An Access Control Scheme for Partially Ordered Set Hierarchy with Provable Security

Jiang Wu
Department of Computer Science
Lakehead University

March 2005

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Library and Archives Canada
Bibliothèque et Archives Canada

Published Heritage Branch
Direction du Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-494-11195-X
Our file Notre référence
ISBN: 0-494-11195-X

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Preface

In many multi-user information systems, the users are organized as a hierarchy. Each user is a subordinate, superior and/or coordinate of some others. In such systems, a user has access to the information if and only if the information belongs to the user or his/her subordinates. Hierarchical access control schemes are designed to enforce such access policy. In the past years, hierarchical access control schemes based on cryptography are intensively researched. Much progress has been made in improving the schemes' performance and security.

The main contribution of this thesis is a new hierarchical access control scheme. This is the first one that provides strict security proof under a comprehensive security model that covers all possible cryptographic attacks to a hierarchical access control scheme. The scheme is designed and analyzed based on the modern cryptography approach, i.e., defining the security model, constructing the scheme based on cryptography primitives, and proving the security of the scheme by reducing the cryptography primitives to the scheme. Besides the security property, this scheme also achieves good performance in consuming small storage space, supporting arbitrary and dynamic hierarchial structures. In the thesis, we also introduce the background in cryptography and review the previous schemes.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Acknowledgments

I am most grateful to Dr. Wei for his valuable guidance and help throughout my study in Lakehead University. It is a great pleasure to study and work with him.

Thanks to the faculty of the Department of Computer Science and the Department of Mathematical Science. As a student and a teaching assistant, I learned a lot and received a lot of help from them.

Thanks also to David Wagner and Kristian Gjøteen for discussing some tricky problems I posted on the news groups. Their discussion are instructive, and made the research more interesting.

Last but not least, I would like to thank my family for understanding and supporting my career goal and effort.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Contents

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Figures

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Tables

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Algorithms

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 1

Introduction

1.1 Problem Statement

In a multi-user information system, the access permission to certain information objects is usually only granted to certain users. To enforce the access policy, secret values for certain objects is assigned to the users with access privilege. The secret values may be the password used to authenticate the users for accessing to the objects, or the cryptographic key for the users to decrypt the encrypted data to recover the original information. In either case, the access control to the protected information relies on the secret values assigned to the users. For simplicity, we call the secret values a key, although it might be either a cryptographic key or a password.

In many situations, the organization of the users is a hierarchy. In the hierarchy, each user has his/her subordinates, superiors and/or coordinates. The access control policy in such an organization usually grants a user the access privileges of all his/her subordinates. We call such a policy hierarchical access control policy, and call the scheme that implements such a policy a hierarchical access control scheme. In the scheme, the group of users with the same access privileges is called a security class.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A hierarchical system can be represented as a partially ordered set (poset). In such a hierarchy, all users are allocated into a number of disjoint sets of security classes (or classes in short,) $C_1, C_2, \cdots, C_n$. A binary relation $\leq$ partially orders the set $\mathbb{C} = \{C_1, C_2, \cdots, C_n,\}$. The users in C_j have access to the information held by users in C_i if and only if the relation $C_i \leq C_j$ held in the poset $(\mathbb{C}, \leq)$. We denote $C_i < C_j$ if $C_i \leq C_j$ and C_i is not C_j. If $C_i < C_j$, C_i is called a successor of C_j, and C_j is called a predecessor of C_i. If $C_i < C_j$ and there is no C_k such that $C_i < C_k < C_j$, then C_i is called an immediate successor of C_j, and C_j is called an immediate predecessor of C_i. A class without any predecessor is called a root class. A class without any successor is called a leaf class. A class with both predecessors and successors is called an internal class. An example of the structure of a poset hierarchy is shown in Figure 1.1.

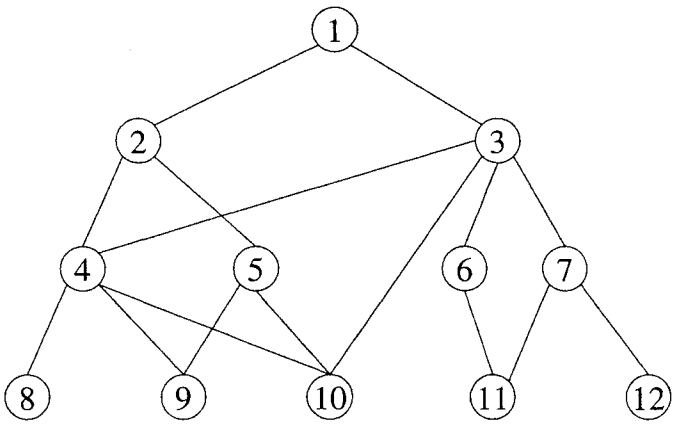

Figure 1.1: Example of the structure of a poset hierarchy

A straightforward access control scheme for poset hierarchy is to assign

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

each class with a key, and let a class have the keys of all its successors. The information belonging to a class is encrypted with the key assigned to that class, therefore the predecessors have access to the information of their successors. This is simple but awkward because the classes in higher hierarchy have to store a large number of keys. In the past two decades, many schemes based on cryptography have been proposed to ease the key management for poset hierarchy. Generally, these schemes are aimed to fully or partly achieve the following goals:

- *Support any arbitrary poset.* It is desirable that any arbitrary poset is supported. Some schemes only support special cases of poset such as a tree. Such schemes are considered restrictive in application.

