concatenating identities to some attribute templates that are publicly known as shown earlier. The signer requests private keys that correspond to such concatenation. The verifier will use such concatenation to create a verification key. We can then implement the scenario in 5.1.1.Two different signers can not join their attributes together to sign a message since that requires contacting the key generator for a new private key. The signer is forced to attach his identity to the signature which leads to breaking the anonymity and unlinkability of the scheme. There is no way to maintain the anonymity of attributes of the signer since the attribute templates are disclosed with the signature. Finally, we are assuming some previous knowledge given that the identity of the signer is used by the verifier.

- **Anonymous Credentials:** In anonymous credential systems (or pseudonym systems) there are users and organizations. Organizations know the users by their pseudonyms and they issue credentials to these pseudonyms. Users can prove possession of a credential even to organizations that know them with a different pseudonym. Such proofs have to guarantee anonymity of the user and it has to be unlinkable. Credentials should also be coalition resistant and unforgeable. Such a system was first introduced by Chaum in [40]. In 2001, Camenisch and Lysyanskaya proposed the first practical solution for such a scheme [32], in the sense that their solution enabled proving possession of credentials multiple times without involving the organization each time and proofs are still unlinkable. They offer for the first time anonymity revocation under certain conditions and terms. Finally, they offer organization separability.

  In that same year, Verheul also proposed a scheme that improves practicality of Chaum's system. He named the new scheme a "Self-Blindable Credential Certificates" [132]. The system binds a user's pseudonym to their public key corresponding to the private key to which the user possesses. Verheul added to the system a third entity so that now it includes a user, an organization (referred to

in his paper as the trust providers), and service provider. Self blindable credentials are issued by an organization. A credential is an organization's testimonial about the user. Service providers are parties that rely on such statements.

In anonymous credential systems there can be two types of credentials: multiple-show or single-show. The usage of each type depends on whether the credential needs to be used more than once or not. For example, a credential that represents a driving license should be of type multiple-show while a credential used to represent a medical prescription is a single-show.

A lot of research has been done in this topic mainly to improve security and practicality of the first scheme of Chaum's. As regards security, forgeability, anonymity and unlinkability were the main concerns. Other desirable properties are restricting credential sharing, anonymity revocation, credential revocation ... There are two ways to discourage users from sharing credentials. The first is referred to as "PKI-assured non-transferability" [58, 67, 91] where sharing a credential implies sharing a valuable secret key from outside the system (e.g. credit card detail). The second way of restricting sharing credentials was having all-or-nothing policy where sharing one credential reveals all others [32].

Revocation is another powerful modification to the scheme. It can refer to either attribute revocation or user revocation [33, 34].

Breaking anonymity was also studied. Anonymity will facilitate misbehavior of users; so in case of a quarrel some authority should be able to revoke that anonymity [32].

Using anonymous credential systems is one suggestion to rectify the predicament of scenario 5.1.1. Nonetheless, some core properties are missing. For example, schemes introduced so far are not coalition resistant even though some existing proposals have provided techniques to discourage credential sharing. Anonymous credential systems do not provide attribute anonymity because the attributes used are disclosed with the credentials. On the bright side the existing systems guarantee anonymity and unlinkability. The traceability of the signature is decided by the signer therefore it is conditional. Anonymous credentials can be used to implement our scheme but will not achieve all properties we need.

- **Trust Negotiation** Trust negotiation is the process of exchanging digital credentials between strangers in order to built trust gradually. In old systems authentication simply relied on previous knowledge about the other party such as having to use passwords, public keys, identities ... Nowadays, in open systems where two strangers need to communicate, property based authentication is needed and using trust negotiation is ideal.

  The idea of trust negotiation was first proposed in [137]. The authors introduced trust negotiation protocols that describe the exact method of exchanging the

credentials. The protocols enabled resource owners and users to establish trust in one another through cautious, iterative, bilateral disclosure of credentials. A sequence of exchanges take place starting from the least significant credentials. As the level of exchanges increase, trust should increase and disclosure of more sensitive credentials take place until resource is granted and all policies are met. Lots of trust negotiation strategies and protocols are suggested in literature. The challenge researchers had to face is finding the balance point between privacy and completeness. The aim of the protocols is to minimize the amount of credentials being exchanged without sacrificing the success of the negotiation. The negotiation should allow access mediators or resource requesters to terminate once they lose trust in one another. Lots of times the fear of information exposure causes the negotiation to end. Improving the strategies to accomplish such balance was the focus of research [138, 139, 136].

Trust negotiation has been used for authentication and authorization purposes especially in open system environments. Strangers have to interact with each other in such environments where no third party existed, at least not online. The scheme does not serve our scenario 5.1.1 since it slowly reveals the privacy of the user, and breaks both the anonymity and unlinkability of the scheme.

- **Mesh Signatures** The idea of mesh signatures is relatively new. In EuroCrypt'07 Boyen presented a paper that proposed the first mesh signature ever [27]. The idea can be considered as an extension to ring signatures (See Section 4.1.1), but with added modularity and a much richer language for expressing signer ambiguity. Intuitively, mesh signatures (as in ring signatures) need to be anonymous and unforgeable.

  A mesh signature is a non-interactive witness-indistinguishable proof that some monotone boolean expression is true, where each input of that expression is labeled with a key and message pair and is true only if the mesh signer is in possession of a valid signature on the stated message under the stated key.

  Boyen did not present his scheme as an attribute oriented authentication. The purpose of the scheme was to enable secret leaking with "unwitting and unwilling participants" as mentioned in the title of his paper. Mesh signature seemed as a promising scheme for our purposes because of the boolean expression proof. If the keys and messages in the expression presented attributes then the proof of the expression is equivalent to proof of possession of attributes. In Maji *et al.*'s paper [93], an attribute based authentication scheme was built extending our work in [81, 79] and using mesh signatures.

  The drawback of such an idea is the fact that mesh signatures are a continuation of ring signatures therefore traceability is not an option. Each user would also have a pair of public and private keys. The signer needs to know and use public

keys of other members of the group too.

The shortcomings of existing attribute oriented schemes were the inspiration of the development of our new scheme. This chapter explains in details the scheme developed.

## 5.2 Attribute Tree

An attribute tree is the structure used to present the verifier's request. It was first proposed in [69] to implement an attribute based encryption scheme where the authors used a tree structure, bilinear maps and Lagrange interpolation to build a policy for decryption. In such a tree 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 5-1 that demonstrates the scenario 5.1.1. $\Gamma$ will be used as a description to

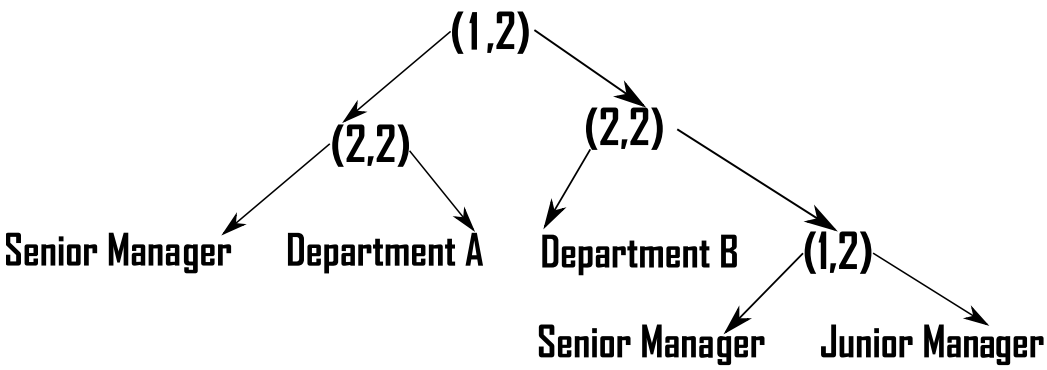


Figure 5-1: Attribute Tree

our attribute tree. For example, to represent the tree in Figure 5-1, $\Gamma = \{(1,2),(2,2),$ $(2,2)$, Senior Manager, Dept. $A$, Dept. $B$, $(1,2)$, Senior Manager, Junior Manager$\}$, where $(m,n)$ represents a threshold gate $m$ of $n$. The description $\Gamma$ is representing the tree in a Top-Down-Left-Right manner. Let $\kappa$ be the number of leaves in $\Gamma$. For simplicity, in the example above we have chosen $m$ of $n$ to either represent an "or" relation where $m = 1$ and $n = 2$ or it represents an "and" relation where $m = 2$ and $n = 2$.

$\Upsilon_i$ is the set describing all attributes owned by a member. For example, if Smith is a senior manager and works in Department $A$, $\Upsilon_{Smith} = \{$Senior, Dept. $A$ $\}$. The size of $\Upsilon_i$ is represented by $\mu$.

$\Im_i$ is a subset of $\Upsilon_i$. It is sufficient to prove possession of some subset $\Im_i$ of the attributes rather than all. For example, if Pat requires the employee signing to be a

senior manager only, then Smith can use $\Im_{Smith} = \{\text{Senior}\}$. The size of $\Im_i$ is $\tau$. Consider the following scenario:

**Scenario 5.2.1.** *Victor wants Pat to prove possession of a set of attributes. He builds the tree $\Gamma$ and sends it to Pat. Pat checks her $\Upsilon_i$ which will be presented as a set of private keys. She will use whatever key(s) she requires to prove to Victor that she indeed satisfies the tree he requested. In other words she decides on a set $\Im_i$. She creates a proof $\mathcal{P}$ and sends it to Victor. Victor can verify $\mathcal{P}$.*

To implement such a scenario we will have three main algorithms as described below:

- **Setup**$(k)$ : This algorithm is run by a trusted third party. It takes as an input a security parameter $k$ and generates a bilinear map (Definition 2.2.9) $e : G_1 \times G_2 \rightarrow G_3$ defined on groups of prime order $p$. A generator $w \in G_2$ is chosen and is public to everyone. Each member of the system will have a private key $A_i \in G_1$ where $i$ is an index referring to the user. The exact way of choosing $A_i$ and $w$ will be according to the AAS scheme used and we will explain that in detail in later sections. For now we will assume the key $A_i$ is chosen uniquely for each user.
  The trusted third party has a master key $t_j \in \mathbb{Z}_p^*$ for each attribute $j$. That key will be used in creating what we refer to as attribute public key $bpk_j$ and attribute private key $T_{i,j}$. A set of attribute public keys is notated as $\overline{B}$. Every public key of an attribute $j$ equals $bpk_j = w^{t_j}$ and every private key of the attribute $j$ of user $i$ equals $T_{i,j} = A_i^{1/t_j}$.

- **TCreate**$(\mathbf{\Gamma}, \alpha, \overline{\mathbf{B}})$ : Victor starts with deciding the structure of the tree $\Gamma$ as shown in Figure 5-1. Accordingly he decides $\overline{B}$. The structure of the tree $\Gamma$ includes the attributes needed, the threshold gates of the nodes, and their indexes while the set $\overline{B}$ is the attribute public key needed to construct such a tree. He then picks a secret $\alpha \in \mathbb{Z}_p^*$. The choice of $\alpha$ is dependent on the AAS scheme (details will be explained in later sections).
  Victor randomly assigns all nodes (other than the root) an index and adds the indexes to $\Gamma$. We use the notation $Index(Node)$ to represent the index of a node where $Index(Node) \in \mathbb{Z}_p^*$. Starting from the root he 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 the number required by the threshold gate $q_{node}$ and the indexes, one can calculate $\alpha$ with Lagrange interpolation (see Definition 2.1.13). However, $\alpha$ will be Victor's secret that will help him verify the tree is satisfied. Victor will calculate $D_j = bpk_j^{q_j(0)}$ for all leaves,

he will let $\overline{D} = D_1, ..., D_\kappa$ and he will send $\langle \Gamma, \overline{D} \rangle$ to Pat.

- **TVerify**$(\overline{\mathbf{D}}, \Im_{\mathbf{i}}, \bar{\mathbf{T}})$ : Pat decides from $\Gamma$ and $\Upsilon_i$ (she owns) which private keys to use (i.e. she decides $\Im_i$). $\bar{T}$ is elements that are calculated using the set of private keys in $\Im_i$ and $\overline{D} = D_1, ..., D_\kappa$ produced in the previous algorithm. For now we will assume $\bar{T} = \{CT_1, CT_2, ..., CT_\tau\}$ where $CT_j = T_{i,j}^{\beta}$, and $\beta \in_R \mathbb{Z}_p^*$. Given Pat has enough attributes she can run a recursive function $Sign_{Node}$ as follows; If the node is a leaf in the tree the algorithm returns the following:

$$Sign_{Node}(leaf) = \begin{cases} \text{If } (j \in \Im_i)\text{; return } e(CT_j, D_j) \\ \text{Otherwise return } \perp \end{cases}$$

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

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

and compute

$$F_\rho = \prod_{z \in \hat{S}_\rho} F_z^{q_z(0)\Delta_{\hat{S}_\rho, index(z)}} = \prod_{z \in \hat{S}_\rho} e(T_{i,j}^{\beta}, D_j)^{q_z(0).\Delta_{\hat{S}_\rho, index(z)}} = \prod_{z \in \hat{S}_\rho} e(A_i^{\beta}, w)^{q_z(0).\Delta_{\hat{S}_\rho, index(z)}}$$

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

Calculate $F_{root}$ then if the tree is satisfied $F_{root} = e(A_i^{\beta}, w)^{\alpha}$(see Example 5.2.2).

---

**Example 5.2.2.** *This is an example of how the tree actually works. The indexes are random and so are the polynomials and they are chosen over $\mathbb{Z}_p^*$. The number on top of each node in the tree Figure 5-2 is the index of that node $Index(Node)$. Assume*

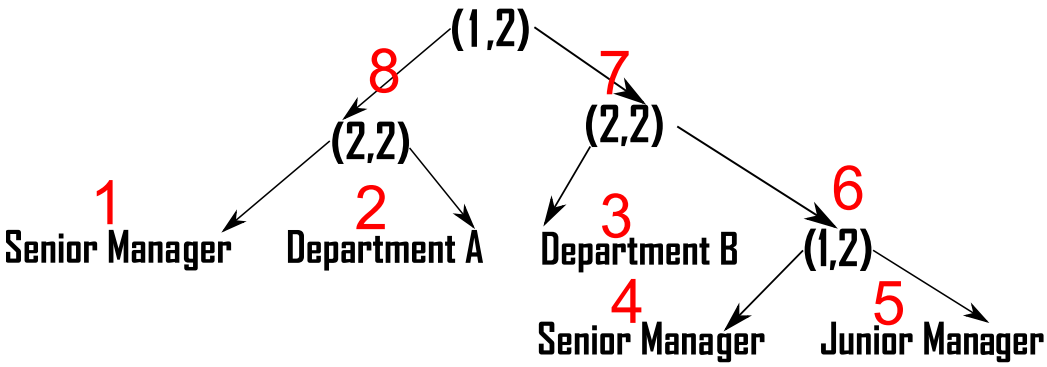


Figure 5-2: Attribute Tree with indexing

*$\alpha = 9$ and further assume that the polynomials are set to the following:*

$q_{root}(0) = 9$ $\quad$ $q_{root}(x) = 9$

$q_8(0) = q_{root}(8) = 9$ $\quad$ $q_7(0) = q_{root}(7) = 9$

$q_8(x) = 2x + 9$ $\quad$ $q_7(x) = 3x + 9$

$q_1(0) = q_8(1) = 11$ $\quad$ $q_2(0) = q_8(2) = 13$

$q_3(0) = q_7(3) = 18$ $\quad$ $q_6(0) = q_7(6) = 27$

$q_6(x) = 27$ $\quad$ $q_4(0) = q_6(4) = 27$

$q_5(0) = q_6(5) = 27$

*Let Pat be a senior manager in department A. She will be using $D_1 = bpk_1 = w^{t_1}$ and $D_2 = bpk_2 = w^{t_2}$ in the $TVerify$ algorithm. She has the private keys $T_{i,1} = A_i^{1/t_1}$ and $T_{i,2} = A_i^{1/t_2}$. She is interested in the left subtree therefore indexes 1 and 2. Note that the degree of the root is 1 therefore we can stop in index 8. Pat will calculate the product of :*

$e(T_{i,1}, D_1)^{\frac{-2}{1-2}} e(T_{i,2}, D_2)^{\frac{-1}{2-1}}$

$= e(A_i, w)^{q_1(0)(\frac{-2}{(1-2)})} e(A_i, w)^{q_2(0)(\frac{-1}{2-1})}$

$= e(A_i, w)^{11(\frac{-2}{(1-2)})} e(A_i, w)^{13(\frac{-1}{2-1})}$

$= e(A_i, w)^{22} e(A_i, w)^{-13}$

$= e(A_i, w)^{9}$

$= e(A_i, w)^{\alpha}$

*Pat will raise the whole element to the power of a $\beta$ which she chooses randomly.*

---

Pat can create the proof $\mathcal{P}$ that consists of two parts $F_{root}$ and a trapdoor $td$ (i.e. $\mathcal{P} = \langle F_{root}, td \rangle$ is sent to Victor). The trapdoor is a way to reveal that $F_{root}$ contains $\alpha$ implying satisfaction of the tree. Meanwhile the trapdoor has to maintain the randomization in $F_{root}$. The trapdoor will be calculated using the public key of the attribute authentication scheme. For instance, in this section the public key is $w$ therefore the trapdoor is $td = w^{\beta}$ where $\beta \in_R \mathbb{Z}_p^*$ is the randomization factor we want to reserve. Later sections in this chapter will explain in more details how the registration key $A_i$, $w$ and $td$ are exactly calculated.
Given that Victor knows $\mathcal{P} = \langle F_{root}, td \rangle$, $w$, and $\alpha$ he can verify that Pat satisfies the tree by verifying $F_{root}^{1/\alpha} = e(A_i, td)$. Notice forging the pair $\langle F_{root}, td \rangle$ is as hard as the BDH problem (See Definition 2.2.9).
To prove that consider the following game which we will refer to as Forgery of Attribute Possession (FAP). In this game we have a *Charles* as a challenger and *Adam* as an adversary. The game runs as follows:

- Setup: Charles starts by creating the public key $w$ and private keys $A_i$ of all users. He sends the information to *Adam*. *Charles* then chooses the attribute master keys $t_0, ...., t_m$ for the whole universe of attributes.
- Phase 1: In this phase *Adam* can query any of the three oracles below:
  - AttPriKey Oracle: *Adam* sends a users private key base $A_i$ together with an index for an attribute $j$ and *Charles* responds back with $T_{i,j}$.
  - TVfy Oracle: *Adam* runs TCreate on $\Gamma$, $\alpha$, and $\overline{B}$ of his choice. He sends the outcome $\overline{D}$ together with index of user $i$ to *Charles* and asks for a verification. *Charles* should reply with a valid $\mathcal{P}$. Since *Charles* has all master keys of all attributes such a proof is straight forward.
  - AttMasKey Oracle: *Adam* sends an attribute $j$ and gets its master key $t_j$.
- Challenge: *Adam* asks to be challenged on a certain tree $\Gamma_1$, user $l$ and attribute $z$ where $l$ has not been queried in AttPriKey for attribute $z$ and $z$ has not been queried in the AttMasKey Oracle. *Charles* sends the $\overline{D}$ of a new tree $\Gamma_2$ with a root of threshold (2 out of 2) where the first child is $\Gamma_1$ and the second is with base $bpk_z$.
- Phase 2: This phase is similar to phase 1 as long as the challenged user $l$ is not queried in AttPriKey for attribute $z$ and $z$ is not queried in AttMasKey.
- Output: *Adam* should respond with a valid $\mathcal{P} = (F_{root}, td)$. If $\mathcal{P}$ is verifiable then *Adam* wins else he loses the game.

The game assumes the issuer of the private key bases is corrupted since all $A_i$ are given to *Adam*. It gives *Adam* the power of knowing attribute private keys of different

attributes and for different users by querying the oracle AttPriKey. It also gives him the power to generate private keys since he can query their master keys in AttMasKey oracle. The information that would be hidden from *Adam* is the master key $t_z$ and the private attribute key $T_{l,z}$ where $l$ is the user and $z$ is the attribute in which *Adam* requested to be challenged upon. It is natural to hide these keys for knowing them contradicts the purpose of the game.

In the literature such an attribute tree was used in attribute based encryption only. The proof was merged with the CPA game (Section 3.2). The general idea of the proofs was by assuming there exist an adversary *Adam* that breaks the CPA and concluding that a simulator running *Adam* can solve the DBDH problem. When *Adam* requests the challenge he sends two messages $m_1$ and $m_2$ to the simulator who chooses one randomly and simulates a ciphertext with the elements of the DBDH. Recall the elements are ($A$, $B$, $C$, $Z$)=$(g^a, g^b, g^c, e(g,g)^z)$ where $z$ is either random or $z = abc$ (See Definition 2.2.10). In case $z = abc$ then the simulation of the ciphertext is equivalent to a normal ciphertext. Therefore if *Adam* guesses the message being encrypted correctly then $z$ most probably is not random. If he guesses it incorrectly then $z$ must be random. Consequently the advantage of solving the DBDH for the simulator is non-negligible.

Using the same technique in signatures is not easy because the winning condition is coming up with a proof $\mathcal{P}$ rather than giving out a boolean guess.

To prove the game is hard to win we use a non-traditional approach as follows:

Let the root polynomial be $q_r(x)$ and has two child nodes. The subtree $\Gamma_1$ has a root with the polynomial $q_1(x)$ which represents the first child of $\Gamma_2$ and is indexed as $x_1$. The other child is holding attribute $t_z$, has polynomial $q_2(x)$ and is indexed as $x_2$. Further let that $q_r(0) = \alpha$, $q_1(0) = q_r(x_1) = y_1$ and $q_2(0) = q_r(x_2) = y_2$. The root polynomial is of degree 1 since the threshold gate is 2. This implies that *Adam* knows that Lagrange is applied therefore the following formula must hold:

$e(A_i, w)^\alpha = (e(A_i, w)^{y_2.x_2} e(A, w)^{-y_1.x_2})^{1/(x_1-x_2)}$

*Adam* also knows the value of elements ($x_1$, $x_2$, $e(A_i, w)^{y_2}$, $A_i$, $w$, $w^{y_1 t_z}$, $w^{t_z}$ ).

*Adam* does not know $e(A_i, w)^\alpha$, $e(A_i, w)^{y_1}$, $\alpha$, $y_1$, and $t_z$.

Note that values $x_1$, $x_2$, $y_1$, $y_2$, and sometimes $\alpha$ itself change each round a $\overline{D}$ is created.

*Adam* might have also obtained a set of $T_{i,k}$ keys for users he queried in AttPriKey Oracle but definitely not the one in the challenge (i.e. $T_{i,k} = A_k^{1/t_z}$ where $k \neq l$). He can also calculate $e(A_k, w)^\alpha$ for all the $T_{i,k}$ he possess.

*Adam* also would have obtained a set of $t_j$ where $j \neq z$.

Table 5.2 summarizes the maximum set of data *Adam* can obtain from the game model.

To win the game *Adam* needs to calculate a pair $\mathcal{P} = (F_{root}, td)$ that is valid for user $l$ (i.e. $F_{root} = e(A_l^\beta, w)^\alpha$ and $td = w^\beta$). The challenge is actually in calculating $e(A_l, w)^\alpha$ rather than $\mathcal{P}$. The reason behind that is the fact that $w$ is known for *Adam* and $\beta$ is of his choice so calculating $e(A_l, w)^\alpha$ implies calculating $\mathcal{P}$.

Table 5.1: Information Obtained by *Adam*

| Information | Source |
|---|---|
| List of $A_i$ | Setup |
| $w$ | Setup |
| 2D List of $T_{i,j}$ where $j \neq z$ and $i \neq l$ | AttPriKey Oracle |
| List of $\mathcal{P} = (F_{root}, td)$ | TVfy Oracle |
| List of $t_j$ where $j \neq z$ | AttMasKey Oracle |
| $\overline{D} = D_1, ..., D_\kappa$ | Challenge |
| $(x_1,\ x_2,\ e(A_l, w)^{y_2},\ w^{y_1 t_z},\ w^{t_z}\ )$ | Challenge and Lagrange Interpolation |
| List of $e(A_k, w)^\alpha$ | $T_{i,k}$ from AttPriKey and $\overline{D}$ of the Challenge |

As mentioned earlier *Adam* knows from Lagrange interpolation the following:
$e(A_l, w)^\alpha = (e(A_l, w)^{y_2.x_2} e(A_l, w)^{-y_1.x_2})^{1/(x_1 - x_2)}$
To win the challenge *Adam* either calculates the right hand side of the equation or the left hand side.
From what *Adam* (See Table 5.2) has $\alpha$ is not enclosed implicitly or explicitly except in the bilinear maps he calculates (i.e. List of $e(A_k, w)^\alpha$ ) and deriving it from them is as hard as solving the BDH.
Alternatively, to calculate the right hand side of the equation, *Adam* can try to calculate either $w^{y_1}$ or $e(A_l, w)^{y_1}$ which will help him calculate $e(A_l, w)^\alpha$ using Lagrange. From Table 5.2 *Adam* knows $w^{y_1 t_z}$ and $w^{t_z}$. Calculating $w^{y_1}$ from that is impossible because $y_1$ is totally bound with $t_z$. Further more, calculating $e(A_l, w)^{y_1}$ from what *Adam* has requires breaking the DLP.

## 5.3 Phase I: Attribute Based Group Signature

The first phase for creating attribute based authentication systems was to modify group signatures to include attribute verifications using the tree structure in Section 5.2.
The reason behind using group signatures as the foundation of our work was the security notions studied in literature for such cryptographic schemes (See Section 4.1.2 for details). A lot of the security requirements we need in our scheme overlap with the one's proposed for group signatures. Examples will be anonymity, unlinkability, unforgeability, ... This made using group signatures for our scheme sound promising.
Similar to group signatures we have three main entities: the signer, the verifier and the authority. However, there is more than one public key each representing the group as a whole. The difference between the public keys is the corresponding set of attributes the signer needs to possess in order to consider the signature as valid. A database is added so everyone has access to such keys. The verifiers can retrieve keys from the database (Step 3 in Figure 5-3) since it is publicly known. If the key does not exist he

can request it to be added to the database (Step 1 and 2 in Figure 5-3). Each member in the group (potential signer) has a set of private attribute keys and a main private key (Step 4 in Figure 5-3). When signing the main private key is used together with a subset of the private attribute key set. The subset is decided according to the public key chosen for verification and the signature is sent to the verifier (Step 5 in Figure 5-3). We are assuming that the public key to be used is published to the group by the verifier before hand together with the tree structure. The verifier can then check the validity of the signature (Step 6 in Figure 5-3). We will refer to this type of schemes as Attribute Based Group Signatures (ABGS).

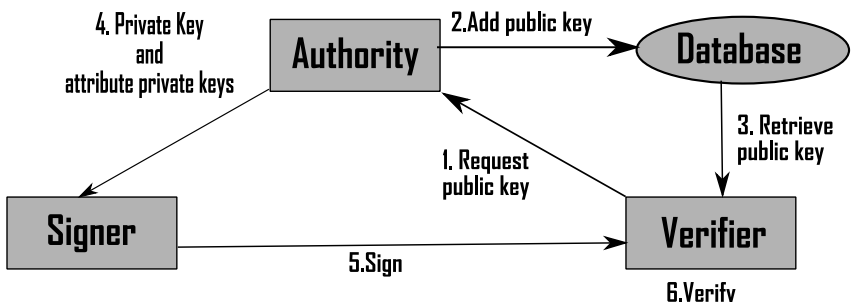


Figure 5-3: Attribute Based Group Signatures

In the following section we provide a formal definition of the scheme by explaining the algorithms used in implementing it.

### 5.3.1 Definition

In this section we define the algorithms of an ABGS. To distinguish between algorithms in this section and the ones in any other section we shall be using the prefix *ABGS*. In general we have seven main algorithms in an ABGS scheme. The following is a description of each:

- $\mathbf{ABGS.Setup(k)}$: A randomized algorithm that takes a security parameter $k$ as an input. It outputs a set of public parameters $S_{pub}$, a set of private parameters $S_{pri}$ and a tracing key $tk$.

- $\mathbf{ABGS.M.KeyGen(S_{pub}, S_{pri})}$: An algorithm that takes the system parameters. It generates what is called private key bases $bsk[i]$ for any user $i$. It also generates a public key base $w$. Finally, it calculates for all attributes a master key $t_j$, where $j$ presents an index of the attribute.

- $\mathbf{ABGS.A.KeyGen_{pub}(\Gamma, S_{pri}, S_{pub}, t_1, ..., t_\kappa)}$: This algorithm generates public

keys *gpk* for an attribute tree described in $\Gamma$ (See Figure 5-1 as an example).

- $\mathbf{ABGS.A.KeyGen_{pri}(bsk[i], \Upsilon_i, t_1, ..., t_\mu)}$: Creates the private key *gsk* for user $i$ to enable him to authenticate himself and his properties which are described in $\Upsilon_i$. The key *gsk* includes a set of private attribute keys $T_{i,j}$ that are used according to the need.

- $\mathbf{ABGS.Sign(gpk, gsk, M)}$: Given a public key of an attribute tree, a private key of a user $i$ and a message, output a signature $\sigma$ and $\Im_i$.

- $\mathbf{ABGS.Verify(gpk, M, \sigma, \Im_i)}$: Given a message, a public key of a certain attribute tree, a signature and a set $\Im_i$, output either an acceptance or a rejection for the signature.

- $\mathbf{ABGS.Open(S_{pub}, gpk, tk, t_1, ...., t_\mu, M, \sigma, \Im_i)}$: The open algorithm is given a specific signature, a public key and the tracing key as inputs. It traces to the signer $i$ even if he is a member in forging coalition. The attributes that belong to $\Im_i$ can also be traced using the tracing keys and the master keys $t_1$,...,$t_\mu$.

From the algorithms above one can spot two main disadvantages of an ABGS scheme. The first disadvantage is that the signer encloses all his attributes with the signature by sending $\Im_i$, therefore attribute anonymity is not covered. The second disadvantage is that the verifier has to retrieve a new public key from the database every time the verification policy changes. He might even need to contact the authority asking it to update the database. Later in this Chapter, Section 5.4 and Section 5.5, these two problems will be addressed. In the next section we will explain the security notions of full anonymity and full traceability for an ABGS scheme.

### 5.3.2 Security Notions

Before discussing the security of the scheme we need to define correctness. Informally, a scheme is correct if and only if valid signatures verify and all signatures trace to a member of the group even if that member is part of a coalition.