- *Be secure under attacks.* The schemes are supposed to withstand attacks. For example, a user may try to derive the key of a class that is not his/her successor. The schemes should be secure under all possible attacks.

- *Require small storage space.* Any scheme needs a user in a class to store a certain amount of secret or public parameters. All the schemes tried to reduce the amount of parameters stored.

- *Support dynamic poset structures.* The structure of a hierarchy may change. Classes may be added to or deleted from the hierarchy. In these cases the users in the classes (not only the ones added and deleted) need to update the parameters they store. It is desirable that when a change takes place, the number of classes involved in updating their parameters is as small as possible.

Regarding the security of the schemes, it is important to define how to evaluate whether the schemes are secure. In many of the previous schemes, a list of attacking scenarios are given. However, we can easily give more

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

attacking scenarios. Other than elaborating a list of attacks which may still be incomplete, we prefer to a security definition that can cover any attacks that are possible to the scheme. We come up with the following security model:

Definition 1.1 *A hierarchical access control scheme for poset hierarchy is secure if for any group of classes in the poset, it is infeasible to derive the key of any class that is not a member of that group, nor a successor of any member of that group.*

This model covers any attacks presented in previous schemes. Within this model, only the legitimate predecessors of a class have access to this class. All other users, no matter how they conspire, are not able to access this class.

1.2 Contribution of This Thesis

In the past years a lot of hierarchical access control schemes have been proposed. They made a great progress in improving the performance and security. However, as we will review in details in the next chapter, although some schemes achieve good performance in supporting arbitrary poset, small storage and dynamic structures, none of them have thoroughly proved to be secure under the security model in Definition 1.1.

In this thesis, we propose a new scheme that is superior to the previous schemes in that it provides both good performance and provable security. Our scheme supports arbitrary poset, has similar performance in storage and dynamics achieved by other schemes. The most significant part of our scheme is its formal security proof under Definition 1.1, which the previous schemes did not provide.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1.3 Outline of this thesis

The rests of the thesis are organized as follows. In Chapter 2, we introduce the cryptographic background, including the definitions, concepts and notations, which is necessary for us to present and analyze the previous schemes as well as ours. In Chapter 3 we review the previous schemes, showing the progress, direction and open problems in this topic. In Chapter 4 we present our scheme and its security proof. Chapter 5 summarizes the schemes.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 2

Cryptography Background

In this chapter we introduce the cryptographic background based on which schemes are designed and analyzed. It includes the definitions, notes and algorithms in mathematics (number theory, abstract algebra and finite field); basic cryptographic primitives, cryptography scheme design methodologies. We only include what are necessary for the following chapters. For extensive contents of the background, please refer to [2], [8] and [15].

2.1 Mathematics Background

Integers mod N

Let $\mathbb{Z}$ denote the set of integers, $\mathbb{Z}_+$ denote the set of positive integers.

If a, b are integers, not both zero, then their greatest common divisor, denoted $\gcd(a, b)$, is the largest integer d such that d divides a (denoted as $d|a$) and d divides b ($d|b$). If $\gcd(a, b) = 1$ then we say that a and b are relatively prime. If a, N are integers with $N > 0$ then there are unique integers r, q such that $a = Nq + r$ and $0 \leq r < N$. We call r the remainder upon division of a by N, and denote it by $a \mod N$. If a, b are any integers

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and N is a positive integer, we write

$$a \equiv b \mod N$$

if $a \mod N = b \mod N$. We associate to any positive integer N the following two sets:

$$\mathbb{Z}_N = \{0, 1, \cdots, N-1\}$$

$$\mathbb{Z}_N^* = \{i \in \mathbb{Z} : 1 \leq i \leq N-1 \text{ and } \gcd(i, N) = 1\}$$

The first set is called the set of integers mod N. Its size is N, and it contains exactly the integers that are possible values of $a \mod N$ as a ranges over $\mathbb{Z}$. We define the Euler Phi (or totient) function $\phi(N) = |\mathbb{Z}_N^*|$ for all $N \in \mathbb{Z}_+$. That is, $\phi(N)$ is the size of $\mathbb{Z}_N^*$.

Group

Let $\mathbb{G}$ be a non-empty set and let $\cdot$ denote a binary operation on $\mathbb{G}$. We say that $\mathbb{G}$ is a group if it has the following properties:

1. Closure: For every $a, b \in \mathbb{G}$ it is the case that $a \cdot b$ is also in $\mathbb{G}$.
2. Associativity: For every $a, b, c \in \mathbb{G}$ it is the case that $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.
3. Identity: There exists an element $1 \in \mathbb{G}$ such that $a \cdot 1 = 1 \cdot a = a$ for all $a \in \mathbb{G}$.
4. Invertibility: For every $a \in \mathbb{G}$ there exists a unique $b \in \mathbb{G}$ such that $a \cdot b = b \cdot a = 1$.

The element b in the invertibility condition is referred to as the inverse of the element a, and is denoted a^{-1}.

A group $\mathbb{G}$ is abelian (or commutative) if $a \cdot b = b \cdot a$ for all $a, b \in \mathbb{G}$.

Let N be a positive integer. The operation of addition modulo N takes input any two integers a, b and returns $(a + b) \mod N$. The operation of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

multiplication modulo N takes input any two integers a, b and returns ab mod N. Then $\mathbb{Z}_N$ is a group under addition modulo N, and $\mathbb{Z}_N^*$ is a group under multiplication modulo N.

In $\mathbb{Z}_N$, the identity element is 0 and the inverse of a is $-a \mod N = N - a$. In $\mathbb{Z}_N^*$, the identity element is 1 and the inverse of a is a $b \in \mathbb{Z}_N^*$ such that $ab \equiv 1(\mod N)$.

In any group, we can define an exponentiation operation which associates to any $a \in \mathbb{G}$ and any integer i a group element we denote a^i, defined as follows. If $i = 0$ then a^i is defined to be 1, the identity element of the group. If $i > 0$ then

$$a^i = \underbrace{a \cdot a \cdots a}_{i}.$$

If i is negative, then we define $a^i = (a^{-1})^{-i}$.

With these definition in place, we can manipulate exponents in the way in which we are accustomed with ordinary numbers. Namely, identities such as the following hold for all $a \in \mathbb{G}$ and all $i, j \in \mathbb{Z}$:

$$a^{i+j} = a^i \cdot a^j$$

$$(a^i)^j = a^{ij}$$

$$a^{-i} = (a^i)^{-1}$$

$$a^{-i} = (a^{-1})^i$$

The size of a group $\mathbb{G}$ is called its order, denoted $|\mathbb{G}|$. It is the number of elements in the group. We will often make use of the following basic fact. It says that if any group element is raised to the power the order of the group, the result is the identity element of the group.

Let $\mathbb{G}$ be a group and let $m = |\mathbb{G}|$ be its order. Then $a^m = 1$ for all $a \in \mathbb{G}$.

This means that computation in the group indices can be done modulo m:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Let $\mathbb{G}$ be a group and let $m = |\mathbb{G}|$ be its order. Then $a^i = a^{i \bmod m}$ for all $a \in \mathbb{G}$ and all $i \in \mathbb{Z}$.

Example 2.1 *Let us work in the group $\mathbb{Z}_{21}^*$ under the operation of multiplication modulo 21. The members of this group are 1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20, so the order of the group is $m = 12$. Suppose we want to compute 5^{86} in this group. Applying the above we have*

$$5^{86} \mod 21 = 5^{86 \bmod 12} \mod 21 = 5^2 \mod 21 = 4.$$

□

If $\mathbb{G}$ is a group, a set $\mathbb{S} \subseteq \mathbb{G}$ is called a subgroup if it is a group in its own right, under the same operation as that under which $\mathbb{G}$ is a group. If we already know that $\mathbb{G}$ is a group, there is a simple way to test whether $\mathbb{S}$ is a subgroup: it is one if and only if $x \cdot y^{-1} \in \mathbb{S}$ for all $x, y \in \mathbb{S}$. Here y^{-1} is the inverse of y in $\mathbb{G}$.

Let $\mathbb{G}$ be a group and let $\mathbb{S}$ be a subgroup of $\mathbb{G}$. Then the order of S divides the order of G.

Cyclic groups and generators

Let $\mathbb{G}$ be a group, let 1 denote its identity element, and let $m = |\mathbb{G}|$ be the order of $\mathbb{G}$. If $g \in \mathbb{G}$ is any member of the group, the order of g is defined to be the least positive integer n such that $g^n = 1$. We let

$$< g >= \{g^i : i \in \mathbb{Z}_n\} = \{g^0, g^1, \cdots, g^{n-1}\}$$

denote the set of group elements generated by g. A fact is that this set is a subgroup of $\mathbb{G}$. The order of this subgroup is the order of g, thus the order n of g divides the order m of the group. An element g of the group is called a generator of G if $< g >= \mathbb{G}$, or, equivalently, if its order is m. If g is a

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

generator of $\mathbb{G}$ then for every $a \in \mathbb{G}$ there is a unique integer $i \in \mathbb{Z}_m$ such that $g^i = a$. This i is called the discrete logarithm of a to base g, and we denote it by $DLog_{\mathbb{G},g}(a)$. Therefore, $DLog_{\mathbb{G},g}(a)$ is a function that maps $\mathbb{G}$ to $\mathbb{Z}_m$, and moreover this function is a bijection. The function of $\mathbb{Z}_m$ to $\mathbb{G}$ defined by $i \rightarrow g^i$ is called the discrete exponentiation function, and the discrete logarithm function is the inverse of the discrete exponentiation function.