**Definition 5.3.1.** (Correctness of an ABGS scheme):
*For all keys gsk generated correctly, every signature generated by a member i verifies as valid unless the member did not have enough attributes where the signature is verified as invalid. Every valid signature traces to a member of the group. In other words,* $Verify(gpk, M, \Im_i, Sign(gpk, gsk, M)) = valid;$

*and* $Open(Sign(gpk, gsk, M), S_{pub}, tk, t_1, ..., t_\mu, M, \Im_i) = i$ *and* $\Im_i$ *is sufficient according to* $\Gamma$*;*
*Otherwise* $Verify(...) =$ *not valid;*

Anonymity and traceability are the standard acceptable notions of security for group signatures [9, 20, 19]. Hence, it is natural to require that attribute based group signatures satisfy these security notions. However, the definition of those notions must be strengthened, to adjust to the fact that the verifier decides the role of a signer in a group. We start with explaining the different oracles queried in the game model:

- **PriKey Oracle:** The Private-Key oracle allows an adversary to obtain a private key for a specific user. The adversary queries the oracle by sending it an index $i$. The oracle responds with sending the private key $bsk[i]$.
- **AttKey Oracle:** The Attribute Key oracle allows an adversary to obtain a set of private attribute keys $T_{i,j}$ for a set $\Upsilon_i$ for user $i$.
- **Signature Oracle:** This oracle allows the adversary to obtain a signature $\sigma$ on a message $M$ from user $i$. The adversary sends the tuple $(M, i, \Gamma, \Im_i)$ as its query, implying that it requires a signature of user $i$ on message $M$ under the policies described in $\Gamma$ using the subset $\Im_i$. The oracle responds with $\sigma$.
- **Open Oracle:** This oracle is meant for tracing a particular signature to a signer. The adversary queries it by sending $(\sigma, M, \Im_i)$, expecting to know the user $i$ that signed $M$ and generated $\sigma$. The oracle sends back $i$.
- **PubKey Oracle:** The Public-Key oracle allows an adversary to request a public key of a certain tree of its choice to be added to the database.

Recall that *Adam* is the adversary and *Charles* is the challenger in the game. *Charles* runs the algorithms *ABGS.Setup*, *ABGS.M.KeyGen*, and $ABGS.A.KeyGen_{pub}$. *Charles* will have created the parameters $S_{pri}$ and $S_{pub}$, $n$ private keys $bsk[i]$, tracing key $tk$, and a list of attribute public keys $gpk$. In the anonymity game the master keys, $t_j$ for every attribute $j$, are chosen by *Charles* since they are part of the Open oracle while in the traceability game they are chosen by *Adam* and given to *Charles*.

**ABGS Full Anonymity:** We say that an attribute based group signature scheme is fully *anonymous* if no polynomially bounded adversary *Adam* has a non-negligible advantage against *Charles* in the following attribute based group signature anonymity game (AAGS):

- **AAGS.Setup:** *Charles* sets up the system as described earlier by running *ABGS.Setup*, and *ABGS.M.KeyGen*. Creating the outputs $S_{pub}$, $S_{pri}$, $tk$, a

list of $bsk[i]$, $w$, and a list of $t_j$. The list of public keys, $gpk$, is created and initially it is empty. *Adam* has access to that list and is also given the public parameters $S_{pub}$. Note that *Charles* keeps the tracing key $tk$ and the master keys $t_1$,...,$t_m$ to himself since they can be used for opening signatures. The rest is given to *Adam*.

- **AAGS.Phase (1):** *Adam* can query the oracles PriKey, Open, AttKey, Signature, and PubKey as described earlier.

- **AAGS.Challenge:** *Adam* decides when to request his challenge. He sends *Charles* two indexes $(i_0, i_1)$, a message $M$, a public key of $\Gamma$ and $\Im_i$ of his choice. *Charles* replies with a signature $\sigma_b$ where $b \in \{0, 1\}$ and $\sigma_b$ is the result of signing with the triple $\langle i_b, M, \Im_i \rangle$.

- **AAGS.Phase (2):** Phase two is exactly the same as phase one except that *Adam* cannot query the Open oracle on the message challenged.

- **AAGS.Guess:** *Adam* tries to guess $\bar{b} \in \{0, 1\}$. If $b = \bar{b}$, *Adam* succeeds otherwise he fails.

The adversary is given strong attack capabilities. He has access to oracles like PriKey, AttKey, Signature, Open, and PubKey. The challenge can be queried again as a Signature oracle in Phase (2), which makes the game model defined include the notion of unlinkability If the scheme was linkable then *Adam* can win the full anonymity game explained earlier. If they were linkable, *Adam* would have queried both $i_0$ and $i_1$ from the Signature oracle, then he would have linked the signatures to the inputs of the challenge in order to determine the signer and win the game.
We define the advantage of attacking the scheme as $Adv_{AAGS}(n, k) = Pr[b = \bar{b}] - 1/2$ where $n$ is number of users and $k$ is the security parameter.

**Definition 5.3.2.** Full Anonymity:
*An ABGS scheme is fully anonymous if for any polynomial time adversary Adam, the advantage of winning the game is negligible. In other words, $Adv_{AAGS}(n, k) < \varepsilon$ where $\varepsilon$ is negligible.*

**ABGS Full Traceability:** We say that an attribute based group signature scheme is fully *traceable* if no polynomially bounded adversary *Adam* has a non-negligible advantage against *Charles* in the following attribute based group signature traceability game (TAGS):

- **TAGS.Init:** *Adam* chooses the universal set of attributes he would like to be challenged on and that is by choosing the master keys $t_1$,...,$t_m$. Both *Charles* and *Adam* can access that set.

- **TAGS.Setup:** *Charles* sets up the system as described earlier by running *ABGS.Setup* and *ABGS.M.KeyGen*. The public parameters $S_{pub}$ and the tracing key $tk$ are given to *Adam*. The parameters $S_{pri}$ on the other hand are kept a secret.

- **TAGS.Oracles:** *Adam* can query the oracles PriKey, AttKey, and Signature as described earlier. There is no need for querying the PubKey or Open oracles since *Adam* has all master keys and the tracing key.

- **TAGS.Output:** If *Adam* is successful he outputs a forged signature $\sigma$ that *Charles* fails to trace using the open algorithm. Otherwise *Adam* fails. *Charles* outputs 1 if *Adam* wins otherwise he outputs 0.

*Adam* has strong attack capabilities. He is provided with the secret tracing keys therefore he can revoke the anonymity of any signer. *Adam* also can query PriKey, AttKey, and Signature oracles. Note that the game includes unforgeability and is a strong form of coalition resistance. If we are to represent a game model individually for each of the security notions, unforgeability would be the same challenge as long as the message in that stage has not been queried before whereas coalition resistance is the same game without giving *Adam* the tracing elements.
We represent the advantage of the adversary in winning the attack as $Adv_{TAGS}(n,k) = Pr[Exp = 1]$ where $n$ is the number of users, $k$ is security parameter used in setting up the system and $Exp = 1$ refers to the game returning 1.

**Definition 5.3.3.** Full Traceability:
*An ABGS scheme is fully traceable if for any polynomial time adversary, Adam, the advantage of winning the game is negligible. That is $Adv_{TAGS}(n,k) < \varepsilon$ where $\varepsilon$ is negligible.*

In both adversarial game models we have given *Adam* the knowledge of all master keys. Therefore the game models do not capture attribute unforgeability. Given the fact that we are using the attribute tree structure which is secure against FAP attacks (Section 5.2), our scheme most probably inherits that security notion. However this is just an assumption. One of the main strengths of the schemes proposed in Section 5.4 and Section 5.5 over the ABGS scheme is that we provide a third game model to capture attribute forgeability.

In the following section we give an example on a construction of an ABGS scheme. It is based on the group signature proposed by Boneh, Boyen, and Shacham (See Section 4.3.1).

### 5.3.3 Construction

In this section we construct an ABGS scheme based on Boneh *et al*'s. work in "Short Group Signatures" in [20]. We add the prefix AGSC to distinguish between algorithms in this section and other sections. The algorithms are described below:

- **AGSC.Setup(k)**: Consider a bilinear map $e : G_1 \times G_2 \rightarrow G_3$ with all three groups multiplicative and of prime order. A computable isomorphism $\psi$ is between $G_1$ and $G_2$. Furthermore, the q-SDH (See Definition 2.2.11 and Assumption 2.2.12) is hard to solve in $G_1$ and $G_2$ and the linear problem (See Definition 2.2.14) is hard to solve in $G_1$. Select a hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$. Select a generator $g_2 \in_R G_2$ and then set $g_1 = \psi(g_2)$. Select $h \in_R G_1$ and $\xi_1, \xi_2 \in_R \mathbb{Z}_p^*$. The tracing key $tk = \langle \xi_1, \xi_2 \rangle$ will be used later in the open algorithm. Set $u, v \in G_1$ such that $u^{\xi_1} = v^{\xi_2} = h$ (by computing $u = h^{-\xi_1}$ and $v = h^{-\xi_2}$). Let $\gamma \in_R \mathbb{Z}_p^*$. Define a universe of attributes $U = \{1, 2, ..., m\}$ and for each attribute $j \in U$ choose a number $t_j \in_R \mathbb{Z}_p^*$. Let $S_{pub} = \langle G_1,\ G_2, G_3, e, H, g_1, g_2, h, u, v\ \rangle$. $S_{pri} = \langle \gamma, tk \rangle$.

- **AGSC.M.KeyGen($\mathbf{S_{pri}}, \mathbf{S_{pub}}$)**: Calculate the main public key base $w = g_2^{\gamma}$. Using $\gamma$ in $S_{pri}$ generate for each user $i$ a private key base $bsk[i] = \langle A_i, x_i \rangle$. The $bsk[i]$ should be a SDH pair where $x_i \in_R \mathbb{Z}_p^*$ and $A_i = g_1^{1/(\gamma + x_i)} \in G_1$.

- **AGSC.A.KeyGen$_{\mathbf{pub}}$($\mathbf{\Gamma}, \mathbf{S_{pri}}, \mathbf{S_{pub}}, \mathbf{t_1}, ..., \mathbf{t}_\kappa$)**: To generate a public key for a certain attribute tree $\Gamma$ the $\overline{B}$ is calculated. The $\overline{B}$ is the set of public attribute keys for attributes used in $\Gamma$. Each attribute public is $bpk_j = g_2^{t_j}$.
  The algorithm $\overline{D} = TCreate(\Gamma, \gamma, \overline{B})$ is run. $\overline{D} = \{D_1, ..., D_\kappa\}$ where $D_j = bpk_j^{q_j(0)}$. Sending $\gamma$ as the second argument in $TCreate$ implies that $q_{root} = \gamma$. The public key will be $gpk = \langle g_1,\ g_2,\ h,\ u,\ v,\ w,\ \overline{D},\ h_1, ..., h_\kappa \rangle$ where $h_j = h^{1/t_j}$.

- **AGSC.A.KeyGen$_{\mathbf{pri}}$(bsk[i], $\mathbf{\Upsilon_i}, \mathbf{t_1}, ..., \mathbf{t}_\mu$)** For every attribute $j$ that user $i$ owns (i.e. $j \in \Upsilon_i$) calculate $T_{i,j} = A_i^{1/t_j} = g_1^{1/(t_j(\gamma + x_i))}$. The private key for a user $i$ will be the tuple $gsk = \langle A_i, x_i, T_{i,1}, ..., T_{i,\mu} \rangle$.

- **AGSC.Sign(gpk, gsk, M)**: For signing user $i$, starts with choosing the set $\Im_i$ he will be using then he needs to do the following:

Let $\zeta, \beta, \alpha \in_R \mathbb{Z}_p^*$.
Compute the linear encryption (Definition 2.3.2) of $A_i$ and $T_{i,j}$ where $j \in \Im_i$. The ciphertext of the encryption will equal $C_1 = u^\zeta$, $C_2 = v^\beta$, $C_3 = A_i h^{\zeta+\beta}$, and $CT_j = (T_{i,j} h_j^{\zeta+\beta})^\alpha$. The variable $\alpha$ is used to avoid linkability of the signature, otherwise the ratio between $CT_j$ and $C_3$ is constant. In other words if $\alpha$ did not exist an adversary can compare two signatures by dividing $\frac{CT_j}{C_3}$. If the result is equal for the two signatures then the signatures must have been created by the same signer.
Let $\delta_1 = x_i\zeta, \delta_2 = x_i\beta$.
Let $r_\zeta$,$r_\beta$,$r_x$,$r_{\delta_1}$ and $r_{\delta_2} \in_R \mathbb{Z}_p^*$.
Calculate
$R_1 = u^{r_\zeta}$,
$R_2 = v^{r_\beta}$,
$R_4 = C_1^{r_x} u^{-r_{\delta_1}}$,
$R_3 = e(C_3, g_2)^{r_x} e(h, w)^{-r_\zeta - r_\beta} e(h, g_2)^{-r_{\delta_1} - r_{\delta_2}}$
$R_5 = C_2^{r_x} v^{-r_{\delta_2}}$.
Let $c = H(M, C_1, C_2, C_3, R_1, R_2, R_3, R_4, R_5) \in \mathbb{Z}_p^*$.
Construct the values $s_\zeta = (r_\zeta + c\zeta)$, $s_\beta = (r_\beta + c\beta)$, $s_x = (r_x + cx_i)$, $s_{\delta_1} = (r_{\delta_1} + c\delta_1)$, and $s_{\delta_2} = (r_{\delta_2} + c\delta_2)$.
Let the trapdoor used in verifying the tree satisfaction be $td = w^\alpha$ (Section 5.2).
The signature equals $\sigma = \langle C_1, C_2, C_3, c, CT_1,...,CT_\tau, s_\zeta, s_\beta, s_x, s_{\delta_1}, s_{\delta_2}, td, \Im_i \rangle$.

- **AGSC.Verify**$(\mathbf{gpk}, \mathbf{M}, \sigma, \Im_\mathbf{i})$: The verifier needs to run the algorithm $F_{root} = TVerify(\overline{D}, \Im_i, \bar{T})$ where $\bar{T} = \{CT_1,...,CT_\tau\}$ as shown in Section 5.2.
Note that in $Sign_{Node}(leaf)$ the value returned is $e(CT_j, D_j) = e(A_i h^{\zeta+\beta}, g_2^\alpha)^{q_j(0)}$.
Note that the value of the root polynomial when evaluated at 0 is $q_{root}(0) = \gamma$, therefore if the tree is satisfied $F_{root} = e(C_3, td)$.
Calculate
$\bar{R}_1 = u^{s_\zeta} C_1^{-c}$,
$\bar{R}_2 = v^{s_\beta} C_2^{-c}$,
$\bar{R}_4 = C_1^{s_x} u^{-s_{\delta_1}}$,
$\bar{R}_5 = C_2^{s_x} v^{-s_{\delta_2}}$,
$\bar{R}_3 = e(C_3, g_2)^{s_x} e(h, w)^{-s_\zeta - s_\beta} e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} (\frac{e(C_3, w)}{e(g_1, g_2)})^c$.
If $c = H(M, C_1, C_2, C_3, \bar{R}_1, \bar{R}_2, \bar{R}_3, \bar{R}_4, \bar{R}_5)$ then accept the signature, otherwise reject it.

- **AGSC.Open**$(\mathbf{S_{pub}}, \mathbf{gpk}, \mathbf{tk}, \mathbf{t_1}, ..., \mathbf{t}_\tau, \mathbf{M}, \sigma, \Im_\mathbf{i})$: This algorithm traces a signature to a signer. To do so the authority will be using the $\langle G_1, G_2, G_3, e, H, g_1,$

$g_2$, $h$, $u$, $v$, $\Gamma$, $w\rangle$ derived from $S_{pub}$ and $gpk$.

Step one in the tracing will be verifying the signature. Afterwards, the group manager can recover $A_i$ by calculating $A_i = C_3/(C_1^{\xi_1} C_2^{\xi_2})$. Now the manager can look up the user with index $A_i$. The manager can also verify the attributes. For each attribute, he checks the following equality $e(CT_j, w) = e((A_i C_1^{\xi_1} C_2^{\xi_2})^{1/t_j}, td)$. If the equality holds for an attribute $j$ then the $j$ is said to be traced to the same user $i$.

The reason behind limiting the possibility of being the group manager to the key generator is the need to use $t_j$ when calculating $j$. Furthermore, the key generator can not convey $t_j$ in an encrypted matter to the group manager because that implies that he can create private attribute keys therefore increasing the number of authorities we need to trust other than the key generator.

### 5.3.4 General Discussion of the ABGS Scheme

In this section we discuss the advantages and disadvantages of the scheme in general. In the proposed scheme we managed to cover some issues discussed in Section 5.1. The scheme is anonymous and unlinkable if proven secure under the full anonymity game. The scheme is unforgeable, traceable and coalition resistant when proved secure under the full traceability game. However, in the ABGS scheme we have failed to achieve attribute anonymity since the signer encloses $\Im_i$ with the signature. This means the verifier knows what attributes were used in creating the signature. The other disadvantage is not providing separability at any level. The authority creates all keys whether it is attribute based or non-attribute based. The authority is also responsible of tracing the signatures. It would have been nice if we had some kind of hierarchy. For example, if the employee is in department "A", he should get his key from the department rather than the manager Bob. It will also be nice to have the department trace the signer rather than Bob. In the ABGS scheme this is not possible since it requires knowing the tracing key and the private master key(s) of the signer. For instance, in scenario of Section 5.2 the verifier needs the signer to be a senior manager in department "A". If the department is responsible of tracing it needs to know the master key of attribute "senior manager". Separability decreases the bottleneck on the manager. It also improves the security of the scheme since not even Bob can tell which attributes a user possess. Another disadvantage is having the verification requirement known to all, the policy of the verifier is exposed. Finally, the security proofs can be strengthened by providing a third game model, which has not been done in this section. That game model should capture attribute unforgeability property rather than depending on the fact that we are using the tree structure. A possible design for that game model is to allow *Adam* to obtain all private keys and $m-1$ master keys in the system. *Adam* is asked to provide a signature that proves

possession of the missing attribute. If he succeeds in generating a valid signature then he wins the game otherwise he loses. In the schemes proposed in Section 5.4 and Section 5.5 we provide such game model.
ABGS scheme has covered a considerable amount of properties desired in implementing a new attribute based scheme, however it has its drawbacks.

### 5.3.5 Analysis of the Construction of the ABGS Scheme 5.3.3

Previously we discussed how the ABGS has some advantages and disadvantages. These were inherent in the construction itself. We opt to discuss correctness of the constructed scheme. We then give a brief proof of security of the scheme under the definitions of full anonymity and traceability. In Appendix A we give a more comprehensive proof. We also discuss some efficiency issues in our construction. We shall start with correctness.

**Theorem 5.3.4.** *The Construction in Section 5.3.3 is correct according to definition 5.3.1.*

In order to prove correctness we need to show that $\bar{R}_1 = R_1$, $\bar{R}_2 = R_2$, $\bar{R}_3 = R_3$, $\bar{R}_4 = R_4$, $\bar{R}_5 = R_5$ because that leads to $c = H(M, C_1, C_2, C_3, \bar{R}_1, \bar{R}_2, \bar{R}_3, \bar{R}_4, \bar{R}_5)$ which means the signature is accepted.

$$\bar{R}_1 = u^{s_\zeta} C_1^{-c} = u^{r_\zeta + c\zeta}(u^\zeta)^{-c} = u^{r_\zeta} = R_1$$

$$\bar{R}_2 = v^{s_\beta} C_2^{-c} = v^{r_\beta + c\beta}(v^\beta)^{-c} = v^{r_\beta} = R_2$$

$$\bar{R}_4 = C_1^{s_x} u^{-s_{\delta_1}} = u^{\zeta(r_x + cx)} u^{(-r_{\delta_1} - c\delta_1)} = C_1^{r_x} u^{-r_{\delta_1}} = R_4$$

$$\bar{R}_5 = C_2^{s_x} v^{-s_{\delta_2}} = v^{\beta(r_x + cx)} v^{(-r_{\delta_2} - c\delta_2)} = C_2^{r_x} v^{-r_{\delta_2}} = R_5$$

Finally, $\bar{R}_3 = R_3$ holds for the following reasons:

$$\bar{R}_3 = e(C_3, g_2)^{s_x} e(h, w)^{-s_\zeta - s_\beta} e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} (\tfrac{e(C_3, w)}{e(g_1, g_2)})^c$$

$$= e(C_3, g_2)^{r_x + cx} e(h, w)^{-r_\zeta - r_\beta - c\zeta - c\beta} e(h, g_2)^{-r_{\delta_1} - r_{\delta_2} - cx\zeta - cx\beta} (\tfrac{e(C_3, w)}{e(g_1, g_2)})^c$$

$$= (e(C_3, g_2)^{cx} e(h, w)^{-c\zeta - c\beta} e(h, g_2)^{-cx\zeta - cx\beta})(e(C_3, g_2)^{r_x} e(h, w)^{-r_\zeta - r_\beta} e(h, g_2)^{-r_{\delta_1} - r_{\delta_2}})(\tfrac{e(C_3, w)}{e(g_1, g_2)})^c$$

$$= (e(C_3, g_2^x)^c e(h^{-\zeta - \beta}, w)^c e(h^{-\zeta - \beta}, g_2^x)^c)(R_3)(\tfrac{e(C_3, w)}{e(g_1, g_2)})^c$$

$$= e(C_3 h^{-\zeta - \beta}, w g_2^x)^c e(C_3, w)^{-c} (R_3)(\tfrac{e(C_3, w)}{e(g_1, g_2)})^c$$

$= (\frac{e(A_i, wg_2^x)}{e(C_3, w)})^c (R_3)(\frac{e(C_3, w)}{e(g_1, g_2)})^c$

$= (\frac{e(g_1, g_2)}{e(C_3, w)})^c (R_3)(\frac{e(C_3, w)}{e(g_1, g_2)})^c$

$= R_3$

Under "correctness of the scheme" we should also discuss the open algorithm. In the open algorithm we recover $A_i$ by calculating

$C_3/(C_1^{\xi_1} C_2^{\xi_2}) = (A_i h^{\zeta+\beta})/(u^{\zeta\xi_1} v^{\beta\xi_2})$

$= (A_i h^{\zeta+\beta})/(h^{\zeta+\beta}) = A_i$

The next step is to verify $e(CT_j, w) = e((A_i C_1^{\xi_1} C_2^{\xi_2})^{1/t_j}, td)$ for each attributes $CT_j$. They must be equal if $A_i$ used in computing $CT_j$ is the same as the one used in calculating the right hand side of the equation:

$e(CT_j, w) = e((T_{i,j} h_j^{\zeta+\beta})^\alpha, w) = e((A_i h^{\zeta+\beta})^{1/t_j}, td) = e((A_i C_1^{\xi_1} C_2^{\xi_2})^{1/t_j}, td).$

After proving correctness of the scheme we shall discuss full anonymity and full traceability.

**Theorem 5.3.5.** *If the linear encryption is IND-CPA secure then the ABGS scheme is fully anonymous, under the same attribute set, under the random oracle assumption.*

In other words, if there is an adversary *Adam* that breaks the scheme's full anonymity then there exists an adversary *Eve* that breaks into the linear encryption IND-CPA security. It makes sense to assume anonymity under the same attribute set, otherwise you can easily distinguish between signatures from attributes owned by each signer. For instance if user $i_0$ in the challenge has a different set of attributes than $i_1$, then *Adam* can know the signer from the list $\bar{T}$=$\{CT_1,...,CT_\tau\}$ enclosed with the signature. To prove Theorem 5.3.5, we run the adversarial model defined in Section 5.3.1. We will assume we have an adversary *Adam* attacking the ABGS scheme. Let *Eve* be the adversary threatening the linear encryptions IND-CPA security. *Eve* will play a role of a challenger with *Adam*. She will make use of his talent to break the IND-CPA security. When *Adam* wants to be challenged, he sends $i_0, i_1$, a message $M$, an attribute structure $\Gamma$ and a set $\Im_i$ to *Eve*. *Eve* has the values $A_{i_0}, A_{i_1}$ since she is the one who ran the setup. She will give $A_{i_0}, A_{i_1}$ as messages to challenge the IND-CPA security of the linear encryption. She will get back a ciphertext of one of them, $A_{i_b}$. The ciphertext is in the form $\bar{C} = \langle C_1, C_2, C_3 \rangle$, where $C_1 = u^\zeta$, $C_2 = v^\beta$, and $C_3 = A_{i_b} h^{\zeta+\beta}$. *Eve* can calculate $CT_j = C_3^{\alpha/t_j}$. She can then calculate

$c, \Im_i, s_\zeta, s_\beta, s_x, s_{\delta_1}$, and $s_{\delta_2}$ (Details in Appendix A.1.1). *Eve* sends *Adam* the signature of $i_b$ as $\sigma_b = \langle C_1, C_2, C_3, c, CT_1, ..., CT_\mu, s_\zeta, s_\beta, s_x, s_{\delta_1}, s_{\delta_2}, \Im_i, td\rangle$. Notice that *Eve* herself does not know $b$. If *Adam* breaks the ABGS anonymity, he will send *Eve* the right value of $b$. *Eve* will use it to know whether $A_{i_0}$ or $A_{i_1}$ has been encrypted. Therefore, *Eve* breaks the IND-CPA security of linear encryption. In Appendix A.1.1 we describe more details about the proof.

**Theorem 5.3.6.** *If q-SDH is hard on group $G_1$ and $G_2$ then the ABGS scheme is fully-traceable under the random oracle assumption where $q$ is related to the number of user $n$.*

In other words, if there is an adversary *Adam* that breaks the traceability of the scheme then the q-SDH problem is solved. The comprehensive proof of Theorem 5.3.6 is given in the Appendix A.2.1. A simplified version will be explained in this section.
In our proof we use the game described in Section 5.3.1, the Forking Lemma (Theorem 3.4.1), and Boneh–Boyen theorem (Theorem 2.2.13). A signature will be represented as $\langle M, \sigma_0, c, \sigma_1, \sigma_2\rangle$. $M$ is the signed message. $\sigma_0 = \langle C_1, C_2, C_3, R_1, R_2, R_3, R_4, R_5\rangle$. $c$ is the value derived from hashing $\sigma_0$. $\sigma_1 = \langle s_\zeta, s_\beta, s_x, s_{\delta_1}, s_{\delta_2}\rangle$ which are values used to calculate the missing inputs for the hash function. Finally $\sigma_2 = \langle CT_1, ..., CT_\tau, \Im_i, td\rangle$ the values that depend on the set of attributes in each Signature oracle. Notice $\sigma_2$ does not exist in the Forking lemma (Theorem 3.4.1) and the reason is it was introduced in this thesis is to hold attribute related elements of the signature.
We will run the game in Section 5.3.1 twice. In both simulated runs, *Charles* is given an $(n)$ SDH instance, $(\grave{g}_1, \grave{g}_2, \grave{g}_2^\gamma, \grave{g}_2^{\gamma^2}, ..., \grave{g}_2^{\gamma^q})$. By applying the Boneh–Boyen's theorem, *Charles* can obtain $g_1 \in G_1, g_2 \in G_2, w = g_2^\gamma$ and $(n-1)$ SDH pairs $(A_i, x_i)$ which he will use as the private key bases $bsk[i]$.
The next step is showing how the Forking Lemma can be applied here to prove that a new SDH pair can be generated, if a forgery exists. The difference between the two simulated runs is the response to the hash oracle (See Appendix A.2.1). According to the Forking Lemma, if *Adam* can find with non-negligible probability a valid signature $\langle M, \sigma_0, c, \sigma_1, \sigma_2\rangle$, then with a replay another valid signature $\langle M, \sigma_0, \grave{c}, \grave{\sigma}_1, \sigma_2\rangle$ is outputted with a non-negligible probability. Recall from the Forking Lemma that $c \neq \grave{c}$.
We show how we can 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 $\Delta c = c - \grave{c}$, $\Delta s_\zeta = s_\zeta - \grave{s}_\zeta$, and similarly for $\Delta s_\beta$,$\Delta s_x$,$\Delta s_{\delta_1}$, and $\Delta s_{\delta_2}$.
Divide two instances of the equations used previously in proving Theorem 5.3.4 where one instance is with $\grave{c}$ and the other is with $c$ to get the following:

- Dividing $R_1/\grave{R}_1$:

$$1 = \frac{R_1}{\grave{R}_1} = \frac{\bar{R}_1}{\grave{\bar{R}}_1} = \frac{u^{s_\zeta} C_1^{-c}}{u^{\grave{s}_\zeta} C_1^{-\grave{c}}} = \frac{u^{\Delta s_\zeta}}{C_1^{\Delta c}}$$

$u^{\tilde{\zeta}} = C_1$; where $\tilde{\zeta} = \Delta s_\zeta / \Delta c$

- Dividing $R_2/\grave{R}_2$:

  $1 = \frac{R_2}{\grave{R}_2} = \frac{\bar{R}_2}{\grave{\bar{R}}_2} = \frac{v^{s_\beta} C_2^{-c}}{v^{\grave{s}_\beta} C_2^{-\grave{c}}} = \frac{v^{\Delta s_\beta}}{C_2^{\Delta c}}$

  $v^{\tilde{\beta}} = C_2$; where $\tilde{\beta} = \Delta s_\beta / \Delta c$

- Dividing $C_1^{s_x}/C_1^{\grave{s}_x}$:

  $C_1^{\Delta s_x} = C_1^{\Delta cx} = u^{\zeta x \Delta c} = u^{\delta_1 \Delta c} = u^{\delta_1 c - \delta_1 \grave{c}} = u^{\delta_1 c - \delta_1 \grave{c} + r_{\delta_1} - r_{\delta_1}} = u^{\Delta s_{\delta_1}}$

  where $\Delta s_{\delta_1} = \tilde{\zeta} \Delta s_x$ because

  $\tilde{\zeta} \Delta s_x = (\frac{\Delta s_\zeta}{\Delta c}) \Delta s_x = \left( \frac{\zeta \Delta c + (r_\zeta - r_\zeta)}{\Delta c} \right) (x \Delta c + (r_x - r_x)) = \delta_1 \Delta c = \Delta s_{\delta_1}$

- Dividing $C_2^{s_x}/C_2^{\grave{s}_x}$:

  $C_2^{\Delta s_x} = C_2^{\Delta cx} = v^{\beta x \Delta c} = v^{\delta_2 \Delta c} = v^{\delta_2 c - \delta_2 \grave{c}} = v^{\delta_2 c - \delta_2 \grave{c} + r_{\delta_2} - r_{\delta_2}} = v^{\Delta s_{\delta_2}}$

  where $\Delta s_{\delta_2} = \tilde{\beta} \Delta s_x$ because

  $\tilde{\beta} \Delta s_x = \left( \frac{\Delta s_\beta}{\Delta c} \right) \Delta s_x = \left( \frac{\beta \Delta c + (r_\beta - r_\beta)}{\Delta c} \right) (x \Delta c + (r_x - r_x)) = \delta_2 \Delta c = \Delta s_{\delta_2}$

- Dividing $\frac{(e(g_1,g_2)/e(C_3,w))^c}{(e(g_1,g_2)/e(C_3,w))^{\grave{c}}}$:

  $(e(g_1, g_2)/e(C_3, w))^{\Delta c}$

  $= e(C_3, g_2)^{\Delta s_x} e(h, w)^{-\Delta s_\zeta - \Delta s_\beta} e(h, g_2)^{-\Delta s_{\delta_1} - \Delta s_{\delta_2}}$

  $= e(C_3, g_2)^{\Delta s_x} e(h, w)^{-\Delta s_\zeta - \Delta s_\beta} e(h, g_2)^{-\tilde{\zeta} \Delta s_x - \tilde{\beta} \Delta s_x}$

  The above equations are similar to the calculation of $\bar{R}_3 = R_3$ in the correctness proof.