Here we give an example. Let $p = 11$, which is a prime. Then $\mathbb{Z}_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ has order $p - 1 = 10$. Let us find the subgroups generated by group elements 2 and 5. We raise them to the powers $i = 0, \cdots, 9$. We get:

i	0	1	2	3	4	5	6	7	8	9
$2^i \mod 11$	1	2	4	8	5	10	9	7	3	6
$5^i \mod 11$	1	5	3	4	9	1	5	3	4	9

Looking at which elements appear in the row corresponding to 2 and 5, respectively, we can determine the subgroups these group elements generate:

$$\langle 2 \rangle = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

$$\langle 5 \rangle = \{1, 3, 4, 5, 9\}$$

Since $\langle 2 \rangle$ equals to $\mathbb{Z}_{11}^*$, the element 2 is a generator. Since a generator exists, $\mathbb{Z}_{11}^*$ is cyclic. On the other hand, $\langle 5 \rangle \neq \mathbb{Z}_{11}^*$, so 5 is not a generator of $\mathbb{Z}_{11}^*$. The order of 2 is 10, while the order of 5 is 5. Note that these orders divide 10, the order of the group. The table also enables us to determine the discrete logarithms to base 2 of the different group elements:

a	1	2	3	4	5	6	7	8	9	10
$Dlog_{\mathbb{Z}_{11}^*,2}(a)$	0	1	8	2	4	9	7	3	6	5

The discrete exponentiation function is conjectured to be one-way (meaning the discrete logarithm function is hard to compute) for some cyclic groups

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$\mathbb{G}$. Due to this fact we often seek cyclic groups for cryptographic usage. Here are two example sources of such groups:

- Let p be a prime. Then the group $\mathbb{Z}_p^*$ is cyclic.

  The operation here is multiplication modulo p, and the size of this group is $\phi(p) = p - 1$. This is the most common choice of group in cryptography.

- Let $\mathbb{G}$ be a group and let $m = |\mathbb{G}|$ be its order. If m is a prime number, then $\mathbb{G}$ is cyclic. In other words, any group having a prime number of elements is cyclic.

Another source of cyclic group is from finite field, which is defined later.

Groups of prime order

A group of prime order is a group $\mathbb{G}$ whose order $m = |\mathbb{G}|$ is a prime number. Such a group is always cyclic. These groups turn out to be quite useful in cryptography, so let us take a brief look at them and some of their properties.

An element h of a group $\mathbb{G}$ is called non-trivial if it is not equal to the identity element of the group.

Suppose $\mathbb{G}$ is a group of order q where q is a prime, and h is any non-trivial member of $\mathbb{G}$. Then h is a generator of $\mathbb{G}$.

A common way to obtain a group of prime order for cryptographic schemes is as a subgroup of a group of integers modulo a prime. We pick a prime p having the property that $q = (p - 1)/2$ is also prime. It turns out that the subgroup of quadratic residues modulo p then has order q, and hence is a group of prime order.

Let us now explain what we perceive to be the advantage conferred by working in a group of prime order. Let $\mathbb{G}$ be a cyclic group, and g a generator. We know that the discrete logarithms to base g range in the set $\mathbb{Z}_m$ where

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$m = |\mathbb{G}|$ is the order of $\mathbb{G}$. This means that arithmetic in these exponents is modulo m. If $\mathbb{G}$ has prime order, then m *is* prime. This means that any non-zero exponent has a multiplicative inverse modulo m. In other words, in working in the exponents, we can divide. It is this that turns out to be useful.

Ring

A ring $(\mathbb{R}, +, \cdot)$ consists of a set $\mathbb{R}$ with two binary operations arbitrarily denoted + (addition) and $\cdot$ (multiplication) on $\mathbb{R}$, satisfying the following axioms.

- $(\mathbb{R}, +)$ is an abelian group with an identity denoted 0.
- The operation $\cdot$ is associative. That is, $a \cdot (b \cdot c) = (a \cdot b) \cdot c)$ for all $a, b, c \in \mathbb{R}$
- There is a multiplicative identity denoted 1, with $1 \neq 0$, such that $1 \cdot a = a \cdot 1 = a$ for all $a \in \mathbb{R}$.
- The operation $\cdot$ is distributive over +. That is, $a \cdot (b+c) = (a \cdot b) + (a \cdot c)$ and $(b + c) \cdot a = (b \cdot a) + (c \cdot a)$ for all $a, b, c \in \mathbb{R}$.

The ring is a commutative ring if $a \cdot b = b \cdot a$ for all $a, b \in \mathbb{R}$.

For example, the set $\mathbb{Z}_n$ with addition and multiplication performed modulo n is a commutative ring.

Field

A field is a commutative ring in which all non-zero elements have multiplicative inverses.

For example $\mathbb{Z}_n$ is a field (under the usual operations of addition and multiplication modulo n) if and only if n is a prime number.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Polynomial rings

If $\mathbb{R}$ is a commutative ring, then a polynomial in the indeterminate x over the ring $\mathbb{R}$ is an expression of the form

$$f(x) = a_n x^n + \cdots + a_2 x^2 + a_1 x + a_0$$

where each $a_i \in \mathbb{R}$ and $n \geq 0$. The element a_i is called the coefficient of x_i in $f(x)$. The largest integer m for which $a_m \neq 0$ is called the degree of $f(x)$, denoted $deg\ f(x)$.

Division algorithm for polynomials is defined as follows if $g(x), h(x) \in \mathbb{F}[x]$, with $h(x) \neq 0$, then ordinary polynomial long division of $g(x)$ by $h(x)$ yields polynomials $q(x)$ and $r(x) \in \mathbb{F}[x]$ such that $g(x) = q(x)h(x) + r(x)$, where $deg\ r(x) < deg\ h(x)$. Moreover, $q(x)$ and $r(x)$ are unique. The polynomial $q(x)$ is called the quotient, while $r(x)$ is called the remainder. The remainder of the division is sometimes denoted $g(x) \mod h(x)$, and the quotient is sometimes denoted $g(x)$ div $h(x)$.