Let $\tilde{x} = \Delta s_x / \Delta c$ and $\tilde{A} = C_3 h^{-(\tilde{\zeta} + \tilde{\beta})}$ we can compute the following:

Recall that $(e(g_1, g_2)/e(C_3, w))^{\Delta c} = e(C_3, g_2)^{\Delta s_x} e(h, w)^{-\Delta s_\zeta - \Delta s_\beta} e(h, g_2)^{-\tilde{\zeta} \Delta s_x - \tilde{\beta} \Delta s_x}$

This implies that $e(g_1, g_2)/e(C_3, w)$=$e(C_3, g_2)^{\tilde{x}} e(h, w)^{-\tilde{\zeta}-\tilde{\beta}} e(h, g_2)^{-\tilde{x}(\tilde{\zeta}+\tilde{\beta})}$

Furthermore the equality $e(g_1, g_2)$=$e(\tilde{A}, wg_2^{\tilde{x}})$ holds.
Hence we obtain a new SDH pair $(\tilde{A}, \tilde{x})$ breaking Boneh–Boyen's theorem.
The ABGS is fully anonymous and fully traceable under the random oracle assumption.
Finally, we shall comment on the efficiency of the scheme. The private key size, the public key size and the signature are linearly dependent on the number of attributes used in creating them. This is not practical if we are to use the system for larger scale. It will also be nice to have the verifier decide the attribute tree and create a verification key without the need to contact the authority each time.
Although our proposed scheme implements the major properties we require, we desire more. This led us to come up with the idea of an AAS scheme that is explained in the following section.

## 5.4 Phase II: Attribute Authentication Scheme

As discussed in Section 5.3.4 an ABGS did improve a lot on existing authentication schemes. Nevertheless, it had drawbacks. It does not achieve all required properties explained for scenario 5.1.1. Therefore we have proposed a new system referred to as the Attribute Authentication Scheme (AAS).
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 1a Figure 5-4) and this will be used in verifying a signature. The central authority creates many pairs of private key bases and registration keys to be used in signing messages. Each member (possible signer) of the group will be given their unique pair (i.e. Bob gives each employee a pair as shown in Step 1b Figure 5-4). Each member will be using the registration key to register with an attribute authority (Step 2 in Figure 5-4). Registering with an authority implies getting a copy of a private attribute key (Step 3a in Figure 5-4). 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 3b in Figure 5-4). The verifier (Alice) will create a verification key using the public key base and as many public attribute keys as needed (Step 4 in Figure 5-4). 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 5-4). The system can have many attribute authorities but has to have one central authority. The central authority can be an attribute authority

itself.


Figure 5-4: Attribute Authentication Scheme

### 5.4.1 Definition

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 by the authority, while $S_{pub}$ is published for all to see and use.

- $\mathbf{M.KeyGen(S_{pri}, S_{pub})}$: This algorithm is run by the central authority. It takes the inputs $S_{pri}$, and $S_{pub}$. It then generates pairs of private key bases $bsk[i]$ and registration keys $A_i$ where the pairs are distributed to the members of the group. Then $S_{pub}$ and $S_{pri}$ are used to generate a public key base $w$ known to all. The $bsk[i]$ is kept private to the user, while the $A_i$ is given to trusted third parties, as shown in $A.KeyGen_{pri}$. Later on, in this chapter, we will show how to replace this algorithm with a protocol that conceals $A_i$ (Section 5.4.6) from everyone but the member.

- $\mathbf{A.KeyGen_{pub}(S_{pub}, w)}$ : This algorithm is run by the attribute authorities. Each authority is responsible for one or more attributes and for every attribute $j$, the authority creates a corresponding master key $t_j$. It is this key which is used to create the attribute-related keys. Using the master keys, the public parameters $S_{pub}$ and public key base $w$, 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 $t_j$.

- $\mathbf{A.KeyGen_{pri}(A_i, j, RL)}$ : 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

$T_{i,j}$. Due to the fact that the attribute authority is supposed to be independent from the central authority it is best to check 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 $gsk = \langle bsk[i], A_i, T_{i,1}, ..., T_{i,\mu} \rangle$ 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. We will exchange this algorithm with a protocol that hides $A_i$ from the attribute authority and gets the same output of this algorithm (Section 5.4.6).

- **Verifign**$(\mathcal{U}_\mathbf{i}(\mathbf{gsk}, \mathbf{M}), \mathcal{V}(\mathbf{M}, \bar{\mathbf{B}}, \mathbf{w}))$ : 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 $\overline{D}$ to be sent to the group. The signer uses his private key $gsk$, the message $M$ and $\overline{D}$ as inputs to an algorithm **Sign** that creates a signature $\sigma$ that corresponds to $\overline{D}$ as follows $\sigma = Sign(gsk, M, \overline{D})$. $\mathcal{U}_i$ sends $\sigma$ to $\mathcal{V}$ and the verifier runs an algorithm **Verify** using $\sigma$, $M$, $\overline{D}$ and $w$ as inputs. The algorithm checks the validity of the signature and whether or not the signer is revoked as follows $Verify(\sigma, M, \overline{D}, w, RL) = \{Accept, Reject\}$.

- **Revoke**$(\mathbf{A_i}, \mathbf{RL})$ : This algorithm is run by the central authority. It adds a registration key $A_i$ of revoked users to the revocation list $RL$.

- **ChkRvk**$(\sigma, \mathbf{RL})$ : This algorithm takes a signature $\sigma$ and a revocation list $RL$. It returns $i$ if the user is revoked and on the list otherwise it returns $-1$. $ChkRvk$ is an algorithm run by either the verifier or the central authority. The index returned does not mean much to the verifier. The verifier only wants to know whether the user has been revoked or not. So if the value of the index is anything other than $-1$ the user has been revoked. If $ChkRvk$ is run by the central authority then the index refers to a user and that will help in tracing signatures to users as explained in the following *Open* algorithm.

- **Open**$(\sigma, \mathbf{FRL})$ : This algorithm is meant for tracing a signer in case of a dispute and revoking his anonymity. $FRL$ is a list created by the central authority who stores the value $A_i$ for all members of the groups and adds all members (while maintaining indexes) to that 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 $FRL$.

### 5.4.2 Security Notions

Like any other cryptographic scheme before we prove security of an AAS scheme we have to ensure correctness.

**Definition 5.4.1.** (Correctness of an AAS scheme): *For all keys gsk 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. Every valid signature should trace to a member of the group. In other words,*
$Verify(\overline{D}_v, w, M, RL, Sign(M, \overline{D}_s, gsk)) = valid;$
*where* $ChkRvk(Sign(M, \overline{D}_s, gsk), RL) = -1$, $\overline{D}_s = \overline{D}_v$
*and* $Open(Sign(M, \overline{D}_s, gsk)) = i;$
*Otherwise* $Verify(...) = not\ valid;$

The next stage is to define adversarial models that will include security notions mentioned in Section 5.1. 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*. Three adversarial models are enough to cover all security notions. We refer to them as the "full anonymity" model, the "full traceability" model, and the "unforgeability of attributes" model. The three models allow *Adam* to query certain oracles run by *Charles*. In the full anonymity and full traceability game model 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.
Before we define the adversarial models we list the oracles:

- **USK Oracle**: To query this oracle the adversary sends the challenger an index $i$ in order to find the user's private key base $bsk[i]$ and registration key $A_i$. The challenger will send the pair $(bsk[i], A_i)$ to the adversary. Note that since the adversary 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 the adversary sends the challenger a verification key he created $\overline{D}$, a message $M$ and an index $i$. The adversary requires the signature of member $i$ on message $M$ using $\overline{D}$. The challenger sends $\sigma$

- **Revoke Oracle**: To query this oracle the adversary sends the challenger an index $i$. The challenger adds $A_i$ to the list $RL$. The list $RL$ is public and the adversary can access it (Note that the Revoke oracle replaces the Open oracle in Section 5.3 and 5.5. This is a logical consequence of the fact that the open algorithm depends on the revocation technique as explained in the definition).

- **AttPriKey Oracle**: To query this oracle the adversary sends a registration key $A_i$ to the challenger together with an index of the attribute $j$. The output of this query is a private attribute key $T_{i,j}$.

- **AttMasKey Oracle**: To query this oracle the adversary sends an index $j$ to the challenger. The challenger responds with sending the output $t_j$.

*Adam* initializes both the traceability and anonymity 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 *AAAS* game:

- **AAAS.Setup:** *Charles* plays the role of the central authority. He runs the algorithms *Setup*, and *M.KeyGen* to produce the systems $S_{pub}$, and $S_{pri}$. *Charles* also generates the $n$ private key bases $bsk[i]$ and $n$ registration keys $A_i$. He sends to *Adam* parameters $S_{pub}$. Finally, both *Adam* and *Charles* generate $m$ public attribute keys $\langle bpk_1,...,bpk_m \rangle$.

- **AAAS.Phase (1):** *Charles* runs the three oracles, USK, Signature oracle, and Revoke oracle as explained above. Adam can query these oracles to obtain any information he thinks he will require in breaking the scheme.

- **AAAS.Challenge:** *Adam* asks to be challenged on a message $M$, two indexes $i_0$,$i_1$, and verification key $\overline{D}$. *Charles* responds back with a signature $\sigma_b$, 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 Signature oracle.

- **AAAS.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.

- **AAAS.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 Signature. *Adam* does not need to query oracles AttMasKey and AttPriKey because he obtains all master keys $t_j$. 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 (*ChkRvk* is used) or if he has the private key base then *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 winning the game is $Adv_{AAAS}(n,k) = Pr[b = \bar{b}] - 1/2$, where $n$ is the maximum bound of the numbers of members and $k$ is the security parameter used for setting up the system.

**Definition 5.4.2.** Full Anonymity*:*
*A scheme is fully anonymous if for all polynomial time adversary Adam,* $Adv_{AAAS}(n,k) < \varepsilon$ *and* $\varepsilon$ *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 $TAAS$ game:

- **TAAS.Setup:** *Charles* plays the role of the central authority as done in the setup stage of the anonymity game model mentioned earlier. He runs the algorithms *Setup*, and $M.KeyGen$ to produce $S_{pub}$, and $S_{pri}$. *Charles* also generates $n$ private key bases $bsk[i]$ and $n$ registration keys $A_i$. $S_{pub}$ are sent to *Adam* and *Adam* is given all registration keys $A_i$. Finally, both *Adam* and *Charles* generate $m$ public attribute keys $\langle bpk_1,...,bpk_m \rangle$.

- **TAAS.Queries:** *Charles* runs two oracles: USK oracle, and Signature oracle. *Adam* queries them as described earlier.

- **TAAS.Output:** *Adam* asks to be challenged on a message $M$ which he sends to *Charles*. *Charles* calculates a new $\overline{D}$ and sends it back to *Adam*. *Adam* replies with a signature $\sigma$. *Charles* verifies the signature. If it is not valid return 0. If it turns out 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. *Adam* does not need to query oracles AttMasKey and AttPriKey because he obtains all master keys $t_j$. 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 $Adv_{TAAS}(n,k) = Pr[Exp = 1]$.

**Definition 5.4.3.** Full Traceability:
*A scheme is fully traceable if for all polynomial time adversary Adam,* $Adv_{TAAS}(n,k) < \varepsilon$ *and* $\varepsilon$ *is negligible.*

**AAS Unforgeability of Attributes:** We say that our AAS scheme is Attribute-Unforgable if no polynomially bounded adversary *Adam* has a non-negligible advantage against the challenger *Charles* in the following AFAAS game:

- **AFAAS.Setup:** *Charles* plays the role of the central authority and all attribute authorities in the system. He runs the algorithms *Setup*, and *M.KeyGen* to produce $S_{pub}$, and $S_{pri}$. *Charles* also generates $n$ private key bases $bsk[i]$ and $n$ registration keys $A_i$. *Charles* can produce the set of all attributes in the system since he is playing the role of all attribute authorities. He generates a set of master keys $t_1, ..., t_m$. He creates all public attributes needed $bpk_1, ..., bpk_m$. *Adam* is given $S_{pub}$, the list of registration keys $A_i$, and the set of attribute public keys $bpk_j$. *Charles* keeps the list of private keys $bsk[i]$, parameters $S_{pri}$ and the list of master keys $t_j$.

- **AFAAS.Phase (1):** *Charles* runs the oracles Signature, USK, AttPriKey and AttMasKey. *Adam* can query these oracles to obtain extra information he may require for the attack. *Adam* does not need the Revoke oracle since he has all registration keys $A_i$ and can trace signatures using the open algorithm.

- **AFAAS.Challenge:** *Adam* sends a tree $\Gamma_1$, user $l$ and attribute $z$ in which he would like to be challenged on. *Charles* replies with $\overline{D}$ for a tree $\Gamma_2$ where $\Gamma_2$ has two subtrees the first is $\Gamma_1$ and the other is based on $t_z$. The threshold value of the root in $\Gamma_2$ is 2. The challenge condition is that user $l$ has not been queried in AttPriKey for the attribute $z$. Furthermore the challenged index $z$ should not have been queried in AttMasKey. These two conditions are reasonable as violating them would contradict the purpose of the game.

- **AFAAS.Phase (2):** This phase is similar to Phase 1 as long as the challenge conditions are not broken.

- **AFAAS.Output:** *Adam* outputs a signature $\sigma$ for the user $l$ on the verification key $\overline{D}$. If that signature is valid then the adversary wins and *Charles* outputs 1 otherwise *Adam* loses and *Charles* outputs 0.

In this game *Adam* is given strong attack capability. He can trace any signature since he has all registration keys. He can corrupt users since he can query the USK oracle. He can corrupt attribute authorities by obtaining the master keys with the AttMasKey oracle. The challenge is to create a signature using an attribute $z$ that has not been queried in AttMasKey and a signer $l$ that does not have the attribute private key for $z$.

If *Adam* can output a valid signature that proves that user $l$ has attribute $z$ he would win the game. If we refer to the returned value as $Exp$ then the advantage of winning the game is notated as $Adv_{AFAAS}(n,k) = Pr[Exp = 1]$

**Definition 5.4.4.** Unforgeability of Attributes:
*An AAS scheme is attribute-unforgeable if for all polynomial time adversary Adam, $Adv_{AFAAS}(n,k) < \varepsilon$ and $\varepsilon$ is negligible.*

### 5.4.3 Construction of our AAS Scheme

In this section we will construct our AAS scheme based on the work done by Boneh and Shacham [25]. We now go through the algorithms defined in Section 5.4.1 and show how we can build them. We will use the abbreviation *AASC* to distinguish between algorithms here and in other sections of the thesis.

- **AASC.Setup**$(k)$: Consider a bilinear pair $e : G_1 \times G_2 \rightarrow G_3$ where $(G_1, G_2)$ have a computable isomorphism $\psi$ from $G_2$ to $G_1$ and all three groups $G_1, G_2$ and $G_3$ are multiplicative and of prime order $p$. Suppose that the q-SDH problem is hard to solve in $G_1$ and $G_2$ (See Definition 2.2.11 and Assumption 2.2.12) and the decision linear problem (Definition 2.2.14) is hard to solve in $G_2$. Select a hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$. Select a hash function $H_0$ with range $G_2^2$. Let $g_2 \in_R G_2$ and then set $g_1 = \psi(g_2)$. Let $\gamma \in_R \mathbb{Z}_p^*$, then $S_{pub} = \langle G_1, G_2, G_3, e, H, H_0, g_1, g_2 \rangle$ and $S_{pri} = \gamma$.

- **AASC.M.KeyGen**$(\mathbf{S_{pri}}, \mathbf{S_{pub}})$ : Using $\gamma$ generate for each user $i$ a private key base $bsk[i] = \langle A_i, x_i \rangle$. All $bsk[i]$ should be SDH pairs, where $x_i \in_R \mathbb{Z}_p^*$ 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 $w = g_2^{\gamma}$.

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

- **AASC.A.KeyGen$_{\mathbf{pri}}$**$(\mathbf{A_i}, \mathbf{j}, \mathbf{RL})$: 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 $i$. After running this algorithm with different attribute authorities the user $i$ should obtain a private key $gsk = \langle A_i,\ x_i,\ T_{i,1},\ldots,T_{i,\mu} \rangle$.

- **AASC.Verifign**$(\mathcal{U}_\mathbf{i}(\mathbf{gsk}, \mathbf{M}), \mathcal{V}(\mathbf{M}, \overline{\mathbf{B}}))$ : The protocol runs as follows:

  1. $\mathcal{V}$ has collected all public key bases it needs $\overline{B} = (bpk_1,...,bpk_\kappa)$. $\mathcal{V}$ then chooses an element $\alpha \in_R \mathbb{Z}_p^*$ and decides a $\Gamma$. Finally, $\mathcal{V}$ calculates $\overline{D} = TCreate(\Gamma, \alpha, \overline{B}) = (D_1,...,D_\kappa)$ (Section 5.2) and sends to $\mathcal{U}_i$.

  2. $\mathcal{U}_i$ in this stage runs the algorithm $Sign(M, gsk, \overline{D})$ as follows:
     Calculate $F_{root} = TVerify(\overline{D}, \Upsilon_i, \bar{T})$ (Section 5.2) where $\bar{T} = \{T_{i,1}^\beta,...,T_{i,\kappa}^\beta\}$, $\beta \in_R \mathbb{Z}_p^*$ and $\Upsilon_i$ is the set of attributes the user decides to use when signing.

     $\mathcal{U}_i$ selects an $r \in_R \mathbb{Z}_p^*$ to obtain $H_0(\overline{D}, M, r) = (\bar{u}, \bar{v})$. Next $\mathcal{U}_i$ computes the images $u = \psi(\bar{u})$ and $v = \psi(\bar{v})$. Let $\xi$, $\beta$, $r_\xi$, $r_x$, and $r_\delta \in_R \mathbb{Z}_p^*$.

     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 $\delta = x_i\xi$, $s_\xi = r_\xi + c\xi$, $s_x = r_x + cx_i$ 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 $\mathcal{V}$. Note that the trapdoor to prove the satisfaction of the tree $\Gamma$ is $td = (C_3, C_4)$ (Section 5.2).

  3. In this stage the verifier $\mathcal{V}$ calculates $Verify(\sigma, M, \overline{D}, w, RL)$ as follows:
     Calculate $H_0(\overline{D}, M, r) = (\bar{u}, \bar{v})$ then $u = \psi(\bar{u})$, and $v = \psi(\bar{v})$. The 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 $AASC.ChkRvk(\sigma, RL)$ 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.

- **AASC.ChkRvk**$(\sigma, \mathbf{RL})$ : This algorithm takes a signature $\sigma$ 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 the index $i$ 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 **Revoke**$(\mathbf{A_i}, \mathbf{RL})$ and **Open**$(\sigma, \mathbf{FRL})$ are exactly as described in Section 5.4.1 and do not need to be explained further.

### 5.4.4 General Discussion of the AAS Scheme

In this section we give a general discussion about the AAS scheme proposed in Section 5.4.1. An AAS scheme is anonymous and unlinkable once proven secure under full anonymity definition. It is also unforgeable, coalition resistant, and traceable once proven secure under the full traceability definition and the attribute unforgeability model. The idea of attribute authorities enables separability in the scheme. Each attribute authority creates its key independently. The central authority is the only entity capable of tracing signatures to members of the group. The verifier does not have to contact the central authority each time he has changed his policy (i.e. $\Gamma$) unless he does not have the public attribute key $bpk_j$ for an attribute he requires. He is the one who creates the verification key. The scheme covers all properties required in Section 5.1. Anyone can verify a signature belongs to a member of the group but only the verifier can know whether or not the signer satisfies the attribute tree requested in that signature. This is the outcome of enabling the verifier to create the $\overline{D}$ rather than download them from a database as done in Section 5.3. The verification key is linearly dependent on the number of attributes used in creating it. It would have been nice if we managed to hide the verifier's request of attributes from those who do not possess enough attributes.

### 5.4.5 Analysis of the Construction of the AAS Scheme 5.4.3

Naturally, the construction inherits the advantages and disadvantages of the general scheme. We shall discuss more specific properties for the construction. We start by discussing the correctness of the scheme.

**Theorem 5.4.5.** *The construction in Section 5.4.3 is correct according to definition 5.4.1.*

We will start the proof by checking that $\bar{R}_1 = R_1$, $\bar{R}_2 = R_2$, and $\bar{R}_3 = R_3$. If they equalities hold then $c = H(M, r, C_1, C_2, C_3, C_4, \bar{R}_1, \bar{R}_2, \bar{R}_3)$.

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

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

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

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

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

$$= \left(\frac{e(C_2,g_2)^{x_i+\gamma}}{e(v,g_2)^{(\gamma+x_i)\xi} e(g_1,g_2)}\right)^c (R_2)$$

$$= \left(\frac{e(g_1^{1/(x_i+\gamma)} v^\xi, g_2)^{x_i+\gamma}}{e(v,g_2)^{(\gamma+x_i)\xi} e(g_1,g_2)}\right)^c (R_2) = R_2$$

The second step is to prove the attribute authentication is correct too. In other words we need to justify the equality of $F_{root}^{1/\alpha} C_3 = e(C_2, C_4)$.
Recall $F_{root} = e(A_i^\beta, w)^\alpha$ (Section 5.2). The prove is as follows:
$F_{root}^{1/\alpha} C_3 = e(A_i^\beta, w) e(v^\xi, w)^\beta = e(A_i v^\xi, w)^\beta = e(C_2, C_4)$
The final step in proving correctness is to show that the scheme opens the signature correctly. This is done by proving the equality $e(C_2/A^*, \bar{u}) = e(C_1, \bar{v})$ holds when $A^* = A_i$ as shown below:
$e(C_2/A^*, \bar{u}) = e(C_2/A_i, \bar{u}) = e(v^\xi, \bar{u}) = e(u^\xi, \bar{v}) = e(C_1, \bar{v})$.
After proving correctness of the scheme we opt to prove the scheme is fully anonymous, fully traceable and attribute-unforgeable. For the comprehensive proof see Appendix A.1.2, A.2.2 and A.3.1. In this section we give a general idea on of the proof.

**Theorem 5.4.6.** *If the decision linear assumption holds in group $G_2$ then the AAS is anonymous under the random oracle assumption.*

Let *Adam* be an adversary that is attacking the AAS scheme's anonymity. *Eve* is the challenger in the anonymity game, she is also an adversary that is trying to attack the decision linear assumption. *Charles* is the challenger of *Eve*. *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 $a, b \in \mathbb{Z}_p^*$. *Eve*'s challenge is to decide whether $Z$ is random or $Z = u_2^{a+b}$. A simulation run of the adversarial anonymity game takes place between *Eve* and *Adam*. In the setup of the game, *Eve* creates the system parameters $S_{pub}$ and $S_{pri}$. She creates $n-2$ private key bases $bsk[i]$. The missing two will be assumed to be $A_{i_0} = ZW/u_2^a$ and $A_{i_1} = Wu_2^b$ for a random $W \in G_2$. If $Z = u_2^{a+b}$ then $A_{i_0} = A_{i_1}$. Notice that *Eve* does not know $bsk[i_0]$ nor

$bsk[i_1]$ but she will pretend she does if they are ever queried by the Signature oracle. *Eve* will create the signature as follows:

- If $i_0$ is queried do the following: *Eve* picks a random $s, t, l, \beta \in \mathbb{Z}_p^*$ and makes the following assignments:
  $C_1 = h_0 u_0^s$; $C_2 = ZW u_2^s h_0^t u_0^{st}$; $\bar{u} = u_0^l$; $\bar{v} = (u_2 u_0^t)^l$.
  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:*

$$\textit{Fake-Sign}_{Node} = \begin{cases} \textit{If } (j \in \Gamma)\textit{; return } e((u_2^s h_0^t u_0^{st})^{t_j}, D_j)^\beta \\ \qquad\qquad\qquad\qquad = e(u_2^s h_0^t u_0^{st}, w)^{\beta q_j(0)} \\ \textit{Otherwise return } \perp \end{cases}$$

  *$F_{root}$ in this case will equal $e(u_2^s h_0^t u_0^{st}, w)^{\beta\alpha}$. Notice that $F_{root}^{1/\alpha}.C_3 = e(C_2, w)^\beta$. If $\beta_1, \beta_2$ 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 $\langle e(ZW, w)^{\beta_2}, w^{\beta_2}, e(u_2^s h_0^t u_0^{st}, w)^{\beta_2\alpha}\rangle$.*

- *If $i_1$ is queried do the following: 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:*

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

  *$F_{root}$ 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 $\beta_1, \beta_2$ are random elements in $\mathbb{Z}_p^*$, it is hard to distinguish between the triples: $\langle e(v^\xi, w)^{\beta_1}, w^{\beta_1}, e(A_i, w)^{\beta_1\alpha}\rangle$ 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 $r, c, s_\xi, s_x, s_\delta \in_R \mathbb{Z}_p^*$. *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
$R_2 = e(\psi(C_2), g_2)^{s_x} e(\psi(\bar{v}), w)^{-s_\xi} e(\psi(\bar{v}), g_2)^{-s_\delta} \left(e(\psi(C_2), w)/e(g_1, g_2)\right)^c$.
Note that it is hard to distinguish between a valid signature and the signatures created by *Eve* for $i_0$ and $i_1$. If these elements are queried in other oracles *Eve* may abort. In the appendix we show how that has a low probability of happening.
Assume *Adam* can break the anonymity of the scheme. This assumption implies *Eve* always guesses $b$ correctly when $Z$ is not random because a valid signature is simulated.

Otherwise, *Eve* has a probability of 1/2 to guess $b$. Therefore, one would expect that if *Adam* has an advantage in breaking the anonymity of the scheme *Eve* will have an advantage in breaking the linear challenge. The appendix A.1.2 has more details of the proof.
Following the proof of anonymity we have to prove full traceability:

**Theorem 5.4.7.** *If q-SDH is hard on groups $G_1$ and $G_2$ then the AAS is fully traceable under the random oracle where $q$ is related to the number of users $n$.*

For proving full traceability we use the Forking Lemma (See Definition 3.4.1) as done for proving the full traceability of the ABGS (See Section 5.3.5).
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 $\sigma_0$, 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 will run the game in Section 5.4.1 twice. In both simulated runs, *Charles* is given an $(n)$ SDH instance, $(\grave{g}_1, \grave{g}_2, \grave{g}_2^{\gamma}, \grave{g}_2^{\gamma^2},...,\grave{g}_2^{\gamma^n})$. By applying the Boneh–Boyen theorem, *Charles* can obtain $g_1 \in G_1, g_2 \in G_2, w = g_2^{\gamma}$ and $(n-1)$ SDH pairs $(A_i, x_i)$ which he will use as the private key bases $bsk[i]$.
We run the adversarial traceability game twice with difference between the two runs being the response to the hash oracle (See Appendix A.2.2). According to the Forking Lemma if *Adam* can find a valid signature $\langle M, \sigma_0, c, \sigma_1, \sigma_2\rangle$ in the first run with a non-negligible probability, then it can create $\langle M, \sigma_0, \grave{c}, \grave{\sigma}_1, \sigma_2\rangle$ in the second run with the same random elements but a different hash oracle. We will use the signature $\langle M, \sigma_0, c, \sigma_1, \sigma_2\rangle$ resulting from the first round and signature $\langle M, \sigma_0, \grave{c}, \grave{\sigma}_1, \sigma_2\rangle$ outputted in the second to create a new SDH pair. Recall from the Forking Lemma that $c \neq \grave{c}$.
Let $\Delta c = c - \tilde{c}$, and $\Delta s_\xi = s_\xi - \tilde{s}_\xi$, and similarly for $\Delta s_x$, and $\Delta s_\delta$.
Divide two instances of the equations used previously in proving correctness of the scheme. 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 –Boyen theorem (See

Theorem 2.2.13).
. We should prove the ABGS scheme attribute unforgeable.

**Theorem 5.4.8.** *Breaking the Unforgeability of Attributes in the AAS construction is as hard as solving the DLP.*

Our approach in proving that is very similar to the proof in Section 5.2. We run the adversarial game model and then list all possible information *Adam* may obtain. We prove that with such a list *Adam* can not create the element $F_{root}$ in the signature unless the DLP is broken. Appendix A.3.1 explains the details of the proof.
Finally, we need to analyze efficiency of the AAS scheme. The main highlight of the efficiency of the construction is the signature's size. Unlike the ABGS where the signature was linearly dependent on the number of attributes, the AAS construction is of a constant size. Compared to Boneh and Shacham's group signature scheme the signature size in the AAS scheme has increased with three elements only and they are $F_{root}, C_3$, and $C_4$.

### 5.4.6 Attribute Exchange Protocols

Previously, in the game models of full anonymity and traceability, we assumed the attribute authority is dishonest by giving *Adam* the privilege of creating the master keys in the initialization stage of the games. However, in real life, we need the signer to be able to verify that the attribute private keys he obtains are valid and will help in signing. Furthermore, he needs to verify that the attribute public key is accepted by everyone in the system as valid verification keys.. In this section we add two further protocols which will help establish that. We start with defining what we mean by "Honestly Generated Attribute Keys".

**Definition 5.4.9.** (Honestly Generated Attribute Public Keys): *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.*

**Definition 5.4.10.** (Honestly Generated Attribute Private Keys): *The private attribute key is "generated honestly" if the member can verify its correctness with an honestly generated public attribute key and the members' registration key.*

To improve our scheme we add two protocols the APK and the ASK. We start with defining these protocols and then we explain how these protocols work with our construction in Section 5.4.3. The protocols are defined as follows:

- **APK**$(\mathcal{CA} : \mathcal{AA}_{\mathbf{j}})$: This protocol runs between the attribute authority $\mathcal{AA}_j$ and the central authority $\mathcal{CA}$. It takes no input since all calculations are done by choosing random elements. At the end of the protocol the central authority should obtain $bpk_j$ and the attribute authority should obtain $t_j$.

- **ASK**$(\mathcal{AA}_\mathbf{j}(\mathbf{t_j}) : \mathcal{U}_\mathbf{i}(\mathbf{gsk}))$: This protocol runs between the attribute authority $\mathcal{AA}_j$ and the user $\mathcal{U}_i$. The inputs are the master key of the attribute $t_j$ and the users private key $gsk$. At the end of the protocol the attribute authority should have authenticated the user and the user should get $T_{i,j}$ without revealing the registration key $A_i$ to $\mathcal{AA}_j$.

**Attribute Public Key Exchange Protocol (APK)**: This protocol is between the central authority $\mathcal{CA}$ and attribute authority $\mathcal{AA}_j$. 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 6-move key generation protocol adopted from Groth's work in [70]. The procedure is as follows:

1. $\mathcal{AA}_j$ picks a random $a, b \in \mathbb{Z}_p^*$ and $c \in \mathbb{Z}_p^*$. $\mathcal{AA}_j$ 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 $\mathcal{AA}_j$
3. $\mathcal{AA}_j$ picks $f \in \mathbb{Z}_p^*$ and sends it to $\mathcal{CA}$.
4. $\mathcal{CA}$ sends $e, d$ to $\mathcal{AA}_j$.
5. $\mathcal{AA}_j$ 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 $\mathcal{AA}_j$ but it guarantees honesty when generating the master key. We will 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 $\mathcal{AA}_j$ but without revealing the key.

**Attribute Private Key Exchange Protocol (ASK)**: This protocol is between the attribute authority $\mathcal{AA}_j$ and a member of the group $\mathcal{U}_i$. The purpose behind the protocol is 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. The 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 $\mathcal{AA}_j$ run the protocol $Verifign(\mathcal{U}_i(gsk, M), \mathcal{AA}_j(M, \overline{B}))$ where the value of $M$ is not significant to the protocol and can be any random message.

3. From the $Verifign$ protocol $\mathcal{AA}_j$ obtains the signature, $\sigma = (r, C_1, C_2, C_3, C_4, c, s_\xi, s_x, s_\delta)$.

4. $\mathcal{AA}_j$ attempts to verify the signature. If it is valid $\mathcal{AA}_j$ sends $E = C_2^{1/t_j}$ and $F = v^{1/t_j}$ back. Note that $v$ is known to $\mathcal{AA}_j$ because $r$ is known (See Section 5.4.3).