If $g(x), h(x) \in \mathbb{F}[x]$ then $h(x)$ divides $g(x)$, written $h(x)|g(x)$, if $g(x) \mod h(x) = 0$.

If $g(x), h(x) \in \mathbb{F}[x]$, then $g(x)$ is said to be congruent to $h(x)$ modulo $f(x)$ if $f(x)$ divides $g(x) - h(x)$. This is denoted by

$$g(x) \equiv h(x) \mod f(x)$$

Let $f(x)$ be a fixed polynomial in $\mathbb{F}[x]$. The equivalence class of a polynomial $g(x) \in \mathbb{F}[x]$ is the set of all polynomials in $\mathbb{F}[x]$ congruent to $g(x)$ modulo $f(x)$.

$\mathbb{F}[x]/(f(x))$ denotes the set of (equivalence classes of) polynomials in $\mathbb{F}[x]$ of degree less than $n = deg\ f(x)$.

If $\mathbb{R}$ is a commutative ring, the polynomial ring $\mathbb{R}[x]$ is the ring formed by the set of all polynomials in the indeterminate x having coefficients from $\mathbb{R}$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The two operations are the standard polynomial addition and multiplication, with coefficient arithmetic performed in the ring $\mathbb{R}$.

Let $f(x) \in \mathbb{F}[x]$ be a polynomial of degree at least 1. Then $f(x)$ is said to be irreducible over $\mathbb{F}$ if it cannot be written as the product of two polynomials in $\mathbb{F}[x]$, each of positive degree.

If $f(x)$ is irreducible over $\mathbb{F}$, then $\mathbb{F}[x]/(f(x))$ is a field.

Finite Field

A finite field is a field $\mathbb{F}$ which contains a finite number of elements. The order of $\mathbb{F}$ is the number of elements in $\mathbb{F}$.

Facts about finite field:

- If $\mathbb{F}$ is a finite field, then $\mathbb{F}$ contains p^m elements for some prime p and integer $m \geq 1$.
- For every prime power order p^m, there is a unique finite field of order p^m. This field is denoted by $\mathbb{F}_{p^m}$, or sometimes by $\mathbb{GF}(p^m)$.

An irreducible polynomial $f(x) \in \mathbb{Z}_p[x]$ of degree m is called a primitive polynomial if x is a generator of $\mathbb{F}^*_{p^m}$, the multiplicative group of all the non-zero elements in $\mathbb{F}_{p^m} = \mathbb{Z}_p[x]/(f(x))$.

Example 2.2 *Example of a finite field $\mathbb{F}_{2^4}$ of order 16: It can be verified that the polynomial $f(x) = x^4 + x + 1$ is irreducible over $\mathbb{Z}_2$. Hence the finite field $\mathbb{F}_{2^4}$ can be represented as the set of all polynomials over $\mathbb{F}_2$ of degree less than 4. That is,*

$$\mathbb{F}_{2^4} = \{a_3x^3 + a_2x^2 + a_1x + a_0 | a_i \in \{0, 1\}\}.$$

For convenience, the polynomial $a_3x^3 + a_2x^2 + a_1x + a_0$ can be represented by the vector $(a_3a_2a_1a_0)$ of length 4, and

$$\mathbb{F}_{2^4} = \{(a_3a_2a_1a_0) | a_i \in \{0, 1\}\}.$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

□

The following are some examples of field arithmetic.

- Field elements are simply added componentwise: for example,

$$(1011) + (1001) = (0010).$$

- To multiply the field elements (1101) and (1001), multiply them as polynomials and then take the remainder when this product is divided by f(x):

$$\begin{aligned}(x^3 + x^2 + 1) \cdot (x^3 + 1) &= x^6 + x^5 + x^2 + 1 \\ &\equiv x^3 + x^2 + x + 1 \mod f(x).\end{aligned}$$

  Hence $(1101) \cdot (1001) = (1111)$

- The multiplicative identity of $\mathbb{F}_{2^4}$ is (0001).

- The inverse of (1011) is (0101). To verify this, observe that

$$\begin{aligned}(x^3 + x + 1) \cdot (x^2 + 1) &= x^5 + x^2 + x + 1 \\ &\equiv 1 \mod f(x),\end{aligned}$$

  whence $(1011) \cdot (0101) = (0001)$.

$f(x)$ is a primitive polynomial, or, equivalently, the field element $x = (0010)$ is a generator of $\mathbb{F}_{2^4}$. This may be checked by verifying that all the non-zero elements in $\mathbb{F}_{2^4}$ can be obtained as a powers of x. The computations are summarized in Table 2.1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 2.1: The powers of x modulo $f(x) = x^4 + x + 1$.

i	$x^i \mod x^4 + x + 1$	vector notation
0	1	(0001)
1	x	(0010)
2	x^2	(0100)
3	x^3	(1000)
4	$x + 1$	(0011)
5	$x^2 + x$	(0110)
6	$x^4 + x^2$	(1100)
7	$x^3 + x + 1$	(1011)
8	$x^2 + 1$	(0101)
9	$x^3 + x$	(1010)
10	$x^2 + x + 1$	(0111)
11	$x^3 + x^2 + x$	(1110)
12	$x^3 + x^2 + x + 1$	(1111)
13	$x^3 + x^2 + 1$	(1101)
14	$x^3 + 1$	(1001)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 2.2: Bit complexity of basic operations in $\mathbb{Z}_n$.

Operation		Bit Complexity
Modular Addition	$(a+b) \mod n$	$O(\log n)$
Modular subtraction	$(a-b) \mod n$	$O(\log n)$
Modular multiplication	$(a \cdot b) \mod n$	$O(\log^2 n)$
Modular inversion	$a^{-1} \mod n$	$O(\log^2 n)$
Modular exponentiation	$a^k \mod n, k < n$	$O(\log^3 n)$

2.2 Complexity of Algorithms

The numbers arising in cryptographic algorithms are large, having magnitudes like 2^{512} or 2^{1024}. The arithmetic operations on these numbers are the main cost of the algorithm, and the costs grow as the numbers get bigger.

The numbers are provided to the algorithm in binary, and the size of the input number is thus the number of bits in its binary representation. We call this the length, or binary length, of the number, and we measure the running time of the algorithm as a function of the binary lengths of its input numbers. In computing the running time, we count the number of bit operations performed.

Table 2.2 summarizes the bit complexity of basic operations in $\mathbb{Z}_n$.

Table 2.3 summarizes the complexity of basic operations in $\mathbb{F}_{p^m}$. In the table, "operations in $\mathbb{Z}_p$" means either an addition, subtraction, multiplication, inversion, or division in $\mathbb{Z}_p$.

All these operations can be finished within polynomial (in the number of the bits of the inputs) steps.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 2.3: Complexity of basic operations in $\mathbb{F}_{p^m}$.

Operation	Number of operations in $\mathbb{Z}_p$
Addition	O(m)
Substraction	O(m)
Multiplication	$O(m^2)$
Inversion	$O(m^2)$
Exponentiation	$O((\lg(p))m^3)$

2.3 Intractable Computational Problems

In modern cryptography, the security of the cryptographic schemes relies on the intractability of the computational problems. These problems are believed to be intractable, although no proof is known. We present some of them that are used in the schemes we will review and present below. We take the notations mainly from [2]. For an introduction to the problems in more plain English, please refer to [15].

Note all the operations in the following problems are modular operations on corresponding groups. For simplicity we omit the modular expression. For example, we write $a + b$ instead of $a + b \mod N$ if we have indicated that the operation is on the group $\mathbb{Z}_N$.

2.3.1 Discrete Logarithm Problem

As we have seen, on the cyclic group the discrete exponentiation function can be computed by a polynomial algorithm. Its inversion, the Discrete Logarithm Problem (DLP) is defined as the following: given a finite cyclic group $\mathbb{G}$ of order n, a generator g of $\mathbb{G}$, and an element $\beta \in \mathbb{G}$, find the integer $x \in [1, n]$, such that $g^x = \beta$. The DLP is believed to be hard. Next

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

we give a quantitative description about the hardness.

Let $\mathcal{A}$ be an probabilistic polynomial algorithm that inverts the exponentiation function. Let $x \xleftarrow{\$} \mathbb{X}$ denotes the operation of selecting an element x uniformly from some set $\mathbb{X}$ at random. We consider the experiment in Experiment 1.

$x \xleftarrow{\$} \mathbb{Z}_n$
$X \leftarrow g^x$
$x' \leftarrow \mathcal{A}(X)$
if $g^{x'} = X$ **then**
  return 1
else
  return 0
end if

Experiment 1: Experiment $Exp^{dl}_{\mathbb{G},g}(\mathcal{A})$

In this experiment, we anticipated there is some probability that the return value is 1, i.e., $\mathcal{A}$ has some probability that output an x' such that $g^{x'} = g^x$. The *dl*-advantage of $\mathcal{A}$ is defined as

$$Adv^{dl}_{\mathbb{G},g}(\mathcal{A}) = Pr[Exp^{dl}_{\mathbb{G},g}(\mathcal{A}) = 1]$$

The definition above measures how good an algorithm is at solving the discrete logarithm problem.

The discrete logarithm problem is believed to be intractable. Formally speaking, let l be the bit-length of the order of the group $\mathbb{G}$, for any polynomial time (in l) algorithm $\mathcal{A}$ and any polynomial $P(\cdot)$, for sufficiently large l,

$$Adv^{dl}_{\mathbb{G},g}(\mathcal{A}) = Pr[Exp^{dl}_{\mathbb{G},g}(\mathcal{A}) = 1] < \frac{1}{P(l)}.$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.3.2 The RSA problem

The RSA problem is defined as follows: given a positive integer n that is a product of two distinct odd primes p and q, a positive integer e such that $\gcd(e, \phi(n)) = \gcd(e, (p-1)(q-1)) = 1$, and an integer c, find an integer $m \in \mathbb{Z}_n$ such that $m^e = c$.

Let $\mathcal{A}$ be a probabilistic polynomial time algorithm that solves the RSA problem. Let $\mathbb{G} = \mathbb{Z}_n$. We consider the experiment in Experiment 2.