5. $\mathcal{U}_i$ calculates his attribute private key $T_{i,j} = E/F^\xi = C_2^{1/t_j}/v^{\xi/t_j} = (A_i^{1/t_j} v^{\xi/t_j})/v^{\xi/t_j} = A_i^{1/t_j}$.

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 5.4.9 and 5.4.10 holds.

**Claim 5.4.11.** *The ASK protocol ensures honesty of the attribute private key generation as defined in 5.4.10 and APK protocols ensures honestly of attribute public key generation as defined 5.4.9.*

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 $\mathcal{AA}_j$ and $\mathcal{CA}$. The reader is referred to [70] 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 $\mathcal{AA}_j$. Assuming that the AAS scheme is fully traceable, and $\sigma$ obtained in the third step of the protocol ($Verifign$) verifies, then the pair $(C_2, v)$ must be created honestly. $\mathcal{AA}_j$ 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.

## 5.5 Phase III: Dynamic Attribute Based Authentication Schemes

A Dynamic Attribute based Authentication Scheme (DAAS) is an improved AAS scheme. In our previous design of an AAS scheme the attribute authorities were not

trusted and we had methods to judge their honesty. On the other hand, the central authority had a major control over the system. It created the private keys and traced the signatures in order to revoke anonymity. Given such powers central authority can impersonate the signers by using their private keys and therefore misuse the ability to issue keys. In this phase we separate the responsibility of tracing and issuing keys so that a different authority is in control of each job. We also enable the member to choose part of his private key so that only he can sign but if he misbehaves his signature is still traceable.

In a DAAS system we have six entities a signer, a verifier, an attribute authority,

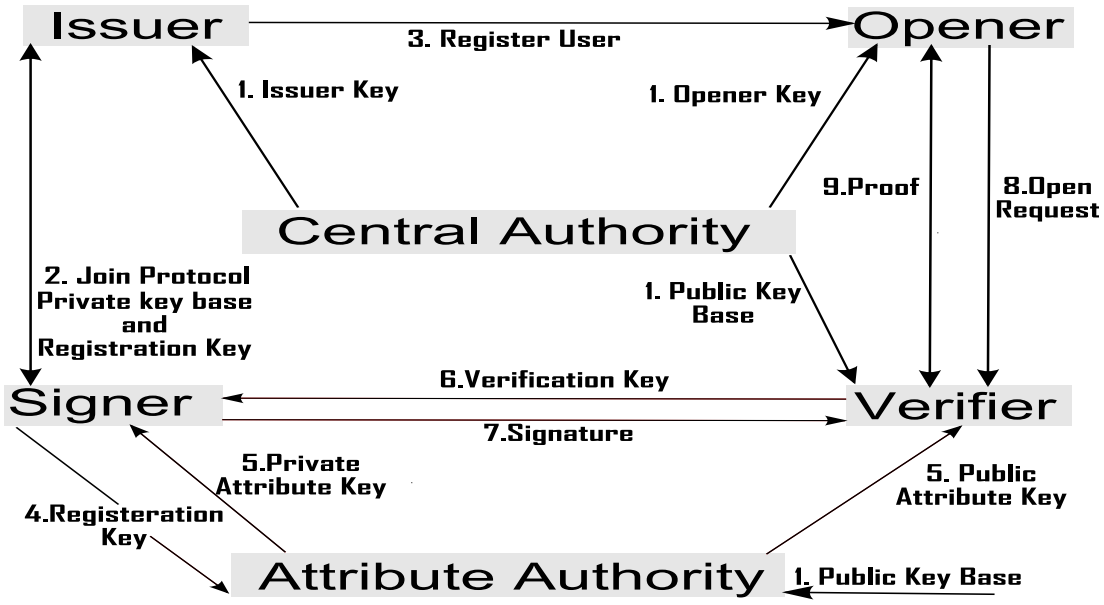


Figure 5-5: Dynamic Attribute based Authentication Scheme

a central authority, an open manager and an issuer manager. The central authority creates a public key base known to all other entities, an issuer key and a tracing key. The issuer key is given to the issuer manager and is used in creating signing keys (Step 1a in Figure 5-5). The tracing key is given to the open manager and is used in tracing signatures (Step 1b in Figure 5-5). The public key base is used by different entities for different reasons. Attribute authorities use it to create an attribute public key (Step 1c in Figure 5-5) while verifiers use it in the verification process (Step 1d in Figure 5-5). A join protocol is executed between users and the issuer manager (Step 2 in Figure 5-5). During that protocol a private key base and a registration key is created for the user such that the issuer has enough information about the user to help the open manager to trace signatures. However, no one other than the user (possible signer) will know

the full private key. The issuer manager sends the registration key to the open manager to add the new member to the list of possible signers when tracing a signature (Step 3 in Figure 5-5). Now that the signer has his private key base, he is ready to register and obtain attribute private keys. He sends his registration key to the authority which gives him in return an attribute private key (Step 4 and 5a in Figure 5-5). The verifier collects all attribute public keys he needs and uses them together with the public key base to generate a verification key (Step 6 in Figure 5-5). That key is sent to all potential signers and any one with enough attribute private keys can sign (Step 7 in Figure 5-5). If the verifier doubts the signature he can ask the open manager to trace it (Step 8 in Figure 5-5). The open manager should be able to send a proof back to the verifier that a certain user did actually sign and that he as an open manager is not framing the wrong user (Step 9 in Figure 5-5). The verifier can verify the signature and decide whether he accepts or rejects it. All these steps do not need to be executed every time a signature is sent. The steps that are repeated constantly are Steps 6 and 7 in Figure 5-5.

### 5.5.1 Definition

The dynamic attribute based authentication scheme is a collection of algorithms and protocols that are executed by the entities (signer, verifier, central authority, attribute authority, open manager and issuer manager). We define the scheme by explaining these algorithms or protocols as follows:

- $\mathbf{D.KeyGen(k_1)}$ : This algorithm is run by the central authority. It uses the security parameter $k_1$ for generating three keys. The issuer key $isk$ that is given to the issuer manager. The tracing key $tk$ given to the open manager. The general public key $gpk$ known to all.

- $\mathbf{D.U.KeyGen(k_2)}$ : This algorithm is run by the user. Using a security parameter $k_2$ the user generates a user public key $upk[i]$ and private key $usk[i]$. The security parameters $k_1$ and $k_2$ are not logically connected, though it might be wise to choose them together. The decision of $k_1$ will affect the full anonymity and full traceability games defined later in Section 5.5.2. On the other hand, the parameter $k_2$ should be chosen so that the pair $(upk[i], usk[i])$ can be used in an unforgeable signature scheme. That signature scheme will be used in the $D.Join$ protocol.

- $\mathbf{D.Join(Iss(isk) : \mathcal{U}_i(upk[i], usk[i]))}$ : This is a protocol that is followed by the issuer manager $Iss$ and a user $\mathcal{U}_i$. The user $\mathcal{U}_i$ uses the keys generated in $D.U.KeyGen$ as inputs to the protocol and $Iss$ uses the issuer key $isk$ as an input. The result of the protocol is a private key base known to the user only

$bsk[i]$ and a registration key $A_i$ saved in a database accessible by the open manager.

- $\mathbf{D.A.KeyGen_{pub}(gpk, j)}$ : This algorithm is run by the attribute authority where using the public key $gpk$, an attribute public key $bpk_j$ is generated for attribute $j$.

- $\mathbf{D.A.KeyGen_{pri}(A_i, j)}$ : This algorithm is run by the attribute authority where an attribute private key $T_{i,j}$ is generated for user $i$ and attribute $j$ using the user's registration key. The user will use $gsk = (bsk[i], T_{i,1}, ..., T_{i,\mu})$ for signing.

- $\mathbf{D.Verifign(\mathcal{U}_i(gsk, M) : \mathcal{V}(M, \overline{B}, gpk))}$ : This protocol is engaged between the user $\mathcal{U}_i$ and verifier $\mathcal{V}$. The user uses his general secret key $gsk$ and a message $M$ as an input and the verifier uses the public key $gpk$ and a set of attribute public key he needs (i.e. $\overline{B} = (bpk_1, ..., bpk_\kappa)$). At the end of the protocol the user sends a signature $\sigma$, created with an algorithm $\sigma = Sign(M, \overline{D}, gsk)$, to the verifier that proves possession of attributes. The verifier can check validity of such a signature by using the $Verify$ algorithm which helps in deciding whether to accept or reject a signature such that $Verify(\overline{D}, gpk, M, \sigma) = \{Accept, Reject\}$.

- $\mathbf{D.Open(\sigma, tk)}$ : The open manager uses the tracing key $tk$ and the registration key database (updated in the join protocol) to trace a signature $\sigma$ to member $i$.

- $\mathbf{D.Judge(\mathcal{V}(td) : \mathcal{OM}(tk))}$ : This is a protocol between the verifier $\mathcal{V}$ and the open manager $\mathcal{OM}$ in order to prove that the open manager is not framing user $i$ and that the open algorithm does in fact trace to him. The verifier calculates a trapdoor $td$ from the signature $\sigma$ and sends it to the manager. The open manager sends a proof $\mathcal{P}$ that user $i$ is the one being traced and that proof uses the trapdoor $td$ to verify.

### 5.5.2 Security Notions

Given the fact that DAAS schemes are an extension to AAS then the correctness definition does not change much. The following is the definition of DAAS correctness:

**Definition 5.5.1.** (Correctness of an DAAS scheme): *For all keys gsk generated correctly, every signature generated by a member i verifies as valid unless the member did not have enough attributes. Every valid signature should trace to a member of the group. In other words,*
$Verify(\overline{D}_v, gpk, M, Sign(M, \overline{D}_s, gsk)) = valid;$
*where* $\overline{D}_s = \overline{D}_v$ *and*
*if* $Verify(...) = valid$ *then* $Open(Sign(M, \overline{D}_s, gsk), tk) = i;$
*Otherwise* $Verify(...) = not\ valid;$

It is natural to assume that the security notions full traceability, full anonymity and attribute unforgeability are required in a DAAS scheme. The definitions are similar to the ones in Section 5.4.2. However, the oracle Revoke is replaced with Open Oracle. Two new oracles are added to include the join protocol and we will refer to them as the CrptJoinUsr and CrptJoinIss. Before we define the adversarial models we recall the oracles needed and add to them the definition of the new oracles.

- **USK Oracle**: To query this oracle the adversary sends the challenger an index $i$ in order to find the user's private key base $bsk[i]$, registration key $A_i$ and the user's secret key $usk[i]$. The challenger will send the triple $(bsk[i], A_i, usk[i])$ to the adversary.

- **Signature Oracle**: To query this oracle the adversary sends the challenger a verification key he created $\overline{D}$, a message $M$ and an index $i$. The challenger sends $\sigma = Sign(M, \overline{D}, gsk)$.

- **Open Oracle**: To query this oracle the adversary sends the challenger a signature $\sigma$. The challenger replies with an index $i$ of the signer and can execute the $Judge$ protocol with that reply.

- **CrptJoinUsr Oracle**: In this oracle the adversary engages with the challenger in a join protocol where the adversary plays the role of the user and the challenger is the issuer manager.

- **CrptJoinIss Oracle**: In this oracle the adversary engages with the challenger in a join protocol where the adversary plays the role of the issuer and the challenger is the user.

- **AttPriKey Oracle**: To query this oracle the adversary sends a registration key $A_i$ to the challenger together with an index of the attribute $j$. The output of this query is a private attribute key $T_{i,j}$.

- **AttMasKey Oracle**: To query this oracle the adversary sends an index $j$ to the challenger. The challenger responds with sending the output $t_j$.

Similar to the adversarial models in Section 5.4.2, both game models full traceability and full anonymity start with $Adam$ choosing the master keys of the universal set of attributes $U = \{t_1,...,t_m\}$. We will first define the full anonymity game.

**DAAS Full Anonymity:** We say that an DAAS 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 $A.DAAS$ game:

- **A.DAAS.Setup:** $Charles$ sets up the system. He creates a tracing key $tk$, issuer key $isk$ and a general public key $gpk$. He gives the $gpk$ and issuing key $isk$ to the adversary.

- **A.DAAS.Phase 1:** *Charles* will run the oracles, USK, a Signature oracle, CrptJoinUsr, CrptJoinIss and Open oracle.
- **A.DAAS.Challenge:** *Adam* asks to be challenged on a message $M$, two indexes $i_0$,$i_1$, and verification key $\overline{D}$. *Charles* responds back with a signature $\sigma_b$, 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 CrptJoinUsr oracle, however it can be queried in the Signature oracle.
- **A.DAAS.Phase 2:** This stage is similar to Phase 1. The signature $\sigma_b$ is not queried in the open algorithm.
- **A.DAAS.Output:** *Adam* outputs a guess $\bar{b} \in \{0, 1\}$. If $\bar{b} = b$, *Adam* wins the game.

In this game we assume the attribute authority is totally corrupted since *Adam* decides the master keys of the attributes (i.e. $t_j$). We also assume that the issuer is corrupted since *Adam* gets the issuer key. The open manager can not be corrupted because if the tracing key is obtained by *Adam* then anonymity is impossible since he can run the open algorithm. We did allow the Signature oracle to be queried for the indexes of the challenge in both phase 1 and 2. This implies that unlinkability is taken into consideration in this game. We assumed in the challenge that CrptJoinUsr is not used to query any of $i_0$ and $i_1$. This is a reasonable assumption since the challenger does not know the private key of the users and can not create signatures. In phase 2, we limited the open oracle so that the signature is not queried in the open oracle. This is another reasonable assumption since if you trace it to a signer with the open oracle the challenge is meaningless because the adversary did not guess the signer but was told who he is. Let the advantage $Adv_{A.DAAS}(k_1) = Pr[b = \bar{b}] - 1/2$. Then the definition of full anonymity is:

**Definition 5.5.2.** Full Anonymity*:*
*A scheme is fully anonymous if for all polynomial time adversaries Adam, $Adv_{A.DAAS}(k_1) < \varepsilon$ and $\varepsilon$ is negligible.*

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

- **T.DAAS.Setup:** *Charles* sets up the system and generates the tracing key $tk$, issuer key $isk$ and the general public key $gpk$. He sends *Adam* the tracing key $tk$ and the general public key $gpk$. Both *Charles* and *Adam* can calculate attribute public keys since they both have access to the master keys of all attribute.

- **T.DAAS.Queries:** *Charles* runs oracles: USK oracle, Signature oracle, and CrptJoinUsr. *Adam* queries them as described earlier.

- **T.DAAS.Output:** *Adam* asks to be challenged on a $M$ which he sends to *Charles*. *Charles* calculates a new $\overline{D}$ and sends it back to *Adam*. *Adam* replies with a signature $\sigma$. *Charles* verifies the signature. If it is not valid return 0. If it turns out 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.

In the game above we assume that the open manager is totally corrupted by giving *Adam* the tracing key. The attribute authorities are also corrupted since everyone knows the master keys of all attributes. The issuing key is kept a secret from the adversary, otherwise he can easily issue a private key that can not trace to a member of the group. *Adam* does not need to query the open oracle since that is redundant with the fact that he has the tracing keys. If we refer to this game as $Exp$ then the advantage is $Adv_{T.DAAS}(k) = Pr[Exp = 1]$ and definition of traceability is as follows:

**Definition 5.5.3.** Full Traceability:
*A scheme is fully traceable if for all polynomial time adversaries Adam, $Adv_{T.DAAS}(k_1) < \varepsilon$ and $\varepsilon$ is negligible.*

**DAAS Unforgeability of Attributes:** We say that our DAAS scheme is Attribute-Unforgable if no polynomially bounded adversary *Adam* has a non-negligible advantage against the challenger *Charles* in the following AFDAAS game:

- **AFDAAS.Setup:** *Charles* sets up the system. He generates the tracing key $tk$, the issuer key $isk$, and the general public key $gpk$. *Charles* plays the role of all attribute authorities in the system. He creates the universe of attributes by choosing a list of master keys $t_1$,...,$t_m$. *Charles* calculates the attribute public keys $bpk_1$,...,$bpk_m$ which he sends together with $tk$ and $gpk$ to *Adam*. *Charles* keeps to himself $isk$ and list of $t_j$.

- **AFDAAS.Phase (1):** *Charles* runs the oracles USK, Signature, CrptJoinUsr, AttPriKey, and AttMasKey. *Adam* can query these oracles in order to obtain information that may help him break the scheme.

- **AFDAAS.Challenge:** *Adam* sends a tree $\Gamma_1$, user $l$ and attribute $z$ which he would like to be challenged on. *Charles* replies with $\overline{D}$ for a tree $\Gamma_2$ where $\Gamma_2$ has two subtrees: the first is $\Gamma_1$ and the other is based on $t_z$. The threshold value of the root in $\Gamma_2$ is 2. The challenge condition is that user $l$ has not been queried in AttPriKey for the attribute $z$. Furthermore the challenged index $z$ should not have been queried in AttMasKey. These two conditions are reasonable as they contradict with the purpose of the game.

- **AFDAAS.Phase (2):** This phase is similar to Phase 1 as long as the challenge conditions are not broken.
- **AFDAAS.Output:** *Adam* outputs a signature $\sigma$ for the user $l$ on the verification key $\overline{D}$. If that signature is valid then the adversary wins and *Charles* outputs 1 otherwise *Adam* loses and *Charles* outputs 0.

*Adam* is given strong capabilities. We assume the open manager is totally corrupted since *Adam* is given the tracing key $tk$. *Adam* can also corrupt attribute authorities of his choice since he can query the AttMasKey. The only condition is that the attribute he asks to be challenged upon should not be corrupted as that contradicts with the purpose of the game model. *Adam* can also corrupt users by querying USK oracle or CrptJoinUsr. Furthermore, *Adam* can get attribute private keys for different users and different attributes as long as he does not query the user $l$ for the attribute $z$ of the challenge itself. This is a logical assumption because the idea of the game is to create a valid signature that proves that a user has an attribute when he does not really own it. If we refer to the output of this game as $Exp$ then the advantage of the game is notated as $Adv_{AFDAAS}(k_1) = Pr[Exp = 1]$.

**Definition 5.5.4.** Unforgeability of Attributes:
*A DAAS scheme is attribute-unforgeable if for all polynomial time adversaries, $Adv_{AFDAAS}(k_1) < \varepsilon$ and $\varepsilon$ is negligible.*

**Non-frameability** The last security notion we require in a DAAS is to prove non-frameability. We have to prove that given the tracing key, the issuing key, the general public key and all private keys of users, the adversary can not run the *Judge* protocol and prove that another user is the signer of a message.

- **F.DAAS.Setup:** *Charles* sets up the systems and gives the issuing key, tracing key, and general public key to *Adam*.
- **F.DAAS.Queries:** In this stage *Adam* can query the CrptJoinUsr, CrptJoinIss, Signature and the USK oracles.
- **F.DAAS.Challenge:** *Adam* chooses an index $i$, a signature $\sigma$, a verification key $\overline{D}$, and a message $M$. He sends them to *Charles*. Charles verifies the signature. If it is not valid return 0 else trace the signature. If it traces to a user not from the system then return 0. Otherwise *Charles* accepts the challenge on the registration key $A_i$ and lets *Adam* know. *Adam* and *Charles* have to engage in a judge protocol where *Adam* proves to *Charles* that $A^*$ is the signer. If $A^* = A_i$ then return 0 else return 1.

In non-frameability we can assume all authorities are corrupted. We did not need an Open oracle since the tracing keys are given to *Adam*. The rest of the oracles are there

since some private keys of honest users are generated and *Adam* may want to corrupt them. If this experiment is referred to as $Exp$ then the advantage of winning the game is defined as $Adv_{F.DAAS}(n,k) = Pr[Exp = 1]$.

**Definition 5.5.5.** Non-frameability:
*A scheme is non-frameable if for all polynomial time adversaries Adam, $Adv_{F.DAAS}(k_1) < \varepsilon$ and $\varepsilon$ is negligible.*

### 5.5.3 Construction of the DAAS

In this section we give an example of a DAAS construction that is an extension to the construction in Section 5.3.3. We use the prefix $DAAS$ in front of the algorithms to distinguish them from the ones in other sections. The algorithms are as follows:

- **DAAS.KeyGen($\mathbf{k_1}$)** : Using the security parameter $k_1$ a bilinear map $e : G_1 \times G_2 \rightarrow G_3$ is chosen where $G_1$, $G_2$ and $G_3$ are of prime order with a computable isomorphism between $G_1$ and $G_2$. Suppose further that the DDH problem is hard in $G_1$ and the q-SDH is hard to solve in $G_1$ and $G_2$ (See Definition 2.2.11 and Assumption 2.2.12). The scheme employs a hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$.
  Let $\xi_1$ and $\xi_2 \in_R \mathbb{Z}_p^*$. Generators $g_1$, $g_2$, $g_3$, and $g_4 \in G_1$ are selected such that $g_1 = g_4^{\xi_1}$ and $g_2 = g_4^{\xi_2}$. Choose $h \in_R G_2$ and assign $w = h^\gamma$ where $\gamma \in \mathbb{Z}_p^*$.
  The issuer key is $isk = \gamma$, the tracing key $tk = \{\xi_1, \xi_2\}$, and the general public key is $gpk = \langle e, G_1, G_2, G_3, H, g_1, g_2, g_3, g_4, h, w\rangle$.

- **DAAS.U.KeyGen($\mathbf{k_2}$)** : The user sets up any digital signature scheme using security parameter $k_2$ and generates a public key $upk[i]$ and a secret key $usk[i]$ for themselves.

- **DAAS.Join(Iss(isk) : $\mathcal{U}_\mathbf{i}$(upk[i], usk[i]))** : The protocol runs as follows:

  1. $Iss$ sends $g_1^\nu$ for some $\nu \in_R \mathbb{Z}_p^*$.

  2. $\mathcal{U}_i$ picks a $y_i \in_R \mathbb{Z}_p^*$. Calculate $g_1^{\nu y_i}$, and send it to $Iss$. $\mathcal{U}_i$ stores $g_1^{y_i}$ for use in Step 4.

  3. $Iss$ can calculate $g_1^{y_i}$ and then chooses $x_i \in_R \mathbb{Z}_p^*$. He calculates $A_i = (g_1^{y_i} g_3)^{1/(x_i+\gamma)}$. $Iss$ sends $A_i g_1^{y_i}$ to $\mathcal{U}_i$.

  4. $\mathcal{U}_i$ calculates $A_i$ from this and $g_1^{y_i}$ that was stored in Step 2. $\mathcal{U}_i$ can then check $A_i \in G_1$, calculate $S = Sign(A_i, usk[i])$ and send $S$ to $Iss$.

5. $Iss$ checks $S$ with respect to $upk[i]$ and $A_i$ and saves $(upk[i], A_i, x_i, S)$ in a database. $Iss$ sends $x_i g_1^{y_i}$ to the user. $S$ here is used as a commitment to the private key parts chosen by the user.

6. $\mathcal{U}_i$ computes $x_i$ and verifies $A_i^{(x_i+\gamma)} = g_3 g_1^{y_i}$. The element $A_i$ will be considered as the registration key. The users basic secret key is $bsk[i] = \langle A_i, x_i, y_i \rangle$.

- $\mathbf{DAAS.A.KeyGen_{pub}(gpk, j)}$ : Attribute authority generates an attribute public key for each attribute $j$ it controls by calculating $bpk_j = w^{t_j}$ for an $t_j \in_R \mathbb{Z}_p^*$.

- $\mathbf{DAAS.A.KeyGen_{pri}(A_i, j)}$ : Generates an attribute private key for each user $i$ that owns $j$. The attribute private key is $T_{i,j} = A_i^{1/t_j}$. The user general secret key is $gsk = \langle bsk[i], T_{i,1},...,T_{i,\mu} \rangle$.

- $\mathbf{DAAS.Verifign(\mathcal{U}_i(gsk, M) : \mathcal{V}(M, \overline{B}))}$ : The protocol runs as follows:

1. $\mathcal{V}$ has a set of public attribute keys $\overline{B} = (bpk_1,...,bpk_\kappa)$. He chooses $\alpha \in_R \mathbb{Z}_p^*$ and decides on a tree $\Gamma$. The verifier can now calculate $\overline{D} = TCreate(\Gamma, \alpha, \overline{B}) = \{D_1,...,D_\kappa\}$.

2. In this stage $\mathcal{U}_i$ runs the algorithm $\sigma = Sign(M, \overline{D}, gsk)$:
He chooses $\beta_1, \beta_2 \in_R \mathbb{Z}_p^*$. Sets $\beta = \beta_1\beta_2$. Then calculates $F_{root} = TVerify(\overline{D}, \Im_i, \bar{T})$ where $\bar{T} = \{T_{i,1}^{\beta},...,T_{i,\kappa}^{\beta}\}$.

   Let $\zeta, \delta, r_\zeta, r_\delta, r_x, r_z \in_R \mathbb{Z}_p^*$.

   Calculate $C_1 = g_4^{\zeta}$, $C_2 = A_i g_1^{\zeta}$, $C_3 = g_4^{\delta}$, $C_4 = A_i g_2^{\delta}$, $C_5 = e(g_2^{\delta} A_i^{1-\beta_2}, w)^{\beta_1}$ and $C_6 = w^{\beta_1}$.

   Then signer calculates
   $R_1 = g_4$,
   $R_3 = g_4^{r_\delta}$,
   $R_4 = g_1^{r_\zeta} g_2^{-r_\delta}$
   $R_2 = e(C_2, h)^{r_x} e(g_1, w)^{-r_\zeta} e(g_1, h)^{-r_z}$.

   Compute $c = H(M, C_1, C_2, C_3, C_4, C_5, C_6, R_1, R_2, R_3, R_4)$.
   Let $s_\zeta = r_\zeta + c\zeta$, $s_\delta = r_\delta + c\delta$, $s_x = r_x + cx_i$ and $s_z = r_z + cz$ where $z = x_i\zeta + y$.

$\mathcal{U}_i$ sends signature $\sigma = (C_1,\ C_2,\ C_3,\ C_4,\ C_5,\ C_6,\ F_{root},\ c,\ s_\zeta,\ s_\delta,\ s_x,\ s_z)$.

3. $\mathcal{V}$ runs the algorithm $Verify(M, \sigma, \overline{D}, gpk)$ as shown below:
   He verifies the signature by first checking the attributes. That is done by checking that $F_{root}^{1/\alpha} C_5 = e(C_4, C_6)$. If that is false the signature is rejected else the $\mathcal{V}$ runs a verification algorithm as described below:
   The verifier starts with deriving the elements
   $\bar{R}_1 = g_4^{s_\zeta} C_1^{-c}$,
   $\bar{R}_3 = g_4^{s_\delta} C_3^{-c}$,
   $\bar{R}_4 = g_1^{s_\zeta} g_2^{-s_\delta} / (C_2 C_4^{-1})^c$
   $\bar{R}_2 = e(C_2, h)^{s_x} e(g_1, w)^{-s_\zeta} e(g_1, h)^{-s_z} (\frac{e(C_2,w)}{e(g_3,h)})^c$.

   Check if $c = H(M,\ C_1,\ C_2,\ C_3,\ C_4,\ C_5,\ C_6,\ \bar{R}_1,\ \bar{R}_2,\ \bar{R}_3,\ \bar{R}_4)$; if it is not equal then reject the signature.

- **DAAS.Open**$(\sigma, \alpha)$ : Open manager first verifies the signature using $\alpha$. He then starts the tracing procedure using the tracing key $tk = \{\xi_1, \xi_2\}$. He compares the $(A)$'s saved for every member in his database with $A_i$. The way he does such a comparison is by deriving $A_i = (C_2/(C_1)^{\xi_1}) = (C_4/(C_3)^{\xi_2})$.

- **DAAS.Judge**$(\mathcal{V}(\mathbf{td}, \sigma) : \mathcal{OM}(\mathbf{tk}))$ : The signature $S$ on $A_i$ is used as the proof that open manager is not framing user $i$. To prove that $A_i$ was used in the signature being opened $\sigma$ a zero knowledge proof is used as follows:

  1. $\mathcal{V}$ picks a random $rnd \in \mathbb{Z}_p^*$ and sends the pair $td = (C_1^{rnd}, C_2^{rnd})$ to the open manager.
  2. Open manager calculates $\mathcal{P} = (C_2^{rnd}/C_1^{rnd\xi_1}) = A_i^{rnd}$. Note that the open manager does not know $rnd$. He sends the proof $\mathcal{P} = A_i^{rnd}$ to the verifier.
  3. $\mathcal{V}$ can calculate $A_i$ from $\mathcal{P}$ since he knows $rnd$.

### 5.5.4 General Discussion of the DAAS scheme

In this section we give a general analysis of the design of the scheme and a more specific analysis of the construction. The scheme inherits the major advantages of the AAS scheme. If proven fully traceable and attribute-unforgeable then it is unforgeable, coalition resistant and traceable. If the scheme is proven fully anonymous then it is both anonymous and unlinkable. The scheme is more dynamic since users can join anytime. The separability of duties is a major advantage to the AAS scheme. Having more authorities each responsible for a specific duty removes the bottle neck from the central authority. It also implies extra security since in the game models we were able to corrupt an authority while testing the other. Finally, the member of the group is

no longer given a private key, instead he creates it jointly with the issuer manager. The attribute authority is still not trusted and we will show in the later section (See Section 5.5.6) how we can guarantee honesty of the key generation of attributes. One other main contribution to this scheme over the AAS scheme is non-frameability where the open manager can not claim anyone to have signed a signature unless they truly did.

### 5.5.5 Analysis of the Construction of the DAAS Scheme 5.5.3

The construction naturally contains all advantages we have mentioned in the general analysis. We shall show that it is correct and secure according to the definitions in Section 5.5.1.

**Theorem 5.5.6.** *The construction in Section 5.5.3 is correct according to the Definition 5.5.1.*

To prove correctness we start with showing that $\bar{R}_1 = R_1$, $\bar{R}_2 = R_2$, $\bar{R}_3 = R_3$, and $\bar{R}_4 = R_4$. The equalities hold as shown:

$$\bar{R}_1 = g_4^{s_\zeta} C_1^{-c} = g_4^{r_\zeta + c\zeta} g_4^{-c\zeta} = g_4^{r_\zeta} = R_1$$

$$\bar{R}_3 = g_4^{s_\delta} C_3^{-c} = g_4^{r_\delta + c\delta} g_4^{-c\delta} = g_4^{r_\delta} = R_3$$

$$\bar{R}_4 = g_1^{s_\zeta} g_2^{-s_\delta} / (C_2 C_4^{-1})^c = g_1^{(r_\zeta + c\zeta)} g_2^{-(r_\delta + c\delta)} / ((A_i g_1^\zeta)(A_i g_2^\delta)^{-1})^c = g_1^{r_\zeta} g_2^{-r_\delta}$$

$$\bar{R}_2 = e(C_2, h)^{s_x} e(g_1, w)^{-s_\zeta} e(g_1, h)^{-s_z} \left(\frac{e(C_2,w)}{e(g_3,h)}\right)^c$$

$$= e(C_2, h)^{r_x + cx_i} e(g_1, w)^{-(r_\zeta + c\zeta)} e(g_1, h)^{-(r_z + cz)} \left(\frac{e(C_2,w)}{e(g_3,h)}\right)^c$$

$$= R_2 (e(A_i, h)^{c_x} e(g_1^\zeta, h)^{cx_i} e(g_1, w)^{-c\zeta} e(g_1, h)^{-c(x\zeta + y)}) \left(\frac{e(C_2,w)}{e(g_3,h)}\right)^c$$

$$= R_2 (e(A_i, h)^{cx_i} e(A_i, h)^{cx_i} e(g_1, w)^{-c\zeta} e(g_1, h)^{-cy}) \left(\frac{e(C_2,w)}{e(g_3,h)}\right)^c$$

$$= R_2 \left(\frac{e(A_i,h)^{x_i} e(C_2,w)}{e(g_1,w)^\zeta e(g_1,h)^y} e(g_3, h)\right)^c$$

$$= R_2 \left(\frac{e(A_i,h)^{x_i} e(A_i,w)}{e(g_1,h) e(g_3,h)}\right)^c$$

$$= R_2 \left(\frac{e(A_i,h)^{x_i + \gamma}}{e(g_1 g_3,h)}\right)^c = R_2$$

The following step is to prove correctness of the attribute verification and the open algorithm. Given that the $F_{root} = e(A_i, w)^{\alpha\beta}$ then

$F_{root}^{1/\alpha}C_5 = e(A_i, w)^{\beta_1\beta_2}e(g_2^{\delta}A_i^{1-\beta_2}, w)^{\beta_1} = e(A_i g_2^{\delta}, w)^{\beta_1} = e(C_4, C_6)$.
Proving the open algorithm is correct is by proving that $A_i = C_2/(C_1)^{\xi_1} = C_4/(C_3)^{\xi_2}$ is correct. So
$C_2/(C_1)^{\xi_1} = (A_i g_1^{\zeta})/g_1^{\zeta} = A_i$
$C_4/(C_3)^{\xi_2} = (A_i g_2^{\delta})/g_2^{\delta} = A_i$
Next step is to prove the systems to be fully anonymous.

**Theorem 5.5.7.** *If the Elgamal Encryption Scheme is IND-CPA secure then the DAAS scheme is fully anonymous under the random oracle.*

Assume the adversary *Adam* has the capability of breaking the scheme's anonymity. *Eve* is trying to break the IND-CPA security of Elgamal. She is given the Elgamal public key $(g_1, g_4) \in G_1^2$ where $g_1 = g_4^{\xi_1}$ and $\xi_1 \in \mathbb{Z}_p^*$. $\xi_1$ is kept secret to the challenger while the rest is public and known to *Eve*. *Eve* calculates $g_2 = g_1 g_4^{rnd_1}$ therefore $\xi_2 = \xi_1 + rnd_1$ where $rnd_1 \in \mathbb{Z}_p^*$. *Eve* chooses a $\gamma \in \mathbb{Z}_p^*$, $h \in G_2$ and $g_3 \in G_1$. *Eve* can calculate *gpk* and then start challenging *Adam*. The oracles of Phase (1) and (2) are explained in details in appendix A.1.3. In the challenge *Adam* sends $i_0$, $i_1$, $M$ and $\overline{D}$ to *Eve*. She sends $A_{i_0}$ and $A_{i_1}$ to *Charles* as messages she wants to be challenged on. *Charles* encrypts one of them and returns ciphertext $(C_1 = g_4^{\zeta}, C_2 = A_b g_1^{\zeta})$. Note that *Eve* has to guess $b$.
*Eve* simulates a signature by choosing a random $rnd_2 \in \mathbb{Z}_p^*$ and calculating $C_4 = C_2 C_1^{rnd_1} g_2^{rnd_2}$ and $C_3 = C_1 g_4^{rnd_2}$. Given that $\delta = \zeta + rnd_2$, then $C_4 = A_b g_2^{\delta}$ and $C_3 = g_4^{\delta}$. *Eve* chooses randomly $s_\zeta$, $s_\delta$, $s_x$, $s_z$ and $c$ from $\mathbb{Z}_p^*$. Note that $c$ should have not been a response to a query to the hash oracle. She calculates $R_1 = g_4^{s_\zeta} C_1^{-c}$, $R_3 = g_4^{s_\delta} C_3^{-c}$, $R_4 = g_1^{s_\zeta} g_2^{-s_\delta}/(C_2 C_4^{-1})^c$ and $R_2 = e(C_2, h)^{s_x} e(g_1, w)^{-s_\zeta} e(g_1, h)^{-s_z} (\frac{e(C_2,w)}{e(g_3,h)})^c$. Finally *Eve* creates $F_{root}$ with $T_{i,j} = (C_2 C_1^{rnd_1})^{1/t_j}$ therefore $F_{root} = e(C_2 C_1^{rnd_1}, w^{\beta})^{\alpha}$ for some random $\beta \in \mathbb{Z}_p^*$. $C_6 = w^{\beta}$ and $C_5 = e(g_2^{rnd_2}, w^{\beta})$.
*Adam* guesses $\bar{b}$ in which he sends to *Eve* and she sends it to *Charles* as her guess. Note that if *Adam* guesses right *Eve* does too.
After proving full anonymity we opt to prove full traceability.

**Definition 5.5.8.** *If the q-SDH is hard in group $G_1$ and $G_2$ then the DAAS scheme is fully traceable under the random oracle assumption.*

The method we use for that is similar to the one we used in the full traceability in AAS scheme. *Adam* is a forger. The signature format is similar to Section 5.4.5 and two simulated runs of the adversarial game take place where *Charles* is given an $(n)$ SDH instance, $(\grave{g}_1, \grave{g}_2, \grave{g}_2^{\gamma}, \grave{g}_2^{\gamma^2}, ..., \grave{g}_2^{\gamma^q})$. By applying the Boneh –Boyen theorem, *Charles* can obtain $g_1 \in G_1, g_2 \in G_2, w = g_2^{\gamma}$ and $(n-1)$ SDH pairs $(A_i, x_i)$ which he will use in creating the private key bases $bsk[i] = (A_i, x_i, y_i)$. In the Appendix A.2.3 we show details of how to respond to the oracles of the game.
The Forking Lemma is applied to obtain $\Delta c = c - \hat{c}$, and $\Delta s_\zeta = s_\zeta - \hat{s}_\zeta$, and similarly

for $\Delta s_x$, $\Delta s_\delta$, $\Delta s_x$ and $\Delta s_z$.
Divide two instances of the equations used previously in proving correctness of the scheme. One instance with $\hat{c}$ and the other with $c$ to obtain the following:

- Dividing $C_1^c/C_1^{\hat{c}} = g_4^{s_\zeta}/g_4^{\hat{s}_\zeta}$ we get $g_4^{\hat{\zeta}} = C_1$; where $\hat{\zeta} = \Delta s_\zeta/\Delta c$
- Dividing $C_2^{s_\delta}/C_2^{\hat{s}_\delta} = g_4^{s_\delta}/g_4^{\hat{s}_\delta}$ we get $g_4^{\hat{\delta}} = C_2$; where $\hat{\delta} = \Delta s_\delta/\Delta c$
- The division of $(C_2C_4^{-1})^{\Delta c} = g_1^{\Delta s_\zeta} g_4^{\Delta s_\delta}$ implies $(C_2C_4^{-1}) = g_1^{\hat{\zeta}} g_4^{\hat{\delta}}$
- The division of $e(C_2,h)^{\Delta s_x} e(g_1,w)^{-\Delta s_\zeta} e(g_1,h)^{-\Delta s_z} = (\frac{e(g_3,h)}{e(C_2,w)})^{\Delta c}$ leads to $e(C_2,h)^{\hat{x}} e(g_1,w)^{-\hat{\zeta}} e(g_1,h)^{-\hat{z}} = (\frac{e(g_3,h)}{e(C_2,w)})^{\Delta c}$ for $\hat{x} = \Delta s_x/\Delta c$ and $\hat{z} = \Delta s_z/\Delta c$

If $\hat{A} = C_2 g_1^{\hat{\zeta}}$ and $\hat{y} = \hat{z} - \hat{\zeta}\hat{x}$ then from the last division we get $e(\hat{A},h)^{\hat{x}} e(\hat{A},w) = e(g_3,h)e(g_1,h)^{\hat{y}}$ this implies we have obtained a certificate $(\hat{A},\hat{x},\hat{y})$ where $\hat{A} = (g_3 g_1^{\hat{y}})^{1/(\hat{x}+\gamma)}$.
This leads to a SDH pair $(\mathcal{A},\chi)$ and breaking Boneh –Boyen theorem (See Theorem 2.2.13). Knowing that $\hat{A} = (g_3 g_1^{\hat{y}})^{1/(\hat{x}+\gamma)} = (g_1^{\hat{y}+rnd_1})^{1/(\hat{x}+\gamma)}$.
Calculate $(\mathcal{A},\chi) = (\hat{A}^{1/(\hat{y}+rnd_1)}, x_i)$.
We now need to prove the scheme secure against attribute forgeability.

**Theorem 5.5.9.** *Breaking the Unforgeability of Attributes in the DAAS construction is as hard as solving the DLP.*

We use a similar technique to the proof in Section 5.2. In Appendix A.3.2 we show the details of the proof. We show how using the game model defined in 5.5.2, *Adam* can obtain a list of information. We then prove that creating a signature with the missing attribute relies on the ability to create a valid $F_{root}$. We further prove that the list of information *Adam* has will not help him in deriving $F_{root}$ unless *Adam* is capable of solving the DLP.
Finally we discuss Non-frameability. Given all secret keys *Adam* can not convince *Charles* that the signer is someone other than who he actually is.

**Theorem 5.5.10.** (Non-frameability) *Breaking the Non-frameability of the scheme is as hard as the Discrete Logarithm problem.*

The adversarial game is shown below:

- **F.DAAS.Setup:** *Charles* generates the keys *isk*, *tk*, and *gpk*. He gives all of them to *Adam*. Recall that *Adam* initiates all games by choosing the universal set of attributes' master keys $U = \{t_1,...,t_m\}$. Therefore both *Adam* and *Charles* can create attribute keys, create private keys, and trace signatures.

- **F.DAAS.Oracles:** The oracles responses are computed as done in the construction since all keys are known to both *Adam* and *Charles*. However, every time a new registration key $A_i$ is created (using oracles CrptJoinUsr, and CrptJoinIss) it is compared with the list of $A_i$ generated by the oracles earlier. This is to ensure that no two users have the same $y_i$. Assume $(A, x)$ is a pair in the list. The comparison is done by first deriving $A_i = C_2/(C_1)^{\xi_1}$. Then looking up the $x_i$ from the list of possible signers. Then compare
  $\frac{A_i^{x_i+\gamma}}{g_3} = \frac{A^{x+\gamma}}{g_3}$
  this equality implies that
  $g_1^{y_i} = g_1^{y}$
  If that holds for any element in the list then $y = y_i$ and *Charles* aborts. If it does not hold add the new $A_i$ to the list.

- **F.DAAS.Output:** *Adam* chooses an index $i$, a signature $\sigma$, a verification key $\overline{D}$, and a message $M$. He sends them to *Charles*. *Charles* verifies the signature and if it is not valid he aborts the game returning 0. He then traces it by calculating $A_i = C_2/(C_1)^{\xi_1} = C_4/(C_3)^{\xi_2}$. *Charles* accepts the challenge on $A_i$ and sends to *Adam* the values $C_2^{rd}$ and $C_1^{rd}$ for a random $rd \in \mathbb{Z}_p^*$. *Adam* has to reply with $(A^*)^{rd}$ where $A^* \neq A_i$ and $A^*$ is on the list of possible signers.

Assuming *Adam* wins the game and the discrete logarithm problem is hard then *Adam* must know a value $s \in \mathbb{Z}$ such that $A_i^s = A^*$ in order to calculate $(A^*)^{rd} = (C_2/(C_1)^{\xi_1})^{s.rd} = A_i^{s.rd}$. Given that we will try deriving the valid key $(A^*, y^*, x^*)$ according to $s$. Consider the following statements:
$A^* = A_i^s$
$A^* = (g_3 g_1^{y^*})^{1/(x^*+\gamma)}$
$A_i = (g_3 g_1^{y_i})^{1/(x_i+\gamma)}$
From these statements one can derive:
$A^* = A_i^s = (g_3 g_1^{y_i})^{s/(x_i+\gamma)} = (g_3 g_1^{y^*})^{1/(x^*+\gamma)}$
$(g_3^{s/(x_i+\gamma)} g_1^{y_i(s/(x_i+\gamma))}) = (g_3^{1/(x^*+\gamma)} g_1^{y^*/(x^*+\gamma)})$
We can conclude that
$s/(x_i+\gamma) = 1/(x^*+\gamma)$, therefore $x^* = ((x_i+\gamma)/s) - \gamma$
and
$y_i(s/(x_i+\gamma)) = y^*/(x^*+\gamma) = y^*(s/(x_i+\gamma))$ therefore $y_i = y^*$
Since we know that $y_i \neq y$ from the comparisons that took place in querying the oracles and that contradicts the equations we derived we can conclude that *Adam* can not have known an $s$ and if he wins the game then he must have broken the discrete logarithm problem to get the $rd$ element.

### 5.5.6 Attribute Exchange Protocols

In this section we want change the algorithms $A.KeyGen_{pri}$ and $A.KeyGen_{pub}$ to protocols just as done in Section 5.4.6. The APK exchange protocol is exactly the same as in Section 5.4.6. The ASK protocol differs slightly. The following is an explanation on how it differs.

**Attribute Private Key Exchange Protocol (ASK)**: This protocol is executed between the attribute authority and the user as shown

1. The 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 $\mathcal{AA}_j$ runs the protocol $Verifign(\mathcal{U}_i(gsk, M), \mathcal{AA}_j(M, \overline{B}))$.
3. From the $Verifign$ protocol $\mathcal{AA}_j$ obtains the signature, $\sigma = (C_1, C_2, C_3, C_4, C_5, C_6, F_{root}, c, s_\zeta, s_\delta, s_x, s_z)$.
4. $\mathcal{AA}_j$ attempts to verify the signature. If it is valid $\mathcal{AA}_j$ sends $E = C_2^{1/t_j}$ and $F = g_1^{1/t_j}$ back.
5. $\mathcal{U}_i$ calculates his attribute private key $T_{i,j} = E/F^\delta = A_i^{1/t_j}$.
6. $\mathcal{U}_i$ verifies $T_{i,j}$ is correct by checking $e(T_{i,j}, bpk_j) = e(A_i, w)$.

Recall from Section 5.4.6 that the attribute public key can not be changed or modified without the knowledge of the central authority in an AAS scheme. Since the protocol APK did not change at all in the DAAS scheme, the same holds for the attribute public key in the DAAS scheme. The attribute authority can not calculate elements $E$ and $F$ without the knowledge of the master key $t_j$. The user can verify that the $t_j$ used in creating the $T_{i,j}$ is the same as the one used in the attribute public key through the sixth step in the protocol. This implies honesty of the attribute authority when generating the attribute private key. Similar to the ASK protocol in 5.4.6 the attribute authority does not have a clue about the value of $A_i$ or $T_{i,j}$.

## 5.6 Chapter Summary

In this chapter we proposed three main schemes an attribute based group signature scheme (ABGS) an attribute based authentication scheme (AAS), and a dynamic attribute based authentication scheme (DAAS). In all schemes the verifier decides the characteristics of the signer by building an attribute tree. An attribute tree is a structure used to prove possession of a set of attributes. It represents a monotone boolean

expressions where inputs are attributes. In Section 5.2 we have defined such tree and its powerful properties.

Attribute based group signatures were proposed in Section 5.3. The idea behind them is to embed attributes in group signatures allowing the verifier to decide on a subgroup of signers in which he prefers the signer to belong from. The verifier decides such subgroup by building a desired attribute tree. In that section we define the scheme and its security notions. We give an example on the construction. We end the section by analyzing that example and the scheme in general.

Attribute based authentication scheme was explained in Section 5.4. After having provided the ABGS we realized some desired properties are missing from the scheme such as separability, and attribute anonymity. Therefore we introduced the AAS scheme. The scheme has stronger capabilities than the ABGS and is much more efficient. In that section we define the scheme and its security. We then give an example of a construction. We later end the section with analyzing the scheme and the example.

Dynamic attribute based authentication scheme is an extension to an AAS scheme (Section 5.5). The AAS scheme achieves all required properties but its efficiency and security can be improved by having an extra level of separability, a dynamic property and extra security notions (non-frameability). The dynamic property implies that users can join the group whenever they want. The extra level of separability is achieved by dividing the two tasks of the central authority (Issuing keys and tracing signatures) in the AAS scheme so that different authorities take control of each. Non-frameability allows the verifier to check that the open manager is not accusing a signer to have signed a certain message when revoking anonymity.

Sections 5.4.6 and 5.5.6 strengthens the AAS and DAAS scheme by adding two protocols. In cryptography the less trusted third parties needed the stronger the scheme. Therefore the attribute authorities are not to be trusted. Nevertheless, they play an important role in our system and a method for verifying their honesty should be provided. The protocols reinforced in our system enables such verification.

In the following chapter we are going to give a general construction for converting any static group signature scheme to an AAS scheme.

# Chapter 6

# General Construction

In this chapter we give a general construction of an AAS scheme from a static group signature scheme using the extra building block, the attribute tree structure defined in Section 5.2. In Section 6.3 we give a proof that our general construction creates an AAS scheme that is fully traceable and anonymous if the inputted static group signature is fully traceable and anonymous. Section 6.4 shows an example of a construction based on the general one in Section 6.2 that uses Bellare, Micciancio and Warinschi as the input static group signature scheme.

## 6.1 Introduction

A massive amount of research has been done on group signatures since Chaum and Heyst proposed them first. Cryptographers tried adding new features to it. They have also improved its efficiency and security. Our proposed AAS scheme is new in literature and to save a vast number of work for researchers we came up with a general construction of the AAS scheme from any static group signature scheme. Our general construction is a powerful tool to convert any existing static group signature system to an AAS scheme while maintaining security features and inheriting properties of the inputted group signature.

## 6.2 The General Construction

Creating an AAS scheme requires three stages. The first stage is finding a secure group signature scheme that will prove that the signer is a member of the group in a fully anonymous and traceable matter. The second stage is proving possession of attributes and that is using the tree in Section 5.2. The third stage is to prove that the signer of the group signature scheme is a signer satisfying the attributes. In this section we demonstrate stage three. We show how to use the group signature algorithms in Section 4.2.1 and the attribute tree structure in Section 5.2 to create the AAS

algorithms in Section 5.4.1. We should highlight the fact that an attribute tree as defined in Section 5.2 requires bilinear maps. To guarantee FAP security (Section 5.2), the bilinear map $e : G_1 \times G_2 \rightarrow G_3$ should be chosen over groups of prime order where the BDH (See Definition 2.2.9) is hard. In order to convert the elements of the group signature, which can be of any key space, into elements in the mathematical groups $G_1$ and $G_2$ we use hash functions. The hash function chosen should be collision resistant. The algorithms of the AAS scheme are constructed as follows:

- **GCS.Setup** : The central authority follows the steps below:
  1. Run *SGS.Setup* to produce the system parameters $S_{pub}$, and $S_{pri}$.
  2. Choose a hash function $H_1 : \{0,1\}^* \rightarrow G_1$ and $H_2 : \{0,1\}^* \rightarrow G_2$.
  3. Output ($S_{pri}$, $S_{pub}$, $H_1$, $H_2$), where $S_{pri}$ is kept private to the central authority and the rest can be accessed by anyone.
- **GCS.M.KeyGen**: The central authority creates the public key base $w$, $n$ private key bases $bsk[i]$ and registration keys $A_i$ as follows:
  1. Run *SGS.KeyGen* algorithm to get $n$ private keys, $bsk[i]$.
  2. Let the private key base of the AAS be $bsk[i]$. Let the registration key, $A_i = H_1(bsk[i])$.
  3. The central authority saves $H_2(bsk[i])$ in a database which we will refer to as $\mathcal{T}$ used for tracing purposes.
- **GCS.A.KeyGen$_{\mathbf{pub}}$**: The attribute authority chooses for attribute $j$, a master key $t_j \in_R \mathbb{Z}_p^*$ in order to create the attribute public key, $bpk_j = H_2(S_{pub})^{t_j}$. To guarantee honesty of attribute authority in generating the key, this algorithm is replaced with the protocol APK as explained in Section 5.4.6.
- **GCS.A.KeyGen$_{\mathbf{pri}}$** : An attribute private key for user $i$ will be $T_{i,j} = A_i^{1/t_j}$. The private key the member will use to sign with is $gsk = \langle bsk[i], T_{i,1}, ..., T_{i,j} \rangle$ where $j$ is the number of attributes he owns. As done in Section 5.4.6, this algorithm is replaced with the protocol ASK. The protocol in this case is different from the one in Section 5.4.6 and we will explain it later in this section.
- **GCS.SignVerify** : This stage is a Sign/Verify protocol (this corresponds to the $Verifign$ algorithm in Section 5.4) that runs between verifier $\mathcal{V}$ and member $\mathcal{U}_i$ as follows:
  1. $\mathcal{V}$ chooses a $\Gamma$ and collects all public key bases it needs $\bar{B} = (bpk_1, ..., bpk_\kappa)$. $\mathcal{V}$ chooses $\alpha \in_R \mathbb{Z}_p^*$. $\mathcal{V}$ calculates $w = H_2(S_{pub})$. $\mathcal{V}$ can calculate $\overline{D} =$

$(D_1, ..., D_\kappa) = TCreate(\Gamma, \alpha, \overline{B})$ as shown in Section 5.2 and sends the values together with $\Gamma$ to $\mathcal{U}_i$.

2. $\mathcal{U}_i$ in this stage runs the AAS signature algorithm by first calculating $GS\sigma = SGS.Sign(M, bsk[i], S_{pub})$. $\mathcal{U}_i$ picks $\beta \in_R \mathbb{Z}_p^*$ and calculates $\bar{T} = \{CT_1, CT_2, ..., CT_\tau\}$ where $CT_j = T_{i,j}^{\beta}$. These values enable $\mathcal{U}_i$ to run the algorithm $TVerify(\overline{D}, \Im_i, \bar{T}) = F_{root}$ as done in Section 5.2. $\mathcal{U}_i$ assigns $td = A_i^{\beta}$. $\mathcal{U}_i$ calculates $\mathcal{TK} = GS.Sign(F_{root}, bsk[i], S_{pub})$ and $\mathcal{L} = e(td, H_2(bsk[i]))$. The signature is $\sigma = (GS\sigma, \mathcal{TK}, F_{root}, td, \mathcal{L})$ and is sent to the verifier.

3. $\mathcal{V}$ in this stage runs the AAS verification algorithm. $\mathcal{V}$ starts with verifying $GS.\sigma$ using the algorithm $GS.Verify$. He can verify the attributes by checking $F_{root}^{1/\alpha} = e(td, w)$. $\mathcal{V}$ verifies the signature on $F_{root}$ whether it is valid or not. If all three hold then output accept signature otherwise reject it.

We shall point out that $\mathcal{L}$ and $\mathcal{TK}$ are meant for tracing a signature as shown later. The element $\mathcal{TK}$ will be used as a proof that verifier $\mathcal{V}$ used a random element $\alpha$ on the signature. This kind of proof will be useful in the opening algorithm.

- **GCS.Open** : First step is to verify the signature $\sigma$. Note that verifier can not manipulate $\alpha$ because it is inbuilt in $F_{root}$ which the signer agreed on by signing it in $\mathcal{TK}$. Second step is to run the $GS.Open$ on $GS.\sigma$ that will identify the member. Finally, check whether $\mathcal{L} = e(td, H_2(bsk[i]))$ of the user where $H_2(bsk[i])$ is retrieved from $\mathcal{T}$.

**Simple Conversion Method:** The purpose behind adding the hash functions $H_1, H_2$ and the element $\mathcal{L}$ in the construction above is to convert any private key and public key to generators that belong to groups $G_1$, and $G_2$. This enables using the bilinear maps and reserves the possibility of tracing to a user. In case the private key and/or the public key are already generators then there is no need to add all these elements (See example in Section 5.4). We can assume that our hash function $H_b(g) = g^{something}$, where $b \in \{1, 2\}$ and $g \in G_b$.

**Attribute Private Key Protocol (ASK):** Earlier we described the procedure of obtaining attribute private keys as the algorithm $A.KeyGen_{pri}(A_i, j)$. This can be replaced with a protocol that runs as follows:

1. The 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 $\mathcal{AA}_j$ run the protocol $Verifign(\mathcal{U}_i(gsk, M), \mathcal{AA}_j(M, \bar{B}))$.

3. From the $Verifign$ protocol $\mathcal{AA}_j$ obtains the signature $\sigma = (GS\sigma, \mathcal{TK}, F_{root}, td, \mathcal{L})$.

4. $\mathcal{AA}_j$ can verify the signature and if it is valid $\mathcal{AA}_j$ sends $td^{1/t_j}$ back.

5. $\mathcal{U}_i$ can calculate his attribute private key $T_{i,j} = (td^{1/t_j})^{1/\beta}$.

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 does not know $\beta$. This protocol and the APK protocol (explained in Section 5.4.6) will guarantee honesty in the generation of the attribute keys. The following section will prove full anonymity and traceability of the scheme.

## 6.3 General Construction Security Proofs

To prove full traceability and full anonymity of the general construction in Section 6.2, we introduce three entities. An adversary *Adam* who tries to break the security of the AAS scheme. *Charles* who will be the challenger of the adversarial game models of the group signature scheme. *Eve* who plays the role of the challenger when dealing with *Adam* and the role of the adversary when dealing with *Charles*. Assuming that *Adam* breaks the security of the AAS scheme, we will show how *Eve* would win the challenge against *Charles*. We shall start with the full anonymity of the general construction.

**Theorem 6.3.1.** Anonymity of AAS: *Given a group signature that is fully anonymous the attribute based authentication scheme created by the general construction is fully anonymous too.*

Consider the following game:

- **Init:** *Adam* chooses $(t_1,...,t_m) \in \mathbb{Z}_p^*$ as master keys.

- **Setup:** *Charles* runs *SGS.Setup* to produce the $S_{pub}$ and $S_{pri}$. *Charles* sends $S_{pub}$ to *Eve*. *Eve* adds two hash functions $H_1$ and $H_2$ to $S_{pub}$ and sends them to *Adam*.

- **Phase(1):** This stage consists of three oracles. In the USK oracle *Adam* sends *Eve* an index $i$. *Eve* sends that index to *Charles* as a query to the PriKey oracle (See Section 4.2.2). *Charles* replies with $bsk[i]$. *Eve* sends $A_i = H_1(bsk[i])$ of member $i$ together with $bsk[i]$ to *Adam*.
  In the Signature oracle, *Adam* can calculate a $\overline{D}$ since he has the master keys $(t_1,...,t_m)$. He sends $\overline{D}$, a message $M$ and an index $i$ to *Eve*. *Eve* sends $i$ to

*Charles* and queries the PriKey oracle. *Charles* responds back with $bsk[i]$. *Eve* calculates $A_i = H_1(bsk[i])$. She calculates $GS\sigma$. She has the master keys to calculate as many $T_{i,j}$ as she needs. She chooses a $\beta \in_R \mathbb{Z}_p^*$ which will help her calculate $(F_{root}, td)$ as shown in the general construction. *Eve* can send *Adam* $\sigma = (GS.\sigma, F_{root}, td)$.

Open oracle *Adam* can either send a signature he received in the Signature oracle or a signature he creates from one of the private keys he got from the private key oracle. Along with $\sigma$ he sends the message $M$ and the verification key $\overline{D}$. *Eve* sends $GS\sigma$ and $M$ to *Charles* accessing the Open oracle of the group signature anonymity game. *Charles* sends back an index. *Eve* sends that index to *Adam*.

- **Challenge:** *Adam* sends *Eve* two indexes $(i_0, i_1)$, a message $M$ and $\overline{D}$ as a challenge. *Eve* sends *Charles* $(i_0, i_1, M)$ as her challenge. *Charles* replies with $GS\sigma_b$. *Eve* can calculate $(F_{root}, td)$ by choosing a random $td \in G_1$. She then calculates $td^{1/t_j}$ for all attributes needed. Finally she can calculate $F_{root} = e(td, w)^{\alpha}$. *Adam* will not be able to tell it is an invalid pair since he can not distinguish between $td = H_1(bsk[i])^{\beta}$ and a random generator. *Eve* sends $\sigma_b = (GS.\sigma, F_{root}, td)$ to *Adam*.
- **Phase 2:** Phase 2 is similar to phase 1 as long as the open oracle is not queried with the same index used in the challenge.
- **Guess:** *Adam* gives a guess $\bar{b}$ to *Eve* and she sends $\bar{b}$ to *Charles* as her own guess.

If *Adam* can guess $\bar{b}$ then *Eve* can guess the signer of $GS\sigma_b$. Note that we dropped $\mathcal{L}$ and $\mathcal{TK}$ from the anonymity analysis. $\mathcal{L}$ is an element that preserves the randomness of the signature in the public key and has nothing related to the signer. It is hard to distinguish between $\mathcal{L} = e(td, H_2(bsk[i]))$ and just a random $\mathcal{L}$ unless you have the tracing list $\mathcal{T}$. $\mathcal{TK}$ is a signature on $F_{root}$ and is anonymous since the group signature is anonymous. It also reveals nothing about the signer unless you have the tracing keys since both $\mathcal{TK}$ and $GS\sigma$ should trace to the same signer.

Next we shall prove the Theorem 6.3.2 by the same methodology we have followed for proving full anonymity. In the game in Section 5.4.2 we gave the adversary any information that will help him trace a signature but in this game since we need the private key $bsk[i]$ for creating the registration key $A_i$ and *Eve* does not have all $bsk[i]$ values, we shall have a Tracing-key oracle. The oracle is defined in the queries phase below:

**Theorem 6.3.2.** Traceability of AAS: *Given a group signature that is fully traceable the attribute based authentication scheme created by the general construction is fully traceable too.*

Consider the following game:

- **Init:** *Adam* chooses $(t_1,...,t_m) \in \mathbb{Z}_p^*$ as master keys.
- **Setup:** The Setup of this game model is similar to the setup in the anonymity game model where *Charles* runs *SGS.Setup* to produce the $S_{pub}$ and $S_{pri}$, then he sends $S_{pub}$ to *Eve*. *Eve* adds hash functions $H_1$ and $H_2$ to $S_{pub}$ and sends them to *Adam*. However, in this game model *Adam* is given the secret tracing keys $tk$.
- **Queries:** This stage consists of three oracles, the USK oracle, the Signature oracle and finally a Tracing-key oracle. The USK and Signature oracle are queried exactly as done in the anonymity game where *Adam* sends a query to *Eve* and *Eve* sends a query to the PriKey Oracle. Once *Eve* gets the private key from *Charles*, she can respond to *Adam* by either signing a message and sending $\sigma = (GS.\sigma, F_{root}, td, \mathcal{L}, \mathcal{TK})$ (Signature oracle) or by sending the pair $(A_i = H_1(bsk[i]), bsk[i])$ (USK oracle).
  The last oracle is the Tracing-key oracle where *Adam* queries a tracing key of a user. He sends *Eve* an index $i$. *Eve* sends the index she got to *Charles* querying the PriKey oracle. She gets $bsk[i]$ and responds to *Adam* by sending $H_2(bsk[i])$. Giving *Adam* access to this oracle and giving him the tracing key $tk$ makes it possible for him to trace any signature he likes, therefore there is no need for the Open oracle in this game.
- **Output:** *Adam* asks to be challenged on a message $M$ which he sends to *Eve*. *Eve* calculates a random $\overline{D}$ and sends to *Adam*. *Adam* replies with a forged signature $\sigma$. *Eve* challenges *Charles* on the same message. She sends $GS.\sigma$ to *Charles* as her output to the challenge.