$x \xleftarrow{\$} \mathbb{G}; X \leftarrow x^e$
$x' \leftarrow \mathcal{A}(X)$
if $(x')^e = X$ **then**
  return 1
else
  return 0
end if

Experiment 2: Experiment $Exp^{RSA}_{\mathbb{G},e}(\mathcal{A})$

The RSA-advantage of $\mathcal{A}$ is defined as

$$Adv^{RSA}_{\mathbb{G},e}(\mathcal{A}) = Pr[Exp^{RSA}_{\mathbb{G},e}(\mathcal{A}) = 1]$$

Like the discrete logarithm problem, the RSA problem is believed to be intractable: let l be the bit-length of the order of the group $\mathbb{G}$, for any $\mathcal{A}$ and any polynomial $P(\cdot)$, for sufficiently large l,

$$Adv^{RSA}_{\mathbb{G},e}(\mathcal{A}) = Pr[Exp^{dl}_{\mathbb{G},e}(\mathcal{A}) = 1] < \frac{1}{P(l)}.$$

2.3.3 Decisional Diffie-Hellman Problem

The DDH problem is to distinguish the two distributions (g, g^x, g^y, g^{xy}) and (g, g^x, g^y, g^z), where g is the generator of a finite cyclic group $\mathbb{G}$ of order m; x,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

y, z are random variables uniformly distributed on $[1, m]$. In another word, given (g, g^x, g^y), DDH problem is to distinguish g^{xy} from a random variable uniformly distributed on $\mathbb{G}$.

The formalization considers a "two worlds" setting. The adversary gets input X, Y, Z. In either world, X, Y are random group elements, but the manner in which Z is chosen depends on the world. In World 1, $Z = g^{xy}$ where $x = DLog_{\mathbb{G},g}(X)$ and $y = DLog_{\mathbb{G},g}(Y)$. In World 0, Z is chosen at random from the group, independently of X, Y . The adversary must decide in which world it is.

Let $\mathbb{G}$ be a cyclic group of order m, let $\mathcal{A}$ be a distinguisher, an probabilistic polynomial algorithm that returns one bit, 0 or 1, depending on which world $\mathcal{A}$ thinks it is in. We consider the experiments in Experiment 3:

Experiment $Exp^{ddh-1}_{\mathbb{G},g}(\mathcal{A})$	Experiment $Exp^{ddh-0}_{\mathbb{G},g}(\mathcal{A})$
$x \xleftarrow{\$} \mathbb{Z}_m$	$x \xleftarrow{\$} \mathbb{Z}_m$
$y \xleftarrow{\$} \mathbb{Z}_m$	$y \xleftarrow{\$} \mathbb{Z}_m$
$z \leftarrow xy$	$z \xleftarrow{\$} \mathbb{Z}_m$
$X \leftarrow g^x; Y \leftarrow g^y; Z \leftarrow g^z$	$X \leftarrow g^x; Y \leftarrow g^y; Z \leftarrow g^z$
return $\mathcal{A}(X, Y, Z)$	return $\mathcal{A}(X, Y, Z)$

Experiment 3: DDH experiments

The ddh-advantage of $\mathcal{A}$ is defined as

$$Adv^{ddh}_{\mathbb{G},g}(\mathcal{A}) = \left| P[Exp^{ddh-0}_{\mathbb{G},g}(\mathcal{A})] - P[Exp^{ddh-1}_{\mathbb{G},g}(\mathcal{A})] \right|$$

The definition above measures how good $\mathcal{A}$ can distinguish the two world.

The DDH problem is believed to be intractable on some cyclic group, i.e., let l be the bit-length of the order of such a group $\mathbb{G}$, for any $\mathcal{A}$ and any polynomial $P(\cdot)$, for sufficiently large l,

$$Adv^{ddh}_{\mathbb{G},g}(\mathcal{A}) < \frac{1}{P(l)}.$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Note DDH is not intractable in every cyclic group. Most groups in which DDH is believed to be intractable have prime order. In [3] a list of such groups are given. Here we just present one that will be used in our scheme:

Let $p = 2q + 1$ where both p and q are prime. Let $\mathbb{Q}_p$ be the subgroup of order q of $\mathbb{Z}_p^*$. $\mathbb{Q}_p$ is a cyclic group of prime order on which DDH is intractable.

2.4 Cryptographic Primitives

2.4.1 One-way function

A one-way function is a function which is easy to compute but hard to invert. Here we give our formal definition.

Definition 2.3 *Let m, n be polynomials. Let l be an integer parameter, $\mathcal{D} = \bigcup_l \{0,1\}^{m(l)}$ and $\mathcal{R} = \bigcup_l \{0,1\}^{n(l)}$, the function $f : \mathcal{D} \to \mathcal{R}$ is a one-way function if the two conditions hold:*

1. *easy to compute. On input $x \in \{0,1\}^{m(l)}$, $f(x)$ can be computed in polynomial time (in l), and*

2. *hard to invert. For any probabilistic polynomial-time (in l) algorithm $\mathcal{A}$, any polynomial $P(\cdot)$, and all sufficiently large l, on input $x \in \{0,1\}^{m(l)}$*

$$P_r[\mathcal{A}(f_l(x)) = x] < \frac{1}{P(l)},$$

where f_l denotes the restriction of f on $\{0,1\}^{m(l)}$.