If *Adam* is successful in breaking full traceability, then $\sigma$ should verify, which means $GS.\sigma$ should verify too. The signature $\sigma$ should also trace to a nonmember of the group or to a member that has not been queried. Running *SGS.Open* on $GS.\sigma$ is one step in the AAS scheme's *Open* algorithm. That means $GS.\sigma$ should also trace to either a nonmember or a member that has not been queried. We can conclude that if *Adam* was successful in winning the game, *Eve* would be successful in winning the game against *Charles*.

After having introduced a general construction and proving its security dependent on the security of the group signature used, we shall give an example of the general construction. Section 6.4 recalls the Bellare, Micciancio and Warinschi Group Signature and uses their scheme in demonstrating the general construction in Section 6.2.

## 6.4 Example of a General Construction

The foundations and formal definitions related to static group signature schemes were proposed in 2003 by Bellare, Micciancio and Warinschi [9]. In that paper, the authors proposed a construction based on digital signatures, encryption schemes and NIZK proof. In this section we first recall that group signature construction (Section 6.4.1) and then use it to demonstrate that the general construction proposed in Section 6.2 works (Section 6.4.2).

### 6.4.1 Bellare, Micciancio and Warinschi (BMW) Group Signature Scheme

Bellare, Micciancio and Warinschi proposed a group signature construction that uses three building blocks as follows:

- Digital Signature with Setup, Sign, and S.Verify as the algorithms (Section 2.4).
- Encryption Scheme with KeyGen, Encrypt and Decrypt as the algorithms (Section 2.3)
- Non-Interactive Zero Knowledge Proof with Prove and Z.Verify as algorithms (Section 2.5.2)

The following are the algorithms of the BMW group signature scheme:

- **BMW.Setup(k, n)**: Using security parameter $k$ set up the group signature for $n$ members as follows:
  1. Let the reference string of the zero knowledge proof be $R \leftarrow \{0,1\}^{Poly(k)}$ (i.e. polynomial in the length of the problem instance).
  2. Run Setup of the digital signature to generate $pk_s$ and $sk_s$.
  3. Run KeyGen of the encryption scheme to generate $pk_e$ and $sk_e$.
  4. For every member in the group run the setup of the digital signature scheme to obtain the pair ($pk_i$,$sk_i$) and then create a certificate $cert_i = Sign(sk_s, \langle i, pk_i \rangle)$.
  5. The public key $gpk = (R, pk_e, pk_s)$ for the group.
  6. The tracing key $tk = (n, pk_e, sk_e, pk_s)$.
  7. The private key $bsk[i] = (k, R, i, pk_i, sk_i, cert_i, pk_e, pk_s)$ for each member $i$ in the group.
- **BMW.Sign(M, bsk[i], gpk)**: To generate a signature $\sigma$ on message $M$ using the user's private key $bsk[i]$ follow the steps below:
  1. Create a signature $s = Sign(sk_i, M)$.

2. Choose a random element $r \in_R \{0,1\}^k$
3. Create ciphertext $C = Encrypt(pk_e, \langle i, pk_i, cert_i, s\rangle, r)$ where $r$ is the randomizing coin of the encryption.
4. Create the proof $\mathcal{P} = Prove(k, R, \langle pk_e, pk_s, M, C\rangle, \langle i, pk_i, cert_i, s, r\rangle)$. The proof is implying that the public key used in creating $s$ and the index $i$ are certified in $cert_i$ with respect to the key $pk_s$.
5. Signature $\sigma = (\mathcal{P}, C)$

- **BMW.Verify**$(\mathbf{M}, \mathbf{gpk}, \sigma)$ : Checking validity of the signature is done by verifying the zero knowledge proof, $Z.Verify(k, R, \mathcal{P}, \langle pk_e, pk_s, M, C\rangle)$.
- **BMW.Open**$(\mathbf{M}, \sigma, \mathbf{tk})$ : To trace a signature the following steps are done:
  1. Verify the signature if it is not valid then return 0.
  2. Calculate $Decrypt(C, sk_e) = \langle i, pk, cert, s\rangle$.
  3. If $n < i$ OR $S.Verify(pk, M, s) = 0$ OR $S.Verify(pk_s, cert, \langle i, pk\rangle) = 0$ return 0.
  4. Else return $i$.

The scheme was proven fully anonymous and fully traceable in [9] if we choose the suitable encryption scheme and digital signature scheme. The following are the full traceability and anonymity theorems.

**Theorem 6.4.1.** Full Anonymity of the BMW scheme:
*If the encryption scheme is an IND-CCA (Section 3.2) secure encryption scheme and NIZK is a simulation sound (Section 2.5.2), computational zero-knowledge proof system then group signature scheme BMW is fully-anonymous.*

**Theorem 6.4.2.** Full Traceability of the BMW scheme:
*If the digital signature scheme is secure against forgery under chosen message attack (Section 3.2) and NIZK is a sound in a non-interactive proof system then group signature scheme BMW is fully-traceable.*

The reader is referred to the main paper for the proofs. In this thesis we aim to use the BMW construction in illustrating the functionality of our general construction defined in Section 6.2.

### 6.4.2 AAS Scheme Based on BMW Group Signature Scheme

In this section we apply the general construction in Section 6.2 to the group signature construction in Section 6.4.1 to build a new AAS scheme. The algorithms of the scheme are described below:

- **GCS.Setup** : The central authority follows the steps below:

  1. Run $BMW.Setup$ to produce the system parameters. The output shall be the public key $gpk = (R, pk_e, pk_s)$ for the group. The tracing key $tk = (n, pk_e, sk_e, pk_s)$. The private key $bsk[i] = (k, R, i, pk_i, sk_i, cert_i, pk_e, pk_s)$ for each member $i$ in the group.
  2. Choose a hash function $H_1 : \{0,1\}^* \rightarrow G_1$ and $H_2 : \{0,1\}^* \rightarrow G_2$.
  3. The public parameters are the $gpk$, $H_1$ and $H_2$. The private parameters are $tk$ and any private parameter formed in the digital signature or encryption scheme chosen. The private key base is $bsk[i]$ while the registration key is $A_i$=$H_1$ ($k$,$R$,$i$,$pk_i$,$sk_i$,$cert_i$,$pk_e$,$pk_s$)
  4. Calculate $H_2$($k$,$R$,$i$,$pk_i$,$sk_i$,$cert_i$,$pk_e$,$pk_s$) and add the value to database $\mathcal{T}$

- **GCS.A.KeyGen$_{\mathbf{pub}}$**: The attribute authority calculates the attribute public key $bpk_j = H_2(R, pk_e, pk_s)^{t_j}$ where $t_j \in_R \mathbb{Z}_p^*$

- **GCS.A.KeyGen$_{\mathbf{pri}}$** : An attribute private key for user $i$ will be $T_{i,j} = A_i^{1/t_j}$ and $gsk = \langle bsk[i], T_{i,1}, ..., T_{i,j} \rangle$

- **GCS.Verifign** : The $Verifign$ runs as follows:

  1. $\mathcal{V}$ chooses a $\Gamma$ and collects $\bar{B} = (bpk_1,...,bpk_\kappa)$. $\mathcal{V}$ chooses $\alpha \in_R \mathbb{Z}_p^*$. $\mathcal{V}$ calculates $w = H_2(R, pk_e, pk_s)$. $\mathcal{V}$ can calculate $\overline{D} = (D_1,...,D_\kappa) = TCreate(\Gamma, \alpha, \overline{B})$ as shown in Section 5.2 and sends the values together with $\Gamma$ to $\mathcal{U}_i$.

  2. $\mathcal{U}_i$ runs the signature algorithm, $GS\sigma = BMW.Sign(M, bsk[i], gpk) = (\mathcal{P}, C)$. In order to run $F_{root} = TVerify(\overline{D}, \Im_i, \bar{T})$, $\mathcal{U}_i$ picks $\beta \in_R \mathbb{Z}_p^*$ and calculates $\bar{T} = \{CT_1, CT_2, ..., CT_\tau\}$.

  3. $\mathcal{U}_i$ calculates $\mathcal{TK} = BMW.Sign(F_{root}, bsk[i], gpk)$, $td = A_i^\beta$ and $\mathcal{L} = e(R, H_2(k, R, i, pk_i, sk_i, cert_i, pk_e, pk_s))$.
     The signature is $\sigma = (GS\sigma, \mathcal{TK}, F_{root}, td, \mathcal{L})$ and is send to the verifier.

  4. $\mathcal{V}$ can now verify $GS.\sigma$ using the algorithm $BMW.Verify$. He can verify the attributes by checking $F_{root}^{1/\alpha} = e(A_i^\beta, w)$. $\mathcal{V}$ verifies the signature on $F_{root}$ whether it is valid or not. If all three hold then output accept signature otherwise reject it.

- **GCS.Open** First step is to verify the signature $\sigma$. Note that the verifier can not manipulate $\alpha$ because $\alpha$ is inbuilt in $F_{root}$ which the signer agreed on by signing it in $\mathcal{TK}$. The second step is to run the $BMW.Open$ on $GS.\sigma$ that will identify the member. Finally, check whether $\mathcal{L} = e(td, H_2(k, R, i, pk_i, sk_i, cert_i, pk_e, pk_s))$ of the user where $H_2(k,R,i,pk_i,sk_i,cert_i,pk_e,pk_s)$ is retrieved from $\mathcal{T}$.

## 6.5 Chapter Summary

In this chapter we proposed a general construction that enables converting static group signatures to attribute authentication schemes (Section 6.2). The general construction is based on the attribute tree and the concept of group signature schemes. The AAS scheme resulting from the general construction was proven fully anonymous and fully traceable if the group signature used holds such properties too (Section 6.3). In Section 6.4 an example was given on the general construction and that example was based on Bellare, Micciancio and Warinschi group signature scheme. A general construction for dynamic AAS scheme is kept as future work and is not covered in this thesis.

# Chapter 7

# Concluding Remarks and Future Work

This Chapter highlights some future plans for taking the research of attribute authentication schemes further. Topics such as revocation, hierarchal authorities, hiding the policy of the verifier and possibility of other attribute authentication schemes are discussed.

## 7.1 Introduction

In this thesis we constructed an attribute authentication scheme that enables the verifier to decide a set of credentials he would like the signing member of a certain group (of potential signers) to possess. The core properties we required in the scheme are traceability, unforgeability, anonymity, unlinkability, attribute anonymity, coalition resistance and separability of authorities. We constructed our scheme in three phases: ABGS (Section 5.3), AAS (Section 5.4), and DAAS (Section 5.5), each time improving the scheme to get closer to our aim. However, there are some desired properties that improve our scheme and provide stronger foundations.

Revocation is one of these properties that can strengthen our foundation. In the AAS scheme we proposed in Section 5.4 we had discussed revocation of members of the group however the revocation of attributes is kept as future work (See Section 7.2).

Another property would be hierarchy in authorities. We have shown different levels of separability in our constructions, in Chapter 5, which is a nice feature in any cryptographic scheme. Nonetheless some authorities have a hierarchal structure rather then a separable one. In Section 7.3 we explain the need of hierarchal structures.

Chapter 5 also discussed the need to preserve attribute anonymity of a signer which we have successfully achieved by using the attribute tree structure, whereas hiding the policy of the verifier is not mentioned. Everyone can tell the type of credentials a

verifier needs even the nonmembers of the group. Section 7.4 suggests general ideas on how to solve the problem.
In Chapter 4 we proposed a general construction for the AAS scheme. The general construction is based on the attribute tree structure and on static group signatures. A similar construction to achieve DAAS schemes will be helpful (See Section 7.5).
The final possible extension for our research is to try create other cryptographic schemes that are attribute based. Section 7.6 explains some of the ideas that can be considered.

## 7.2 Revocation

In the real life scenario humans do not possess their attributes forever. A student can graduate, an employee can retire, a prescription can expire...etc. Revocation is the process of discarding an attribute from a certain user in the system.
The first revocation technique used widely in cryptography and can be applied to the AAS scheme is timestamps. In such a technique an expiry date is implicitly built in the certificate. When the date passes the certificate will not verify anymore. So if it takes a student three years to graduate we can create a student ID that expires by that time. This technique is useful and usually needs less computational time than other known revocation techniques in the literature.
The second method used for revocation is by renewing the certificates. So every time a user is revoked the authority publishes specific information that will help valid users update their certificates accordingly. In other words the user recalculates the private key used in creating certificates. For example every year a group of students graduate, the university publishes certain information that can be used only by current students to update their certificates. The computational overhead of such an approach is usually higher than timestamps especially for signers or certificate owners. The advantage of such approach is that updating can happen at anytime in the future.
The last method known in literature is a revocation list. The authority in control can publish a list of information that helps knowing whether a user of a certain certificate is revoked or not by testing it against certain elements in that list. This approach removes the overhead from the owner of the certificate to the verifier since they are the ones checking the revocation list. It has an advantage on the timestamps since revocation again can happen anytime in the future. Finding a balance of all three methods of revoking is still an unresolved issue in cryptography. Several studies have been done on revocation in cryptographic schemes [101, 50, 116, 75]. Myers established a metric in which these various approaches are analyzed, and that includes [99]:

- Population Size and Symmetry: The absolute size of potentially revocable certificates is a major influence when it comes to deciding the revocation technique adopted. In AAS schemes that means the number of users in the system mul-

tiplied by the maximum number of attributes one can own. The effects of population asymmetry must be considered too. For example in the AAS scheme, this resembles the following question: Is the set of verifiers that need a specific attribute as part of their policy smaller then the set of members owning that attribute?

- Timeliness of revocation information: This is measured by asking oneself: How soon would the verifier need to know that a certain attribute is revoked from a certain user?
- Connectivity and bandwidth utilization: Some revocation approaches will require the verifier to be online others rely on cached data. Since in our AAS scheme we have multiple attributes each from an authority assuming connectivity is a bad idea.
- Responsiveness to security critical needs: Attributes that are sensitive need immediate revocation if exposed and an expiry date is not sufficient. However timestamps are efficient to compute and therefore used for attributes that are less sensitive.

## 7.3 Hierarchal Authorities

In the constructions in Chapter 5 authorities are acting separably. However in real life they are either separate or hierarchal. For example, having a certificate or a student ID proving that Alice is a student in the University of Bath implies she is part of the student group in general, meaning that all universities need to be a hierarchal organization that fall under the Ministry of Education. This kind of hierarchy helps in minimizing the numbers of certificates users need to have. Theoretically speaking, it should also improve computational time of the certificate. Hierarchy has been studied in two fields of cryptography. The first field is "Role Based Access Controls" [125] and the second field is "Identity Based Cryptography" [66]. The underlying technique in both fields is quite similar and depends on identifying either the "User of the System" or the "Authority". This compromises anonymity of the scheme. However, Kim *et al.* proposed a hierarchal group signature scheme [84] that possibly can help in creating a hierarchal AAS scheme.

## 7.4 Concealing the Policy of the Verifier

In all schemes proposed in Chapter 5 the verifier had to decide on the credentials he needs and announce them to all members of the group. Anyone, including an eavesdropper, will know what type of attributes the verifier is looking for in a signature.

The verifier either encloses his policy with the verification key or the policy can be derived from the key. It would have been nice if the policy was encrypted in such a way that only members of the group who satisfy it can decrypt the cipher. A possible solution for this problem is using searchable encryption [22, 80]. Searchable encryption is the ability to search for certain keywords in the ciphertext using trapdoors. Assuming the verifier encrypts the verification request so that the keywords are the attributes he requires, each member of the group will try to search the ciphertext for the attributes they own and if they find it they can sign the message. A major difference though between the normal searchable encryption scheme and the one we need for this purpose is the trapdoors. In our scheme trapdoors should be private elements that only members who own a specific attribute have its trapdoor whereas the main searchable encryption scheme has them public.

Attribute based encryption [69] is another possible solution that seems promising. In such an encryption scheme the encryptor (i.e. verifier in AAS standards) decides the attributes he wants the decryptor (i.e. Potential signer within the group) to possess in order to manage to decrypt the ciphertext. In other words the verifier encrypts his policy with itself and sends it out to the members of the group. Only members who satisfy it can actually decrypt and reveal the policy.

## 7.5 More on The General Construction

In Chapter 4 a general construction for an AAS scheme has been proposed. However the need of dynamic groups requires finding a general construction for DAAS schemes. Loosely speaking, the idea is to find a way to convert any dynamic group signature (See Chapter 4) to a dynamic attribute based scheme (See Chapter 5). If we follow the same methodology we used in Chapter 4 two issues will come up. First of all, the registration key was equal to $A_i = H_1(bsk[i])$. In dynamic group signatures $bsk[i]$ is chosen jointly by the user and the authority in the join protocol and the user is the only one who actually knows its final value. An option is to use key $reg_i$ and hashing it to become the registration key $A_i = H_1(reg_i)$. The second problem is to find a way to run the $Judge$ algorithm on the attributes without revealing anything about $A_i$. Once a general construction is created it has to be proved secure under full anonymity, full traceability and Non-frameability.

In our general construction the attribute tree and bilinear maps are used. It would be desirable to define a general construction based on a higher level cryptosystem. One general idea is to use building blocks such as an attribute based encryption and group signature. The central authority runs a group signature setup to generate one public key $gpk$ for the group, a tracing key $tk$, and a private key $bsk[i]$ for each member $i$ in the group. Each attribute authority creates a public key and a private key for the

attributes. The verifier creates his request by sending an encrypted random message to the group. Any member with sufficient attributes will be able to decrypt and then send that message back signed as a member of the group. This general construction is on a higher cryptosystem level and requires to be proven secure under all the notions defined in Chapter 5.

## 7.6 Attribute Based Cryptography

In the literature there are two types of attribute based cryptographic schemes: Encryption (ABE) and signature (ABS). Goyal *et al.* [69] proposed the first ABE scheme where decrypting relied on owning a set of attributes and much research was done in order to improve that. In [45, 134] the authors proposed ciphertext policy attribute-based encryption (CP-ABE), where all secret keys are associated with sets of attributes and ciphertexts are linked with an access structure on attributes. Decryption in such a scheme is enabled if and only if the users' attribute set satisfies the ciphertext access structure. In [106] the authors used circuits rather then attribute trees and the benefit was to enable a "Not" relation in the policies. Attribute based signatures were proposed in our thesis and extensions were explained in Chapter 5. Maji, Prabhakaran, and Rosulek extended our work in order to disable traceability and making signatures anonymous to everyone including any type of manager or authority [93]. As a backbone to their scheme they have used Mesh Signatures. More cryptographic schemes can be useful if they were attribute based and the following are some examples

- Signcryption: Traditionally, documents were signed and then encrypted in order to maintain authenticity and confidentiality. However that is not efficient since you have to run four algorithms in total, Sign/Encrypt on one side and Verify/Decrypt on the other. In [141] the first signcryption scheme was proposed where Sign and Encrypt were merged into one algorithm and so was Verify and Decrypt. Having an attribute based signcryption will help in hiding the verifier's policy. It will be joining the ABE together with the ABS. It will be a powerful tool that can be used for trust negotiation purposes as described in Section 5.1.1.

- E-voting scheme: Electronic voting schemes are cryptosystems that enable fair voting. They enable people to elect and vote electronically. Chaum was the first to propose the concept in his work of [39]. The system should fulfill the following three properties:

  - Correctness: The computed tally must be correct according to the cast legal votes.
  - Privacy: Votes must be kept private.
  - Availability: Every entitled voter is able to participate in the election.

  - Verification: A voter should be able to verify their vote has been registered correctly.

  In some voting systems the voter should be above a certain age, a citizen of a certain country,...etc. These kind of conditions require an attribute based voting scheme.

- Broadcast encryption: A Broadcast Encryption Scheme is a set of algorithms that allow a transmitter to send encrypted messages to a collection of users such that only a privileged subset of users can decrypt them. Broadcast Encryption was desperately needed for copyright purposes, digital TV channels distribution, and music transmissions. Such an encryption scheme was first introduced in 1994 in [62]. In that paper the authors pictured a scenario where there is a center and a set of users that are provided by that center with keys. Now the center broadcast an encrypted message and that only subscribed users could decrypt. Nonmembers of such system are curious to get to know the message. If they succeed the system is said to be broken. We call the scheme K-resilient if it takes K revoked users to collude and break the system. Broadcast encryption can be used in military forces where the army is sent a message and any soldier can decrypt it. In such a situation it might be necessary that specific soldiers are the only ones capable of decrypting. For instance you might want the decryptor to be a senior officer in the air force. Attribute based broadcast encryption would be a nice invention that serves such purposes.

Other cryptosystems exist and may benefit from being transfered into attribute based schemes. We have just explained some examples in this section.

## 7.7 Expanding to more Complicated Policies

In our thesis the policy was expressed as a boolean monotone expression with “and”, “or”, and “ m of n ” relations. Extra relations can be added, examples on such relations are given below and are based on scenario 5.1.1.

- Not Relation : Alice wants the potential signer to not be from department C.
- If..then..Relation : If employee is a senior manager then he is a manager.
- Greater/Less than Relation: A potential signer should have been employed for less than 10 years.

The attribute tree is a powerful tool to express attributes however there might be a different structure that is capable of capturing more relations between the attributes and is more efficient than the attribute tree. Choosing polynomials and calculating

Lagrange interpolation may get expensive if the number of attributes required is high since each node should have a polynomial of a degree of one less than the threshold gate.

## 7.8 Scalability of the AAS Scheme

The scheme has its disadvantages when it comes to implementing it on a larger scale. The system needs one central authority and it is hard to decide in real life who should play that role. Besides which, having one authority causes a bottleneck. It would be nice if we can change the system to enable more than one higher authority.
Each member of a group has to contact several entities to collect different attributes. However, comparing it to the real life scenario people do contact a variety of authorities in order to get their credentials. For instance, your birth certificate, passport, ID card, driving license, etc, are all collected from a different authorities.
Scaling the system to include a massive number of attributes in one policy needs addressing. The more attributes are needed the larger the attribute tree is and the larger the verification key $\overline{D}$ is. This will make it hard to send it to the potential signers and the computational overhead increases too. One may argue that the ultimate policy has to be agreed by humans, and that does not require a large set of attributes anyway. Nevertheless, it would be nice to have the size of the verification key constant rather than linearly dependent on the number of attributes.

# Bibliography


[1] M. Abe and S. Fehr. Adaptively Secure Feldman VSS and Applications to Universally-Composable Threshold Cryptography. In *Advances in Cryptology - CRYPTO 2000*, volume 3152 of *Lecture Notes in Computer Science*, pages 317–334. Springer-Verlag, 2004.

[2] S. Al-Riyami, J. Malone-Lee, and N. Smart. Escrow free encryption supporting cryptographic workflow. In *International Journal of Information Security*, volume 5, pages 217–230, September 2006.

[3] J. Almansa, I. Damgård, and J. Nielsen. Simplified Threshold RSA with Adaptive and Proactive Security. In *Advances in Cryptology - EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 593–611. Springer-Verlag, 2006.

[4] G. Ateniese, J. Camenisch, S. Hohenberger, and B. de Medeiros. Practical group signatures without random oracles. Cryptology ePrint Archive, Report 2005/385, 2005. `http://eprint.iacr.org/`.

[5] G. Ateniese, D. Song, and G. Tsudik. Quasi-Efficient Revocation in Group Signatures. In *Proceedings of Financial Cryptography'01*, volume 2357 of *Lecture Notes in Computer Science*, pages 183–197. Springer-Verlag, 2001.

[6] G. Ateniese and G. Tsudik. Some Open Issues and New Directions in Group Signature Schemes. In *Proceedings of Financial Cryptography'99*, volume 1648 of *Lecture Notes in Computer Science*, pages 196–211. Springer-Verlag, 1999.

[7] M. Au, J. Liu, T. Yuen, and D. Wong. ID-Based Ring Signature Scheme Secure in the Standard Model. In *Proceedings of Advances in Information and Computer Security*, volume 4266 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 2006.

[8] M. Bellare. A Note on Negligible Functions. *Journal of Cryptology*, 15:271–284, 2002.

[9] M. Bellare, D. Micciancio, and B. Warinschi. Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on Gen-

eral Assumptions. In *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 614–629. Springer-Verlag, 2003.

[10] M. Bellare and G. Neven. Identity-Based Multi-signatures from RSA. In *CT-RSA*, pages 145–162, 2007.

[11] M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[12] M. Bellare and P. Rogaway. Introduction to Modern Cryptography. `http://www.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf`, 2005.

[13] M. Bellare, H. Shi, and C. Zhang. Foundations of Group Signatures: The Case of Dynamic Groups. In *Topics in Cryptology CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 136–153. Springer-Verlag, 2005.

[14] A. Bender, J. Katz, and R. Morselli. Ring Signatures: Stronger Definitions and Constructions Without Random Oracles. In *Theory of Cryptography*, volume 3876 of *Lecture Notes in Computer Science*, pages 60–79. Springer-Verlag, 2006.

[15] I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press, 1 edition, 1999.

[16] I. Blake, G. Seroussi, and N. Smart. *Advances in Elliptic Curves in Cryptography.* Cambridge University Press, 1 edition, 2004.

[17] M. Blum, P. Feldman, and S. Micali. Non-Interactive Zero-Knowledge and Its Applications. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, ACM, pages 103–112, 1988.

[18] B. Boer. Diffie-Hellman is as Strong as Discrete log for Certain Primes. In *Advances in Cryptology - CRYPTO 1988*, volume 403 of *Lecture Notes in Computer Science*, pages 530–539. Springer-Verlag, 1988.

[19] D. Boneh and X. Boyen. Short Signatures without Random Oracles. In *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 382–400. Springer-Verlag, 2004.

[20] D. Boneh, X. Boyen, and H. Shacham. Short Group Signatures. In *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41 – 55. Springer-Verlag, 2004.

[21] D. Boneh and M. Franklin. Identity-Based Encryption from the Weil Pairing. In *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer-Verlag, 2001.

[22] D. Boneh, G.Crescenzo, R. Ostrovsky, and G. Persiano. Public Key Encryption with Keyword Search. In *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer-Verlag, 2004.

[23] D. Boneh, C. Gentry, H. Shacham, and B. Lynn. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer-Verlag, 2003.

[24] D. Boneh, C. Gentry, and B. Waters. Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys. In *Advances in Cryptology - CRYPTO'05*, LNCS, pages 258–275. Springer-Verlag New York Inc, 2005.

[25] D. Boneh and H. Shacham. Group Signatures with Verifier-Local Revocation. In *Proceedings of the 11'th ACM conference on Computer and Communications Security*, pages 168–177, 2004.

[26] C. Boyd. Digital Multisignatures. In *Cryptography and Coding*, H.J.Beker and F.C.Piper Eds., pages 241–246. Oxford University Press, 1989.

[27] X. Boyen. Mesh Signatures: How to Leak a Secret with Unwitting and Unwilling Participants. In *Advances in Cryptology - EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 210–227. Springer-Verlag, 2007.

[28] R. Bradshaw, J. Holt, and K. Seamons. Concealing complex policies with hidden credentials. In *Proceedings of 11th ACM Conference on Computer and Communications Security*, pages 146–157. ACM Press, 2004.

[29] S. Brands. An efficient off-line electronic cash system based on the representation problem. `http://citeseer.ist.psu.edu/488307.html`, 1993. CWI (Centre for Mathematics and Computer Science) Report CS-R9323.

[30] E. Bresson, J. Stern, and M. Szydlo. Threshold Ring Signatures and Applications to Ad-hoc Groups. In *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 465–480. Springer-Verlag, 2002.

[31] J. Camenisch. Efficient and Generalized Group Signatures. In *Advances in Cryptology - EUROCRYPT 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 465–479. Springer-Verlag, 1997.

[32] J. Camenisch and A. Lysyanskaya. Efficient Non-transferable Anonymous Multishow Credential System with Optional Anonymity Revocation. In *Advances in Cryptology - EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer-Verlag, 2001.

[33] J. Camenisch and A. Lysyanskaya. Efficient Revocation of Anonymous Group Membership. Cryptology ePrint Archive, Report 2001/113, 2001. `http://eprint.iacr.org/`.

[34] J. Camenisch and A. Lysyanskaya. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 61–76. Springer-Verlag, 2002.

[35] J. Camenisch and A. Lysyanskaya. Signature Schemes and Anonymous Credentials from Bilinear Maps. In *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer-Verlag, 2004.

[36] J. Camenisch and M. Stadler. Efficient Group Signature Schemes for Large Groups. In *Advances in Cryptology - CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer-Verlag, 1997.

[37] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, revisited (preliminary version). In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, pages 209–218. ACM Press, 1998.

[38] C. Castelluccia, S. Jarecki, J. Kim, and G. Tsudik. A Robust Multisignatures Scheme with Applications to Acknowledgment Aggregation. In *SCN*, volume 3352, pages 193–207, 2004.

[39] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM*, 24(2):84–90, 1981.

[40] D. Chaum. Security without Identification Transaction Systems to Make Big Brother Obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[41] D. Chaum and V. Heyst. Group Signatures. In *Advances in Cryptology - EUROCRYPT 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer-Verlag, 1991.

[42] L. Chen and T. Pedersen. New Group Signature Schemes. In *Advances in Cryptology - EUROCRYPT 1994*, volume 950 of *Lecture Notes in Computer Science*, pages 171–181. Springer-Verlag, 1994.

[43] Z. Chen, J. Wang, Y. Wang, J. Huang, and D. Huang. An Efficient Revocation Algorithm in Group Signatures. In *Information Security and Cryptology - ICISC*, volume 2971 of *Lecture Notes in Computer Science*, pages 339–351. Springer-Verlag, 2004.

[44] J. H. Cheon. Security Analysis of the Strong Diffie-Hellman Problem. In *Advances in Cryptology - EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 1–11. Springer-Verlag, 2006.

[45] L. Cheung and C. Newport. Provably Secure Ciphertext Policy ABE. In *ACM Conference on Computer and Communications Security*, pages 456–465, 2007.

[46] J. Choon and J. Cheon. An Identity-Based Signature from Gap Diffie Hellman Groups. In *Public Key Cryptography PKC*, volume 2567 of *Lecture Notes in Computer Science*, pages 18–30. Springer Verlag, 2003.

[47] S. Chow, L. Hui, and S. Yiu. Identity Based Threshold Ring Signature. In *Information Security and Cryptology*, volume 2971 of *Lecture Notes in Computer Science*, pages 218–232. Springer-Verlag, 2004.

[48] C. Cocks. An Identity Based Encryption Scheme Based on Quadratic Residues. In *Proceedings of Cryptography and Coding*, volume 2260 of *Lecture Notes in Computer Science*, pages 360–363. Springer-Verlag, 2001.

[49] H. Cohen and G. Frey. *The Handbook of Elliptic and Hyperelliptic Curve Cryptography.* Discrete Mathematics and Its Applications-Chapman and Hall/CRC, 2005.

[50] D. A. Cooper. A model of certificate revocation. In *ACSAC '99: Proceedings of the 15th Annual Computer Security Applications Conference*, page 256, Washington, DC, USA, 1999. IEEE Computer Society.

[51] J. Coron and A. May. Deterministic polynomial-time equivalence of computing the rsa secret key and factoring. *J. Cryptology*, 20(1):39–50, 2007.

[52] C. Delerable and D. Pointcheval. Dynamic Fully Anonymous Short Group Signatures. In *Proceedings of VIETCRYPT 2006*, volume 4341 of *Lecture Notes in Computer Science*, pages 193–210. Springer-Verlag, 2006.

[53] H. Delfs and H. Knebl. *Introduction to Cryptography* . Springer, 1 edition, 2002.

[54] A. Dent. Fundamental Problems in Provable Security and Cryptography. In *Philosophical Transactions*, volume 364, pages 3215–3230. Royal society, 2006.

[55] Y. Desmedt and Y. Frankel. Threshold Cryptosystems. In *Advances in Cryptology - CRYPTO 1989*, volume 453 of *Lecture Notes in Computer Science*, pages 307–315. Springer-Verlag, 1989.

[56] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[57] X. Ding, G. Tsudik, and S. Xu. Leak-Free Group Signatures with Immediate Revocation. In *Proc. of 24th International Conference on Distributed Computing Systems*, pages 608–615, 2004.

[58] C. Dwork, J. Lotspiech, and M. Naor. Digital Signets: Self-enforcing Protection of Digital Information. In *Proc. 28th STOC*, Lecture Notes in Computer Science, pages 489–498. Springer-Verlag, 1996.

[59] C. Dwork, J. Lotspiech, and M. Naor. Digital signets: self-enforcing protection of digital information (preliminary version). In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 489–498, New York, NY, USA, 1996. ACM.

[60] K. Eisentrager, K. Lauter, and P. Montgomery. Fast Elliptic Curve Arithmetic and Improved Weil Pairing Evaluation. In *Topics in Cryptology CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 343–354. Springer-Verlag, 2003.

[61] T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology - CRYPTO 1984*, volume IT-31 of *IEEE Transactions on Information Theory*, pages 469–472. Springer-Verlag, 1984.

[62] A. Fiat and M. Naor. Broadcast Encryption. In *Advances in Cryptology - CRYPTO 1993*, volume 773 of *Lecture Notes in Computer Science*, pages 480–491. Springer-Verlag, 1993.

[63] S. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate Pairing . In *Algorithmic Number Theory*, volume 2369 of *Lecture Notes in Computer Science*, pages 69–86. Springer-Verlag, 2002.

[64] S. Galbraith, H. Hopkins, and I. Shparlinski. Secure Bilinear Diffie-Hellman Bits. In *Information Security and Privacy*, volume 3108 of *Lecture Notes in Computer Science*, pages 370–378. Springer-Verlag, 2004.

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

[66] C. Gentry and A. Silverberg. Hierarchical ID-Based Cryptography. In *Advances in Cryptology - ASIACRYPT 2002*, volume 2501 of *Lecture Notes of Computer Science*, pages 149–155. Springer-Verlag, 2002.

[67] O. Goldreich, B. Pfitzman, and R. Rivest. Self-Delegation with Controlled Propagation or What If you Lose your Laptop. In *Advances in Cryptology -*

*CRYPTO 1998*, volume 1642 of *Lecture Notes in Computer Science*, pages 153–168. Springer-Verlag, 1998.

[68] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[69] 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.

[70] J. Groth. Fully Anonymous Group Signatures Without Random Oracles. In *Advances in Cryptology - ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 164–180. Springer-Verlag, 2007.

[71] J. Herranz and F. Laguillaumie. Blind Ring Signatures Secure Under the Chosen-Target-CDH Assumption. In *Proceedings of Information Security*, volume 4176 of *Lecture Notes in Computer Science*, pages 117–130. Springer-Verlag, 2006.

[72] J. Herranz and G. Sez. New Identity-Based Ring Signature Schemes. In *Information and Communications Security*, volume 3269 of *Lecture Notes in Computer Science*, pages 27–39. Springer-Verlag, 2004.

[73] F. Hess. Efficient Identity Based Signature Schemes Based on Pairings. In *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 310–324. Springer-Verlag, 2003.

[74] J. Holt, R. Bradshaw, K. Seamons, and H. Orman. Hidden Credentials. In *2nd ACM Workshop on Privacy in the Electronic Society*, pages 1–8, 2003.

[75] H. Imai and Y. Zheng, editors. *Public Key Cryptography, Third International Workshop on Practice and Theory in Public Key Cryptography, PKC 2000, Melbourne, Victoria, Australia, January 18-20, 2000, Proceedings*, volume 1751 of *Lecture Notes in Computer Science*. Springer, 2000.

[76] K. Itakura and K. Nakamura. A Public Key Cryptosystem Suitable for Digital Multisignatures. In *NEC Research and Development*, pages 71:1–8, 1983.

[77] A. Joux. A One Round Protocol for Tripartite Diffie-Hellman. In *ANTS-IV: Proceedings of the 4th International Symposium on Algorithmic Number Theory*, pages 385–394, London, UK, 2000. Springer-Verlag.

[78] D. Kahn. *The Codebreakers, The Story of Secret Writing*. MacMillan, 1st edition, 1967.

[79] D. Khader. Attribute Based Group Signatures. Cryptology ePrint Archive, Report 2007/159, 2007. `http://eprint.iacr.org/`.

[80] D. Khader. Public Key Encryption with Keyword Search Based on K-Resilient IBE. In *Proceedings of Computational Science and Its Applications ICCSA 2007*, volume 4707 of *Lecture Notes in Computer Science*, pages 1086–1095. Springer-Verlag, 2007.

[81] D. Khader. Authenticating with Attributes. Cryptology ePrint Archive, Report 2008/031, 2008. `http://eprint.iacr.org/`.

[82] H. Kim, J. Lim, and D. Lee. Efficient and Secure Member Deletion in Group Signature Schemes. In *Information Security and Cryptology*, volume 2015 of *Lecture Notes in Computer Science*, pages 150–161. Springer-Verlag, 2001.

[83] S. Kim, S. Park, and D. Won. Convertible Group Signatures. In *Advances in Cryptology - ASIACRYPT 1996*, volume 1163 of *Lecture Notes in Computer Science*, pages 311–321. Springer-Verlag, 1996.

[84] S. Kim, S. Park, and D. Won. Group Signatures for Hierarchical Multigroups. In *Information Security*, volume 1396 of *Lecture Notes in Computer Science*, pages 273–281. Springer-Verlag, 1998.

[85] B. King. Improved Methods to Perform Threshold RSA. In *Advances in Cryptology - ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 359–372. Springer-Verlag, 2000.

[86] D. Kwak, S. Moon, G. Wang, and R. Deng. A secure extension of the Kwak-Moon group signcryption scheme. *Computers & Security*, 25(6):435–444, 2006.

[87] C. Li, T. Hwang, and Y. Lee. Threshold-Multisignature Schemes where Suspected Forgery implies Traceability of Adversarial Shareholders. In *Advances in Cryptology - EUROCRYPT 1995*, volume 950 of *Lecture Notes in Computer Science*, pages 194–204. Springer-Verlag, 1995.

[88] Z. Li, Y. Wang, Y. Yang, and W. Wu. Cryptanalysis of convertible group signature. In *Electronics Letters of IEEE*, volume 35, pages 1071–1072, 1999.

[89] C. Lim and P. Lee. Remarks on Convertible Signatures of ASIACRYPT'96. In *Electronics Letters of IEEE*, volume 33, pages 383–384, 1997.

[90] A. Lysyanskaya and Z. Ramzan. Group Blind Digital Signatures: A Scalable Solution to Electronic Cash. In *Proceedings of Financial Cryptography'98*, volume 1465 of *Lecture Notes in Computer Science*, pages 184–197. Springer-Verlag, 1998.

[91] A. Lysyanskaya, R. Rivest, A. Sahai, and S. Wolf. Pseudonym Systems. In *Selected Areas in Cryptography*, volume 1758 of *Lecture Notes in Computer Science*, pages 184–199. Springer-Verlag, 1999.

[92] Y.-D. Lyuu and M.-L. Wu. Convertible Group Undeniable Signatures. In *Information Security and Cryptology ICISC 2002*, volume 2587 of *Lecture Notes in Computer Science*, pages 48–61. Springer-Verlag, 2003.

[93] H. Maji, M. Prabhakaran, and M. Rosulek. Attribute-Based Signatures: Achieving Attribute-Privacy and Collusion-Resistance. Cryptology ePrint Archive, Report 2008/328, 2008. `http://eprint.iacr.org/`.

[94] W. Mao. *Modern Cryptography: Theory and Practice*. Prentice Hall Professional Technical Reference, 2003.

[95] U. Maurer. Towards the Equivalence of Breaking the Diffie-Hellman Protocol and Computing Discrete Algorithms. In *Advances in Cryptology - CRYPTO 1994*, volume 839 of *Lecture Notes in Computer Science*, pages 271–281. Springer-Verlag, 1994.

[96] U. Maurer and S. Wolf. Lower Bounds on Generic Algorithms in Groups. In *Advances in Cryptology EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, page 72. Springer-Verlag, 1998.

[97] A. Menezes, P. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[98] J. Mikucki. Do You Know What I Know? `http://www.citeseer.ist.psu.edu/mikucki99do.html"`, 1999.

[99] M. Myers. Revocation: Options and Challenges. In *Financial Cryptography*, volume 1465 of *Lecture Notes in Computer Science*, pages 165–171. Springer-Verlag, 1998.

[100] T. Nakanishi and Y. Sugiyama. A Group Signature Scheme with Efficient Membership Revocation for Reasonable Groups. In *Information Security and Privacy*, volume 3108 of *Lecture Notes in Computer Science*, pages 336–347. Springer-Verlag, 2004.

[101] M. Naor and K. Nissim. Certificate Revocation and Certificate Update. In *Proceedings of the 7th USENIX Security Symposium*, pages 217–228, San Antonio, TX, Jan. 1998.

[102] M. Noar. Deniable Ring Authentication. In *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 481–489. Springer-Verlag, 2002.

[103] T. Ohata and T. Okamoto. A Digital Multisignature Scheme based on the Fiat-Shamir scheme. In *Advances in Cryptology - ASIACRYPT 1991*, volume 739 of *Lecture Notes in Computer Science*, pages 75–79. Springer-Verlag, 1991.

[104] T. Okamoto. A digital multisignature scheme using bijective public-key cryptosystems. *ACM Trans. Comput. Syst.*, 6(4):432–441, 1988.

[105] R. Oppliger. *Contemporary Cryptography.* Artech House Publishers, 2005.

[106] R. Ostrovsky, A. Sahai, and B. Waters. Attribute-based Encryption with Non-Monotonic Access Structures. In *ACM Conference on Computer and Communications Security*, pages 195–203, 2007.

[107] S. Park, S. Kim, and D. Won. ID-based Group Signature. In *Electronics Letters*, volume 33, pages 1616–1617, 1997.

[108] T. Pedersen. A Threshold Cryptosystem without a Trusted Party. In *Advances in Cryptology - EUROCRYPT 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526. Springer-Verlag, 1991.

[109] T. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology - CRYPTO 1991*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer-Verlag, 1991.

[110] H. Petersen. How to Convert any Digital Signature Scheme into a Group Signature Scheme. In *Proceedings of Security Protocols Workshop*, pages 177–190, 1997.

[111] 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.

[112] J. Pollard. Monte Carlo Methods for Index Computation . *Mathematics of Computation*, 32:918–924, 1978.

[113] S. Popescu. An Efficient ID-based Group Signature Scheme. In *Studia Univ. Babes-Bolyai Informatica*, volume 2, pages 29–36, 2002. `http://www.cs.ubbcluj.ro/~studia-i/2002-2/`.

[114] J. Quisquater, L. Guillou, M. Annick, and T. Berson. How to explain zero-knowledge protocols to your children. In *Advances in Cryptology - CRYPTO 1989*, pages 628–631, New York NY USA, 1989. Springer-Verlag New York Inc.

[115] R.Gennaro, S. Jarecki, H.Krawczyk, and T. Rabin. Robust Threshold DSS Signatures. In *Advances in Cryptology - EUROCRYPT 2001*, volume 1070 of *Lecture Notes in Computer Science*, pages 354–371. Springer-Verlag, 2001.

[116] R. Rivest. Can We Eliminate Certificate Revocation Lists? In *Financial Cryptography*, volume 1465 of *Lecture Notes in Computer Science*, pages 737–766. Springer-Verlag, 1998.

[117] R. Rivest, A. Shamir, and Y. Tauman. How to Leak a Secret. In *Advances in Cryptology - ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, page 552. Springer-Verlag, 2001.

[118] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[119] K. Rosen. *An Introduction to Cryptography-Second Edition*. Discrete Mathematics and Its Applications. Chapman and Hall, 2006.

[120] A. SAHAI. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *Proceedings of the 40th Symposium on Foundations of Computer Science*, page 543. IEEE, 1999.

[121] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. Wiley, New York, 2nd edition, 1996.

[122] C. Schnorr. Efficient Identification and Signatures for Smart Cards. In *Advances in Cryptology - CRYPTO 1989*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer-Verlag, 1989.

[123] A. Shamir. Identity-Based Cryptosystems and Signature Schemes. In *Advances in Cryptology - CRYPTO 1984*, volume 7 of *Lecture Notes in Computer Science*, pages 47–53. Springer-Verlag, 1984.

[124] C. Shannon. Communication Theory of Secrecy Systems. In *Bell Systems Technical Journal*, volume 28, pages 656–715, 1949.

[125] W. Shiuh-Jeng and C. Jin-Fu. A Hierarchical and Dynamic Group-Oriented Cryptographic Scheme (Special Section on Cryptography and Information Security). *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 79(1):76–85, 1996.

[126] V. Shoup. Practical Threshold Signatures. In *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–220. Springer-Verlag, 2000.

[127] N. Smart. How secure are elliptic curves over composite extension fields? In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, pages 30–39, London, UK, 2001. Springer-Verlag.

[128] N. Smart. *Cryptography: An Introduction*. Mcgraw-Hill College, 2004.

[129] E. Suli and D. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, 2003.

[130] W. Susilo and Y. Mu. Deniable Ring Authentication Revisited. In *Applied Cryptography and Network Security (ACNS 2004)*, volume 3089 of *Lecture Notes in Computer Science*, pages 149–163. Springer-Verlag, 2004.

[131] M. Trolin and D. Wikström. Hierarchical Group Signatures. In *Proceedings of Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 446–458. Springer-Verlag, 2005.

[132] E. Verheul. Self-Blindable Credential Certificates from the Weil Pairing. In *Advances in Cryptology - ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 533–551. Springer-Verlag, 2001.

[133] G. Wang, R. Deng, D. Kwak, and S. Moon. Security Analysis of Two Signcryption Schemes. In *Information Security*, volume 3225 of *Lecture Notes in Computer Science*, pages 123–133. Springer-Verlag, 2004.

[134] B. Waters. Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization. Cryptology ePrint Archive, Report 2008/290, 2008. `http://eprint.iacr.org/`.

[135] V. Wei, T. Yuen, and F. Zhang. Group Signature Where Group Manager Members and Open Authority Are Identity-Based. In *Proceedings of Information Security and Privacy*, volume 3574 of *Lecture Notes in Computer Science*, pages 468–480. Springer-Verlag, 2005.

[136] W. Winsborough and N. Li. Safety in Automated Trust Negotiation. *ACM Trans. Inf. Syst. Secur.*, 9(3):352–390, 2006.

[137] W. Winsborough, K. Seamons, and V. Jones. Automated Trust Negotiation. In *DARPA Information Survivability Conference and Exposition 2000. DISCEX '00*, volume 1, pages 88–102. IEEE, 2000.

[138] W. H. Winsborough and N. Li. Towards Practical Automated Trust Negotiation. In *Proceedings of the Third International Workshop on Policies for Distributed Systems and Networks (Policy 2002)*, pages 92–103. IEEE Computer Society Press, June 2002.

[139] T. Yu, M. Winslett, and K. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Trans. Inf. Syst. Secur.*, 6(1):1–42, 2003.

[140] F. Zhang and K. Kim. ID-based Blind Signature and Ring Signature from Pairings. In *Advances in Cryptology - ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 533–547. Springer-Verlag, 2002.

[141] Y. Zheng. Digital Signcryption or How to Achieve Cost(Signature and Encryption) << Cost(Signature) + Cost(Encryption). In *Advances in Cryptology - CRYPTO 1996*, volume 1294 of *Lecture Notes in Computer Science*, pages 165–179. Springer-Verlag, 1996.

# Appendix A

# Chapter 5 Security Proofs

In this section we give detailed proofs for full traceability and full anonymity of the ABGS scheme, AAS scheme and DAAS scheme. Furthermore, we give details of the attribute unforgeability proofs for AAS and DAAS scheme. We will start with full anonymity.

## A.1 Full Anonymity

In this section we will discuss the full anonymity of the three schemes proposed in Chapter 5. In each three proofs we show how the existence of an adversary *Adam* that breaks the anonymity of the scheme implies the existence of another adversary *Eve* that breaks a known secure encryption scheme or a complexity assumption. The game models are described in the following three sections: A.1.1, A.1.2 and A.2.3.

### A.1.1 Full Anonymity of an ABGS

**Theorem A.1.1.** *If the linear encryption is IND-CPA secure then the attribute based group signature scheme is anonymous under the random oracle.*

Assume *Adam* is an adversary that breaks the anonymity of the ABGS scheme. We will prove that there is an adversary *Eve* that breaks the IND-CPA security of the linear encryption using *Adam*'s talent. The game is demonstrated below:

- **Setup:** *Eve* is given the public key $LE_{PK} = \langle u, v, h \rangle$ from *Charles* (See Section 2.3.2). *Eve* chooses $\gamma$, $t_1$,...,$t_m \in_R \mathbb{Z}_p^*$. Using the $LE_{PK}$ key and the random values, *Eve* can calculate an ABGS public parameter $S_{pub} = \langle G_1, G_2, G_3, e, g_1, g_2, h, u, v \rangle$ for the ABGS scheme. Note that $H$ will be replaced by a random oracle. *Eve* also calculates $n$ private key bases $bsk[i] = \langle A_i, x_i \rangle$ where $1 \leq i \leq n$.

- **Phase 1:** *Eve* runs five oracles: a Signature oracle, PriKey oracle, PubKey oracle, AttKey oracle and a Hash oracle.
  The Hash oracle has a list that saves a unique random value for each nine-element tuple queried and the random value is the response to the query. The Hash oracle should guarantee that no two tuples have the same random value and that each time it responds with the same random value for the same input.
  In the PriKey oracle *Adam* sends an index $i$ and *Eve* responds back with $bsk[i] = \langle A_i, x_i \rangle$.
  The AttKey oracle *Adam* sends a key $A_i$ and $\Upsilon_i$ and *Eve* responds with $gsk = \langle A_i,\ T_{i,1},...,T_{i,\mu} \rangle$.
  The PubKey oracle can be queried by *Adam* where he sends $\Gamma$ and *Eve* adds a $gpk = \langle g_1,\ g_2,\ h,\ u,\ v,\ w,\ \overline{D},\ h_1,...,h_\kappa \rangle$ to a database accessible by *Adam*.
  In the Signature oracle *Adam* sends an index $i$, a random message $M$ and the public key $gpk$ to *Eve*. *Eve* responds back with a signature $\sigma = \langle C_1,\ C_2,\ C_3,\ c,\ CT_1,...,CT_\mu,\ s_\zeta,\ s_\beta,\ s_x,\ s_{\delta_1},\ s_{\delta_2}, td,\ \Im_i \rangle$ on that message from user $i$. $c$ is the response of the Hash oracle for the tuple $\langle M, C_1, C_2, C_3, R_1, R_2, R_3, R_4, R_5 \rangle$. The signature is calculated as done in the construction since *Charles* has the private keys required.

- **Challenge:** Now *Adam* can request from *Eve* his anonymity challenge by choosing two indexes ($i_0$ and $i_1$), set of attributes $\Im_i$, a public key $gpk$ and a message $M$ asking for a signature of one of them. *Eve* sends *Charles* both $\langle A_{i_0}, A_{i_1} \rangle$ as messages requesting her challenge. *Charles* responds back with the ciphertext $\bar{C} = \langle C_1, C_2, C_3 \rangle$ of $A_{i_b}$ where $b \in \{0, 1\}$. *Eve* generates a signature by choosing $\alpha,\ s_\zeta,\ s_\beta,\ s_x,\ s_{\delta_1},\ s_{\delta_2},\ c \in_R \mathbb{Z}_p^*$ then calculating :
  $\bar{R_1} = u^{s_\zeta} C_1^{-c}$,
  $\bar{R_2} = v^{s_\beta} C_2^{-c}$,
  $\bar{R_4} = C_1^{s_x} u^{-s_{\delta_1}}$,
  $\bar{R_5} = C_2^{s_x} v^{-s_{\delta_2}}$,
  $\bar{R_3} = e(C_3, g_2)^{s_x}.e(h, w)^{-s_\zeta - s_\beta}.e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}}.(\frac{e(C_3,w)}{e(g_1,g_2)})^c$.
  *Eve* adds the tuple $\langle M, C_1, C_2, C_3, R_1, R_2, R_3, R_4, R_5 \rangle$ with $c$ to the list of Hash oracles and calculates $td = w^\alpha$.
  *Eve* sends signature $\sigma_b = \langle C_1, C_2, C_3, C_3^{\alpha/t_1}, ..., C_3^{\alpha/t_\tau}, s_\zeta,\ s_\beta,\ s_x,\ s_{\delta_1},\ s_{\delta_2},\ c,\ td,\ \Im_i \rangle$ and sends it to *Adam*.

- **Phase 2:** *Adam* goes back to issuing further queries as done in Phase one.

- **Guess:** *Adam* returns a $\bar{b}$ to *Eve*.

*Eve* outputs $\bar{b}$ as her answer to *Charles*. *Eve* has a high advantage on guessing the right $\bar{b} = b$ if and only if *Adam* can break into the anonymity of the ABGS scheme.

### A.1.2 Full Anonymity of an AAS

**Theorem A.1.2.** *If the decision linear assumption holds in group $G_2$ then the AAS is 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. The resulting game is described below:

- **Init:** *Adam* decides the universe 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 $u_0,\ u_1,\ u_2,\ h_0,\ h_1,\ Z \in G_2$ 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 $\gamma \in_R \mathbb{Z}_p^*$ 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 5.4.3. 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* 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 $\overline{D}$ for a certain attribute tree $\Gamma$, 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 5.4.3. 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:*

$$\textit{Fake-Sign}_{Node} = \begin{cases} \textit{If } (j \in \Gamma)\textit{; return } e((u_2^s h_0^t u_0^{st})^{t_j}, D_j)^\beta \\ \qquad\qquad\qquad\qquad = e(u_2^s h_0^t u_0^{st}, w)^{\beta q_j(0)} \\ \textit{Otherwise return } \perp \end{cases}$$

*$F_{root}$ in this case will equal $e(u_2^s h_0^t u_0^{st}, w)^{\beta\alpha}$. Notice that $F_{root}^{1/\alpha}.C_3 = e(C_2, w)^\beta$. Given that $\beta_1, \beta_2 \in_R \mathbb{Z}_p^*$, 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* $\langle e(ZW, w)^{\beta_2}, w^{\beta_2}, e(u_2^s h_0^t u_0^{st}, w)^{\beta_2\alpha}\rangle$.

*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:*

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

*$F_{root}$ 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 $\beta_1, \beta_2$ are random elements in $\mathbb{Z}_p^*$, it is hard to distinguish between the triples:*
$\langle e(v^\xi, w)^{\beta_1}, w^{\beta_1}, e(A_i, w)^{\beta_1\alpha}\rangle$ *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 values $r$, $c$, $s_\xi$, $s_x$, $s_\delta \in_R \mathbb{Z}_p^*$. 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 $H_0(\overline{D}, M, r)$ 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* $H_0(\overline{D}, M, r) = (\bar{u}, \bar{v})$ *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 $\overline{D}$ for a certain $\Gamma$ 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 $\sigma_b$.*

- ***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.

### A.1.3 Full Anonymity of a DAAS

**Theorem A.1.3.** *If ElGamal encryption scheme is IND-CPA secure then the Dynamic Attribute Authentication Scheme is anonymous under the random oracle.*

*Adam* is an adversary that attacks the schemes anonymity, *Eve* tries to use *Adam*'s capability in order to break the IND-CPA security of ElGamal encryption scheme. The following is the game model:

- **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* sets up the ElGamal Encryption scheme. The public key is $(g_1, g_4) \in G_1$ where $g_1 = g_4^{\xi_1}$ and $\xi_1 \in \mathbb{Z}_p^*$. $\xi_1$ is kept secret to the challenger while the rest is public and known to *Eve*. *Eve* calculates $g_2 = g_1 g_4^{rnd_1}$ therefore $\xi_2 = \xi_1 + rnd_1$ where $rnd_1 \in \mathbb{Z}_p^*$. *Eve* chooses a $\gamma \in \mathbb{Z}_p^*$, and $h \in G_2$. *Eve* can calculate *gpk* and send it to *Adam* together with $\gamma$.

- **Phase 1:** In this phase *Adam* queries the oracles: USK, a Signature oracle, CrptJoinUsr, CrptJoinIss, Open and the Hash oracle.
  In the Hash oracle *Eve* responses with a unique but random $c \in \mathbb{Z}_p^*$ every time the query of the tuple $(M, C_1, C_2, C_3, C_4, C_5, C_6, R_1, R_2, R_3, R_4)$ takes place. By unique we mean for the same input response is always the same and by random we mean the response is different and random for other inputs. A list of responses is kept for such purposes.
  Replies to the rest of the oracles are straightforward except for the Open oracle. The reason is that *Eve* has the issuing key $\gamma$, the master keys $t_1, ..., t_m$ and the *gpk* that are needed in the oracles but she does not have the tracing keys $\xi_1$ and $\xi_2$. Therefore all oracles are run exactly as done in the main scheme except for the open oracle.
  To respond to the open oracle *Eve* will use the list of registration keys she has and $rnd_1$. For every element in the list $A^*$ check the following equality and if it holds then $A^* = A_i$

$$\frac{F_{root}^{1/\alpha} C_5}{e(A^*, C_6) e(C_3^{rnd_1}, C_6)} = e(\frac{C_4}{C_3^{rnd_1} A^*}, C_6)$$

  Note that such an equality can not be checked by *Adam* even if he has the lists of all $A^*$ because the element $rnd_1$ is used and is only known to the challenger.

- **Challenge:** *Adam* decides on a message $M$, two indexes $(i_0, i_1)$ and a verification key $\overline{D}$ in which he would like to be challenged on. *Eve* sends $A_{i_0}, A_{i_1}$ as two messages to challenge *Charles* with. *Charles* encrypts one of them and returns ciphertext $(C_1 = g_4^{\zeta}, C_2 = A_b g_1^{\zeta})$. Note that *Eve* has to guess $b$. *Eve* can simulate a signature by calculating $C_4 = C_2 C_1^{rnd_1} g_2^{rnd_2}$ and $C_3 = C_1 g_4^{rnd_2}$. Given that $\delta = \zeta + rnd_2$, then $C_4 = A_b g_2^{\delta}$ and $C_3 = g_4^{\delta}$. *Eve* chooses randomly $s_\zeta$, $s_\delta$, $s_x$, $s_z$ and $c$ from $\mathbb{Z}_p^*$. Note that $c$ should have not been a response to a query to the Hash oracle. She calculates $R_1 = g_4^{s_\zeta} C_1^{-c}$, $R_3 = g_4^{s_\delta} C_3^{-c}$, $R_4 = g_1^{s_\zeta} g_2^{-s_\delta} / (C_2 C_4^{-1})^c$ and $R_2 = e(C_2, h)^{s_x} e(g_1, w)^{-s_\zeta} e(g_1, h)^{-s_z} (\frac{e(C_2, w)}{e(g_3, h)})^c$. Finally *Eve* creates $F_{root}$ with $T_{i,j} = (C_2 C_1^{rnd_1})^{1/t_j}$ therefore $F_{root} = e(C_2 C_1^{rnd_1}, w^{\beta})^{\alpha}$ for some random $\beta \in \mathbb{Z}_p^*$. $C_6 = w^{\beta}$ and $C_5 = e(g_2^{rnd_2}, w^{\beta})$.

- **Phase 2:** This phase is similar to Phase 1 except that $\sigma_b$ can not be queried in the open oracle.

- **Output:** *Adam* outputs a guess $\bar{b} \in \{0, 1\}$.

*Eve* can respond to *Charles* with her guess being $\bar{b}$.

## A.2 Full Traceability

Sections A.2.1, A.2.2 and A.2.3 prove the constructions in Chapter 5 fully traceable. The proofs are very similar. All three models have a Setup, Queries and Output phase. Setup is for creating all required attributes of the game, while Queries is a stage where *Adam* can access certain oracles, and Output is when the *Adam* decides he can forge a signature and sends it to *Charles* to verify its correctness and traceability. If *Charles* accepts it as a forged signature *Adam* wins the game.
We need three steps in all traceability games in this section. We start with defining a security model for proving full-traceability, then introducing two types of signature forger, and then we show that the existence of such forgers implies that q-SDH is easy. Suppose we are given an adversary *Adam* that breaks the full traceability of the signature scheme. *Charles* is the challenger. The following three sections show how the existence of such an adversary solves the q-SDH problem.

### A.2.1 Full Traceability of an ABGS

**Theorem A.2.1.** *If q-SDH is hard on groups $G_1$ and $G_2$ then the ABGS scheme is fully traceable under the random oracle.*

Recall the first step is to define a security model as an interacting framework between *Charles* and *Adam* as follows:

- **Init:** *Adam* decides the universal set of attributes he would like to be challenged on and that is by sending *Charles* the master keys $t_1,...,t_m \in_R \mathbb{Z}_p^*$.

- **Setup:** *Charles* runs the setup algorithm as in Section 5.3.3 with a bilinear pair $e : G_1 \times G_2 \rightarrow G_3$. Let $g_1$,$u$,$v$,$h \in G_1$, $g_2 \in G_2$ and $\xi_1$, $\xi_2$, $\gamma \in \mathbb{Z}_p^*$ such that they all satisfy properties mentioned in Section 5.3.3. *Charles* is given the pairs $\langle A_i, x_i \rangle$ for an $i = 1, ..., n$. Some of those pairs have $x_i = \star$ which implies that $x_i$ corresponding to $A_i$ is not known; Other pairs are valid SDH pairs. *Adam* is given the tracing keys $(\xi_1, \xi_2)$.

- **Queries:** There are four oracles that *Adam* can access. In the Hash oracle *Adam* asks *Charles* for the hash of ($M$, $C_1$, $C_2$, $C_3$, $R_1$, $R_2$, $R_3$, $R_4$, $R_5$). *Adam* responds with a random element $c \in G_1$ and saves the answer just in case the same query is requested again (As done in Section A.1.1 in the ABGS anonymity game model).
  In the Signature oracle *Adam* asks for a signature on a message $M$ by a member $i$ over a set of attributes $\Im_i$ using a public key *gpk*. If $x_i \neq \star$, *Charles* calculates $T_{i,j} = A_i^{1/t_j}$ for all attributes in $\Im_i$ and signs the message normally to obtain

$\sigma$ and give it to *Adam*. If $x_i = \star$ then *Charles* picks $\zeta, \beta, \alpha \in_R \mathbb{Z}_p^*$ sets $C_1 = u^\zeta, C_2 = v^\beta, C_3 = A_i g_1^{\zeta+\beta}$, and $CT_j = (A_i g_1^{\zeta+\beta})^{\alpha/t_j}$ for every attribute in $\Im_i$. Now *Charles* can get $\sigma$ by choosing $\alpha$, $s_\zeta$, $s_\beta$, $s_x$, $s_{\delta_1}$, $s_{\delta_2}$, $c \in_R \mathbb{Z}_p^*$ then calculating:
$\bar{R_1} = u^{s_\zeta} C_1^{-c}$,
$\bar{R_2} = v^{s_\beta} C_2^{-c}$,
$\bar{R_4} = C_1^{s_x} u^{-s_{\delta_1}}$,
$\bar{R_5} = C_2^{s_x} v^{-s_{\delta_2}}$,
$\bar{R_3}$=$e(C_3, g_2)^{s_x}.e(h, w)^{-s_\zeta - s_\beta}.e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}}.(\frac{e(C_3,w)}{e(g_1,g_2)})^c$.
*Charles* adds the tuple $\langle M, C_1, C_2, C_3, R_1, R_2, R_3, R_4, R_5\rangle$ with $c$ to the list of Hash oracles and calculates $td = w^\alpha$.
*Charles* sends signature $\sigma_b$=$\langle C_1$,$C_2$,$C_3$,$C_3^{\alpha/t_1}$,...,$C_3^{\alpha/t_\tau}$,$s_\zeta$, $s_\beta$, $s_x$, $s_{\delta_1}$, $s_{\delta_2}$, $c$, $td$, $\Im_i\rangle$ and sends it to *Adam*.
In the PriKey oracle *Adam* asks for the private key in a certain index $i$ and *Charles* replies with $bsk[i]$.
In the AttKey oracle *Adam* sends $A_i$ and an attribute set $\Upsilon_i$. If $x_i \neq \star$, *Charles* returns back $\langle T_{i,1}$,...,$T_{i,\tau}\rangle$ where $T_{i,j} = A_i^{1/t_j}$; otherwise *Charles* declares failure.

- **Output:** If *Adam* is successful, he outputs a forged signature on a message $M$ corresponding to any public key *gpk*. The signature should verify correctly yet not trace to a member that has been queried. *Charles* runs the verify then the open algorithm. He then tests the $A^*$ he calculated through the open algorithm. If $A^* \neq A_i$ for all $i$ output $\sigma$. If $A^* = A_{i^*}$ for some $i^*$and if $s_{i^*} = \star$ output $\sigma$. The only possibility left is having $A^* = A_{i^*}$ but $s_i \neq \star$ *Charles* declares failure.

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 $A^* = A_{i^*}$ 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:** Considering Theorem 2.2.13 for a $(n+1)$ SDH, *Charles* can obtain $g_1$,$g_2$ and $w$ and also use the $n$ pairs $(A_i, x_i)$ to calculate the private keys $\langle A_i$, $x_i$, $A_i^{1/t_1}$,..., $A_i^{1/t_\mu}\rangle$. *Charles* uses these values in interacting with *Adam*. *Adam*'s success leads to forgery of Type-I.

**Type-II:** Using the same Theorem 2.2.13 but for an $n$ SDH this time, *Charles* obtains $g_1$, $g_2$ and $w$ and uses the $n-1$ pairs $(A_i, x_i)$ to calculate the private keys $\langle A_i$,$x_i$,$A_i^{1/t_1}$,...,$A_i^{1/t_\mu}\rangle$. In a random index $i^*$, *Charles* can choose the missing pair randomly where $A_{i^*} \in G_1$ and set $x_{i^*} = \star$. The random private key will be $\langle A_{i^*}$, $x_{i^*}$, $A_{i^*}^{1/t_1}$,...,$A_{i^*}^{1/t_\mu}\rangle$. *Adam* in the security model will fail if he queries the PriKey oracle in index $i^*$. Other private key queries will succeed. In the Signature oracle and because of the Hash oracle, it will be hard to distinguish between signatures with a SDH pair

and ones without.

The next step is showing how the Forking Lemma (Definition 3.4.1) can be applied here to prove that we can generate new SDH pairs if a forgery of any type exists. Let *Adam* be a forger of any type in which the security model succeeds. A signature will be represented as $\langle M, \sigma_0, c, \sigma_1, \sigma_2 \rangle$. $M$ is the signed message. $\sigma_0 = \langle C_1,\ C_2,\ C_3,\ R_1,\ R_2,\ R_3,\ R_4,\ R_5 \rangle$. $c$ is the value derived from hashing $\sigma_0$. $\sigma_1 = \langle s_\zeta,\ s_\beta,\ s_x,\ s_{\delta_1},\ s_{\delta_2} \rangle$ which are values used to calculate the missing inputs for the hash function. Finally $\sigma_2 = \langle CT_1,...,CT_\tau, \Im_i \rangle$ the values that depend on the set of attributes in each Signature oracle.

According to the Forking Lemma, having a replay of this attack with the same random tape but a different response of the random oracle, implies a new signature $\langle \sigma_0, \grave{c}, \grave{\sigma}_1, \sigma_2 \rangle$ can be created.

Now we show how we can 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 $\Delta c = c - \grave{c}$, $\Delta s_\zeta = s_\zeta - \grave{s}_\zeta$, and similarly for $\Delta s_\beta$,$\Delta s_x$,$\Delta s_{\delta_1}$, and $\Delta s_{\delta_2}$.

Divide two instances of the equations used previously in proving Theorem 5.3.4 where one instance is with $\grave{c}$ and the other is with $c$ to get the following:

- Dividing $R_1/\grave{R}_1$ we get
  $u^{\tilde{\zeta}} = C_1$; where $\tilde{\zeta} = \Delta s_\zeta / \Delta c$

- Dividing $R_2/\grave{R}_2$ we get
  $v^{\tilde{\beta}} = C_2$; where $\tilde{\beta} = \Delta s_\beta / \Delta c$

- Dividing $C_1^{s_x}/C_1^{\grave{s}_x} = u^{s_{\delta_1}}/u^{\grave{s}_{\delta_1}}$ will lead to
  $\Delta s_{\delta_1} = \tilde{\zeta} \Delta s_x$

- Dividing $C_2^{s_x}/C_2^{\grave{s}_x} = v^{s_{\delta_2}}/u^{\grave{s}_{\delta_2}}$ will lead to
  $\Delta s_{\delta_2} = \tilde{\beta} \Delta s_x$

- Calculating the following equality:
  $(e(g_1, g_2)^{\Delta c}/e(C_3, w))$
  $= e(C_3, g_2)^{\Delta s_x}.e(h, w)^{-\Delta s_\zeta - \Delta s_\beta}.e(h, g_2)^{-\Delta s_{\delta_1} - \Delta s_{\delta_2}}$
  $= e(C_3, g_2)^{\Delta s_x}.e(h, w)^{-\Delta s_\zeta - \Delta s_\beta}.e(h, g_2)^{-\tilde{\zeta} \Delta s_x - \tilde{\beta} \Delta s_x}$

From the equations above if we let $\tilde{x} = \Delta s_x / \Delta c$ and $\tilde{A} = C_3 h^{-(\tilde{\zeta} + \tilde{\beta})}$ we get the following equation:

$e(g_1, g_2)/e(C_3, w) = e(C_3, g_2)^{\tilde{x}}.e(h, w)^{-\tilde{\zeta} - \tilde{\beta}} e(h, g_2)^{-\tilde{x}(\tilde{\zeta} + \tilde{\beta})}$

$e(g_1, g_2)$=$e(\tilde{A}, wg_2^{\tilde{x}})$
Hence we obtain a new SDH pair $(\tilde{A}, \tilde{x})$ breaking Boneh and Boyen's Lemma (Definition 2.2.13).

### A.2.2 Full Traceability of an AAS

**Theorem A.2.2.** *If q-SDH is hard on groups $G_1$ and $G_2$ then the AAS is fully traceable under the random oracle.*

We start with explaining the security model as an interacting framework between *Charles* and *Adam* as follows:

- **Init:** *Adam* decides on the universal set of attributes $U = t_1,...,t_m \in_R \mathbb{Z}_p^*$, 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 2.2.11). *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:
  In the 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 $\langle A_i, x_i \rangle$.
  As for the Hash oracle, *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 $(\overline{D}, M, r)$, *Charles* responds with two random elements in $G_2$ and saves the answer.
  In the Signature oracle, *Adam* runs the $\overline{D} = TCreate(\Gamma, \alpha, \overline{B})$ for a $\alpha \in_R \mathbb{Z}_p^*$ and a set of attribute public keys $\overline{B}$. He then sends $\overline{D}$ 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 5.4.3.

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

and $\beta \in_R \mathbb{Z}_p^*$.

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 picks $c$, $s_\delta$, $s_x$, and $s_\xi \in_R \mathbb{Z}_p^*$. 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 $\overline{D}$ and calculates $e(A_i^{t_j}, D_j)^{\beta}$, this is in place of the recursive $Sign_{Node}$ function in Section 5.2. *Charles* can now calculate $F_{root}$ using the elements it calculated and $\Gamma$.

*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 $\overline{D}$ for a certain $\Gamma$. 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 $\sigma$ or otherwise declare failure. If the for loop goes through all the $(A_i)$'s and no equality is identified output $\sigma$.

As in the ABGS traceability game model in Section A.2.1, there are two types of forgery shown below:

**Type-I:** If we consider Theorem 2.2.13 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.

**Type-II:** Similar to the traceability game in the previous section Theorem 2.2.13 is used for a $(n)$ SDH, *Charles* can obtain $g_1$, $g_2$ and $w$ and uses the $n-1$ pairs $(A_i, x_i)$ to calculate the private key bases $\langle A_i, x_i \rangle$. In a random index $i^*$, the missing pairs are chosen 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.

We shall prove that any of the two forgeries contradict the q-SDH assumption using the Forking Lemma (See Theorem 3.4.1). 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 $\sigma_0$, 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.

According to the Forking Lemma if we have a replay of this attack with the same random tape but a different response of the random oracle [1] we can obtain a signature $\langle \sigma_0, \grave{c}, \grave{\sigma_1}, \sigma_2\rangle$.

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 $\Delta s_x$, and $\Delta s_\delta$.

Divide two instances of the equations used previously in proving correctness of the scheme. 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 Boyen's Lemma (See Theorem 2.2.13).

### A.2.3 Traceability of a DAAS Scheme

**Theorem A.2.3.** *If q-SDH is hard on groups $G_1$, and $G_2$ then the Dynamic Attribute Based Authentication Scheme is fully traceable under the random oracle.*

Similar to the previous two sections and traceability games, 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 $h \in G_2$. He is also given a value $w = h^\gamma$ and $n$ SDH pairs $\langle O_i, x_i\rangle$

[1] In the AAS scheme we mean the random oracle of the hash function $H$ rather than $H_0$

for an $1 \leq i \leq n$, which he will be using in creating private key bases. Some of those pairs have $x_i = \star$ which implies that $x_i$ corresponding to $O_i$ is not known; Other pairs are valid SDH pairs (Definition 2.2.11). Assume users $i = 1,...,n_1$ are the list of honest users that can be corrupted by querying the USK oracle and $i = n_1 + 1, ..., n_2$ are the ones for dishonest user where the adversary runs the join protocol with *Charles*. Note that $n = n_1 + n_2$.
*Charles* chooses $\xi_1, \xi_2 \in \mathbb{Z}_p^*$ and $g_4, g_2 \in G_1$ such that $g_1 = g_4^{\xi_1}$ and $g_2 = g_4^{\xi_2}$. Finally *Charles* sets $g_3 = g_1^{rnd_1}$ for some random $rnd_1 \in \mathbb{Z}_p^*$. *Charles* sends $w$, and $gpk = \langle G_1, G_2, G_3, e, g_1, g_2, g_3, g_4, h \rangle$. The hash function $H$ is 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 $\xi_1$, and $\xi_2$.

- **Queries:** *Adam* queries the oracles USK, Signature, CrptJoinUser and Hash as follows:

  In the USK Oracle, *Adam* asks for a certain private key by sending *Charles* an index $1 \leq i \leq n_1$. If *Adam* queries an index where $x_i = \star$ abort the game and declare failure; Otherwise he chooses a random $y_i \in \mathbb{Z}_p^*$ and calculates $A_i = O_i^{y_i} O_i^{rnd}$. The private key $\langle A_i, x_i, y_i \rangle$ is sent to the adversary.

  The Hash oracle is queried when *Adam* asks *Charles* for the hash of $(M, C_1, C_2, C_3, C_4, C_5, C_6, R_1, R_2, R_3, R_4)$, *Charles* responds with a random element in $\mathbb{Z}_p^*$ and saves the answer just in case the same query is requested again. This represents the hash function $H$.
  The CrptJoinUser presents *Adam* and *Charles* engaging in a join protocol where *Adam* resembles a user $i$ and *Charles* the issuer manager. If $i = *$ abort else run the join protocol. The join protocol is similar to the one in the construction with one exception where *Charles* in the first step sends $O_i^{rnd_2}$. This change helps *Charles* to generate private keys from the $O_i$ he has rather than $\gamma$ since he does not know it. He can now obtain $O_i^{y_i}$ and calculate from that the key $\langle A_i, x_i, y_i \rangle$ as in the USK oracle. The rest of the protocol runs normally.
  In the Signature oracle, *Adam* runs the $\overline{D} = TCreate(\Gamma, \alpha, \overline{B})$ for a random $\alpha \in \mathbb{Z}_p^*$ and a set of attribute public keys $\overline{B}$. He then sends $\overline{D}$ 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 5.4.3.

  If $x_i = \star$, *Charles* simulates a signature. He chooses the random elements $s_\zeta$, $s_\delta$, $s_x$, $s_z$, $\zeta$, $\delta$, $\beta_1$, $\beta_2$ and $c$, all belong to $\mathbb{Z}_p^*$. Let $\beta = \beta_1 + \beta_2$. He calculates $C_1 = g_4^\zeta$, $C_2 = A_i g_1^\zeta$, $C_3 = g_4^\delta$, $C_4 = A_i g_2^\delta$, $C_5 = e(g_2^\delta A_i^{1-\beta_2}, w_1^\beta)$ and finally $C_6 = w^{\beta_1}$.

*Charles* also computes $R_1 = g_4^{s_\zeta}$, $R_3 = g_4^{s_\delta} C_3^{-c}$, $R_4 = g_1^{s_\zeta} g_2^{-s_\delta}/(C_2 C_4^{-1})^{-c}$ and finally $R_2 = e(C_2, h)^{s_x} e(g_1, w)^{-s_\delta} e(g_1, h)^{-s_z} (\frac{e(C_2,w)}{e(g_3,h)})^c$. *Charles* adds $c$ to the list of responses in the hash function.
*Charles* can calculate $F_{root}$ as done in the main scheme since he has all master keys needed.
Signature is $\sigma = (C_1, C_2, C_3, C_4, C_5, C_6, F_{root}, c, s_\zeta, s_\delta, s_x, s_z)$

- **Output:** *Adam* asks to be challenged and sends *Charles* a message $M$. *Charles* responds with a $\overline{D}$ for a certain $\Gamma$. If *Adam* is successful he will output a signature $\sigma = (r, C_1, C_2, C_3, C_4, C_5, C_6, c, s_\xi, s_x, s_\delta, F_{root})$ for a message $M$. Let $A_i^*$ be the value used in signing the forged signature. For $i = 1, ..., n$, *Charles* checks whether $A_i^* = (C_2/(C_1)^{\xi_1}) = (C_4/(C_3)^{\xi_2})$. If the equality holds then this implies that $A_i^* = A_i$. In that case check if $s_{i^*} = \star$ to output $\sigma$ or otherwise declare failure. If the loop goes through all the $(A_i)$'s and no equality is identified output $\sigma$.

There are two types of forgery. Type-I outputs a signature that can be traced to some identity which is not part of $O_{i_0}$,...,$O_{i_n}$. Type-II has $A_i^* = O_i$ where $1 \leq i \leq n$ but *Adam* did not submit a query of $i$ to the USK oracle nor did he participate in the join protocol using it. We prove both forgeries are hard.

**Type-I:** If we consider Theorem 2.2.13 for a $(n+1)$ SDH, we can obtain $g_1$,$g_2$ and $w$. We can also use the $n$ pairs $(O_i, x_i)$ to calculate the private key bases $\langle A_i, x_i, y_i \rangle$. These values are used when interacting with *Adam*. *Adam*'s success leads to forgery of Type-I.

**Type-II:** Using Theorem 2.2.13 once again but for a $(n)$ SDH, we can obtain $g_1$, $g_2$ and $w$. Then we use the $n-1$ pairs $(O_i, x_i)$ to calculate the private key bases $\langle A_i, x_i, y_i \rangle$. In a random index $i^*$, we choose the missing pair randomly where $O_{i^*} \in G_1$ and set $x_{i^*} = \star$. *Adam*, in the security model, will fail if he queries the USK oracle with index $i^*$ or use it in the corrupted join protocol. 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.

A signature will be represented as $\langle M$,$\sigma_0$,$c$,$\sigma_1$,$\sigma_2 \rangle$, $M$ is the signed message, $\sigma_0 = \langle r$,$C_1$,$C_2$,$C_3$,$C_4$,$C_6$, $R_1$, $R_2$, $R_3$, $R_4 \rangle$, $c$ is the value derived from hashing $\sigma_0$, and $\sigma_1 = \langle s_\zeta$,$s_\delta$, $s_x$, $s_z \rangle$ 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.
According to the Forking Lemma if we have a replay of this attack with the same random tape but a different response of the random oracle we can obtain a signature $\langle \sigma_0, \grave{c}, \grave{\sigma}_1, \sigma_2 \rangle$.

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 $\Delta s_\zeta = s_\zeta - \tilde{s}_\zeta$, and similarly for $\Delta s_x$, $\Delta s_\delta$, $\Delta s_x$ and $\Delta s_z$. Divide two instances of the equations used previously in proving correctness of the scheme. One instance with $\tilde{c}$ and the other with $c$ to obtain the following:

- Dividing $C_1^c / C_1^{\tilde{c}} = g_4^{s_\zeta} / g_4^{\tilde{s}_\zeta}$ we get $g_4^{\tilde{\zeta}} = C_1$; where $\tilde{\zeta} = \Delta s_\zeta / \Delta c$
- Dividing $C_2^{s_\delta} / C_2^{\tilde{s}_\delta} = g_4^{s_\delta} / g_4^{\tilde{s}_\delta}$ we get $g_4^{\tilde{\delta}} = C_2$; where $\tilde{\delta} = \Delta s_\delta / \Delta c$
- The division of $(C_2 C_4^{-1})^{\Delta c} = g_1^{\Delta s_\zeta} g_4^{\Delta s_\delta}$ implies $(C_2 C_4^{-1}) = g_1^{\tilde{\zeta}} g_4^{\tilde{\delta}}$
- The division of $e(C_2, h)^{\Delta s_x} e(g_1, w)^{-\Delta s_\zeta} e(g_1, h)^{-\Delta s_z} = (\frac{e(g_3,h)}{e(C_2,w)})^{\Delta c}$ leads to $e(C_2, h)^{\tilde{x}} e(g_1, w)^{-\tilde{\zeta}} e(g_1, h)^{-\tilde{z}} = (\frac{e(g_3,h)}{e(C_2,w)})^{\Delta c}$ for $\tilde{x} = \Delta s_x / \Delta c$ and $\tilde{z} = \Delta s_z / \Delta c$

If $\tilde{A} = C_2 g_1^{\tilde{\zeta}}$ and $\tilde{y} = \tilde{z} - \tilde{\zeta}\tilde{x}$ then from the last division we get $e(\tilde{A}, h)^{\tilde{x}} e(\tilde{A}, w) = e(g_3, h) e(g_1, h)^{\tilde{y}}$ this implies we have obtained a certificate $(\tilde{A}, \tilde{x}, \tilde{y})$ where $\tilde{A} = (g_3 g_1^{\tilde{y}})^{1/(\tilde{x}+\gamma)}$. This leads to a SDH pair $(\mathcal{A}, \mathcal{X})$ and breaking Boneh and Boyen's Lemma (See Theorem 2.2.13). Knowing that $\tilde{A} = (g_3 g_1^{\tilde{y}})^{1/(\tilde{x}+\gamma)} = (g_1^{\tilde{y}+rnd_1})^{1/(\tilde{x}+\gamma)}$.
Calculate $(\mathcal{A}, \mathcal{X}) = (\tilde{A}^{1/(\tilde{y}+rnd_1)}, x_i)$.

## A.3 Attribute Unforgeability

In this section we prove the AAS scheme and DAAS scheme constructed in Chapter 5 are attribute-unforgeable. In both proofs we explain the game model in detail and show how *Adam* can create a list of information about the attribute he lacks but he can not forge it without breaking the DLP.

### A.3.1 Unforgeability of Attributes in AAS

**Theorem A.3.1.** *Breaking the Unforgeability of Attributes in the AAS construction is as hard as solving the DLP.*

The following shows details of the game model defined in Section 5.4. Following that is a table presenting all the information *Adam* can obtain within the game. We later prove that the information is not enough to break the unforgeability of attributes without solving the DLP.

- **Setup:** *Charles* sets up the system. He generates $S_{pub}$ and $S_{pri}$. He also creates a set of private key bases $bsk[i] = \langle A_i, x_i \rangle$. He chooses the set of master keys $t_j$ for every attribute $j$ and calculates the attribute public keys $bpk_j$. He sends $S_{pub}$ to *Adam* along with all $A_i$ and $bpk_j$.

- **Phase 1:** *Adam* queries the following oracles:
  - Signature Oracle as described in section 5.4. Note that *Charles* has all private keys and attribute master keys he needs in order to create a valid signature $\sigma$ and send it to *Adam*.
  - USK Oracle as described in section 5.4. *Charles* has given *Adam* all the $A_i$ but not $bsk[i] = \langle A_i, x_i \rangle$
  - AttPriKey Oracle as described in section 5.2 where *Adam* can send a key $A_i$ and an index $j$ and in return he gets $T_{i,j}$.
  - AttMasKey Oracle as described in section 5.2 where *Adam* sends an attribute index $j$ to get the master key $t_j$.

  Notice that the TVfy Oracle in Section 5.2 is not used here because the Signature Oracle is sufficient enough (i.e runs the $TVerify$ algorithm). Furthermore the Revoke oracle of section 5.4 is not required since all $A_i$ are given to *Adam* and he can run the open algorithm himself.

- **Challenge:** *Adam* sends a tree $\Gamma_1$, user $l$ and attribute $z$ which he would like to be challenged on. *Charles* replies with $\overline{D}$ for a tree $\Gamma_2$ where $\Gamma_2$ has two subtrees the first is $\Gamma_1$ and the other is based on $t_z$. The threshold value of the root in $\Gamma_2$ is 2. The challenge condition is that user $l$ has not been queried in AttPriKey for the attribute $z$. Furthermore the challenged index $z$ should not have been queried in AttMasKey. These two conditions are reasonable as they contradict the purpose of the game.

- **Phase 2:** This phase is similar to Phase 1 as long as the challenge conditions are not broken.

- **Output:** *Adam* outputs a signature $\sigma$ for the user $l$ on the verification key $\overline{D}$. If that signature is valid then the adversary wins otherwise *Adam* loses.

From the traceability game one can conclude that the signature created in the output is for a user who has been queried in the USK oracle. Therefore *Adam* can easily create the elements of the signature $\sigma = (r, C_1, C_2, C_3, C_4, c, s_\xi, s_x, s_\delta)$ since we know he has obtained $bsk[i]$. The table A.3.1 shows elements that *Adam* has gained through his queries. Our approach to show that *Adam* can not create a signature with the missing attribute $z$ is similar to the proof in Section 5.2.
Let the root polynomial be $q_r(x)$. The subtree $\Gamma_1$ has a root with the polynomial $q_1(x)$ and the other child holding attribute $t_z$ has a polynomial $q_2(x)$. Further let $q_r(0) = \alpha$, $q_1(0) = q_r(x_1) = y_1$ and $q_2(0) = q_r(x_2) = y_2$. The root polynomial is of degree 1 since the threshold gate is 2. This implies that *Adam* knows that Lagrange is applied therefore the following formula must hold:

Table A.1: Information Obtained by *Adam*

| Information | Source |
|---|---|
| $S_{pub} = \langle G_1, G_2, G_3, e, H, H_0, g_1, g_2 \rangle$ | Setup Phase |
| List of $A_i$ | Setup Phase |
| List of $bpk_j = w^{t_j}$ | Setup Phase |
| List of $bsk[i] = \langle A_i, x_i \rangle$ | USK Oracle |
| 2D Array of $T_{i,j} = A_i^{1/t_j}$ where $i \neq l$ and $j \neq z$ | AttPriKey Oracle |
| List of $\sigma = (r, C_1, C_2, C_3, C_4, c, s_\xi, s_x, s_\delta, F_{root})$ | Signature Oracle |
| List of $t_j$ where $j \neq z$ | AttMasKey |
| List of $\overline{D} = \langle bpk_1,...,bpk_\kappa \rangle$ | Challenge |

$F_{root} = e(A_l, w)^{\alpha\beta} = (e(A_l, w)^{y_2.x_2} e(A_l, w)^{-y_1.x_2})^{\beta/(x_1 - x_2)}$
*Adam* also knows the value of elements ($x_1$, $x_2$, $\beta$, $e(A_l, w)^{y_2}$, $A_l$, $w$, $w^{t_z}$ ).
*Adam* does not know $e(A_l, w)^{\alpha}$, $e(A_l, w)^{y_1}$, $\alpha$, $y_1$, and $t_z$.
Note that values $x_1$, $x_2$, $y_1$, $y_2$, and $\alpha$ change in each round as $\overline{D}$ is created, including in the challenge. $\alpha$ does not appear explicitly or implicitly in any of the elements *Adam* has obtained since it is random each time a $\overline{D}$ is created and that includes the challenge. This implies that $F_{root}$ can be calculated only by deriving the term $e(A_l, w)^{-y_1.x_2}$. Recall that $y_1$ is random each time $\overline{D}$ is calculated. It appears within the game just once and that is in the challenge in $D_k = bpk_k^{y_1} = w^{t_z y_1}$ where $y_1$ is totally bounded with $t_z$ since neither $t_z$ nor $A_l^{1/t_z}$ are known.

### A.3.2 Unforgeability of Attributes in the DAAS scheme

**Theorem A.3.2.** *Breaking the Unforgeability of Attributes in the DAAS construction is as hard as solving the DLP.*

The following shows details of the game model defined in Section 5.5. Following that is a table presenting all the information *Adam* can obtain within the game. We later prove that the information is not enough to break the unforgeability of attributes without solving the DLP.

- **AFDAAS.Setup:** *Charles* sets up the system. He generates the tracing key $tk$, the issuer key $isk$, and the general public key $gpk$. *Charles* plays the role of all attribute authorities in the system. He creates the universal of attributes by choosing a list of master keys $t_1$,...,$t_m$. *Charles* calculates the attribute public keys $bpk_1$,...,$bpk_m$ which he sends together with $tk$ and $gpk$ to *Adam*. *Charles* keeps to himself $isk$ and list of $t_j$.

- **AFDAAS.Phase (1):** *Charles* runs the oracles USK, Signature, CrptJoinUsr, AttPriKey, and AttMasKey. *Adam* can query these oracles in order to obtain information that may help him break the scheme.

- **AFDAAS.Challenge:** *Adam* sends a tree $\Gamma_1$, user $l$ and attribute $z$ in which he would like to be challenged on. *Charles* replies with $\overline{D}$ for a tree $\Gamma_2$ where $\Gamma_2$ has two subtrees: the first is $\Gamma_1$ and the other is based on $t_z$. The threshold value of the root in $\Gamma_2$ is 2. The challenge condition is that user $l$ has not been queried in AttPriKey for the attribute $z$. Furthermore the challenged index $z$ should not have been queried in AttMasKey. These two conditions are reasonable as the contradict with the purpose of the game.

- **AFDAAS.Phase (2):** This phase is similar to Phase 1 as long as the challenge conditions are not broken.

- **AFDAAS.Output:** *Adam* outputs a signature $\sigma$ for the user $l$ on the verification key $\overline{D}$. If that signature is valid then the adversary wins and *Charles* outputs 1 otherwise *Adam* loses and *Charles* outputs 0.

*Charles* can reply to the oracles without problems since he has all private keys needed in creating the outputs such as $tk$, $isk$, and list of $t_j$. The proof for this scheme is similar to the technique used in Section 5.2. From the traceability game we can conclude that for the signature to be valid, the signer had to query either the USK oracle or the CrptJoionUsr oracle by *Adam*. Therefore *Adam* can easily create the elements of the signature $\sigma = (r, C_1, C_2, C_3, C_4, C_5, C_6, c, s_\xi, s_x, s_\delta, s_z, s_\delta)$ since we know he has obtained $bsk[i]$. The table below summarizes the information *Adam* can obtain through the game model The challenge for *Adam* is to create an $F_{root}$ where $F_{root}^{1/\alpha} C_5 = e(C_4, C_6)$.

Table A.2: Information Obtained by *Adam*

| Information | Source |
|---|---|
| $isk = \gamma$ | Setup Phase |
| $gpk = \langle e,\ G_1,\ G_2,\ G_3,\ H,\ g_1,\ g_2,\ g_3,\ g_4,\ h,\ w \rangle$ | Setup Phase |
| List of $A_i$ | Setup Phase |
| List of $bpk_j = w^{t_j}$ | Setup Phase |
| List of $bsk[i] = \langle A_i, x_i, y_i \rangle$ | USK Oracle |
| List of $bsk[i] = \langle A_i, x_i \rangle$ for a $y_i$ chosen by *Adam* | CrptJoinUsr Oracle |
| 2D Array of $T_{i,j} = A_i^{1/t_j}$ where $i \neq l$ and $j \neq z$ | AttPriKey Oracle |
| List of $\sigma = (C_1,\ C_2,\ C_3,\ C_4,\ C_5,\ C_6,\ F_{root},\ c,\ s_\zeta,\ s_\delta,\ s_x,\ s_z)$ | Signature Oracle |
| List of $t_j$ where $j \neq z$ | AttMasKey |
| List of $\overline{D} = \langle bpk_1, \ldots, bpk_\kappa \rangle$ | Challenge |

and $\alpha$ is unknown to *Adam*.

As in the Section A.3.1, let the root polynomial be $q_r(x)$. The subtree $\Gamma_1$ has a root with the polynomial $q_1(x)$ and the other child holding attribute $t_z$ has a polynomial $q_2(x)$. Further let $q_r(0) = \alpha$, $q_1(0) = q_r(x_1) = y_1$ and $q_2(0) = q_r(x_2) = y_2$. The root polynomial is of degree 1 since the threshold gate is 2. This implies that *Adam* knows

that Lagrange is applied therefore the following formula must hold:
$F_{root} = e(A_l, w)^{\alpha\beta} = (e(A_l, w)^{y_2.x_2} e(A_l, w)^{-y_1.x_2})^{\beta/(x_1-x_2)}$
*Adam* also knows the value of elements ($x_1$, $x_2$, $\beta$, $e(A_l, w)^{y_2}$, $A_l$, $w$, $w^{t_z}$ ).
*Adam* does not know $e(A_l, w)^{\alpha}$, $e(A_l, w)^{y_1}$, $\alpha$, $y_1$, and $t_z$.
Note that values $x_1$, $x_2$, $y_1$, $y_2$, and $\alpha$ change each round a $\overline{D}$ is created, including in the challenge. $\alpha$ does not appear explicitly or implicitly in any of the elements *Adam* has obtained since it is random each time a $\overline{D}$ is created and that includes the challenge. This implies that $F_{root}$ can be calculated only by deriving the term $e(A_l, w)^{-y_1.x_2}$. Recall that $y_1$ is random each time $\overline{D}$ is calculated. It appears within the game just once and that is in the challenge in $D_k = bpk_k^{y_1} = w^{t_z y_1}$ where $y_1$ is totally bounded with $t_z$ since neither $t_z$ nor $A_l^{1/t_z}$ are known.