For example, on the cyclic group $\mathbb{G}$, the exponentiation function is one-way because it is easy to compute, but hard to invert. Its inversion DLP is intractable.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.4.2 One-way hash function

One-way hash function $h : \mathcal{D} \rightarrow \mathcal{R}, \mathcal{D} = \{0,1\}^{m(l)}$ ($\mathcal{R} = \{0,1\}^{n(l)}, m(l) > n(l), m(\cdot), n(\cdot)$ are polynomials) is an one-way function with pre-image resistance and 2nd pre-image resistance properties:

1. pre-image resistance: given $x \in \mathcal{D}$, it is infeasible to find x' such that $x' \neq x$ and $h(x') = h(x)$.

2. 2nd pre-image resistance: it is infeasible to find a pair of $x \neq x'$ such that $h(x') = h(x)$.

2.4.3 Universal one-way hash function family

Universal one-way hash function families are first proposed in [13]. Then [17] extended the work. A generalization of the universal one-way hash function family is proposed in [23] (called sibling intractable function family SIFF), which is used to solve the hierarchical access control problem. Here we give the definition and construction of the universal one-way hash function family.

Definition 2.4 *Let m, n be polynomials. Let l be an integer parameter, $\mathcal{D} = \bigcup_l \{0,1\}^{m(l)}$ and $\mathcal{R} = \bigcup_l \{0,1\}^{n(l)}$, the functions $\{f | f : \mathcal{D} \rightarrow \mathcal{R}\}$ is a family of universal one-way hash functions if for all probabilistic polynomial algorithm $\mathcal{A}$ the following holds for sufficiently large l:*

1. *On input $x \in \{0,1\}^{m(l)}$, $P_r[\mathcal{A}(f,x) = x', f(x) = f(x'), x' \neq x] < \frac{1}{P(l)}$ where the probability is taken over all $f \in \{f_l\}$ and the random choices of $\mathcal{A}$.*

2. *f_l is computable in polynomial time (in l).*

3. *f_l is accessible: there exists an algorithm $\mathcal{G}$ such that $\mathcal{G}$ on input l generates uniformly at random a description of $f \in f_l$.*

   (f_l denotes the restriction of f on $\{0,1\}^{m(l)}$.)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Example 2.5 *An example of universal one-way hash function is constructed below.*

1. *On the finite field* $\mathbb{F}_{2^l}$, $\{f_l\} = \{f_{a,b} | f_{a,b}(x) = chop(ax + b), a, b \in \mathbb{F}_{2^l}\}$ *where all computation are in* $\mathbb{F}_{2^l}$ *and chop:* $\{0,1\}^l \mapsto \{0,1\}^{l-1}$ *chops the last bit.*

2. *Let* g *be an one-way permutation. Define* $H_l = \{h = f \circ g | f \in \{f_l\}\}$.

Then $\bigcup_l (H_l)$ *is a universal one-way hash function.* □

The universal one-way hash function has the following Collision Accessibility property: Given $x \neq x' \in \{0,1\}^{m(l)}$, it is easy to find $h \in h_l$ such that $h(x) = h(x')$. Note that by the definition of the universal one-way hash function family, given the x and h, it is intractable to find the $x' \neq x$ such that $h(x') = h(x)$.

2.4.4 Pseudo-Random function

The pseudo-random function family was proposed by Goldreich, Goldwasser and Micali in [9]. In such a family, each function is specified by a short, random key, and can be easily computed given the key. But without the key, given an input, the output of the function looks like a random number. Next we give a formal description.

A function family is a map $\mathcal{F} : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$. Here $\mathcal{K}$ is the set of keys of $\mathcal{F}$ and $\mathcal{D}$ is the domain of $\mathcal{F}$ and $\mathcal{R}$ is the range of $\mathcal{F}$. The set of keys and the range are finite, and all of the sets are nonempty. The two-input function $\mathcal{F}$ takes a key K and an input X to return a point Y we denote by $\mathcal{F}(K, X)$. For any key $K \in \mathcal{K}$ we define the map $F_K : \mathcal{D} \to \mathcal{R}$ by $F_K(X) = \mathcal{F}(K, Y)$. We call the function F_K an instance of function family $\mathcal{F}$. Thus $\mathcal{F}$ specifies a collection of maps, one for each key.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Let $Func(\mathcal{D}, \mathcal{R})$ denote the family of all functions of $\mathcal{D}$ to $\mathcal{R}$. Suppose $\mathcal{D} = \{0,1\}^l$, $\mathcal{R} = \{0,1\}^L$, then the size of the key space of $Func(\mathcal{D}, \mathcal{R})$ is 2^{L2^l}. There is a key for every function of l-bits to L bits, and this is the number of such functions.

A random function $g : \mathcal{D} \rightarrow \mathcal{R}$ is an instance uniformly picked from $Func(\mathcal{D}, \mathcal{R})$ at random, and put in a black-box. This means that one can give any value X, and get back $g(X)$. But one cannot get the description of the instance g. The dynamic view of a random function can be thought of as implemented by the following computer program. The program maintains the function in the form of a table T where $T[X]$ holds the value of the function at X. Initially, the table is empty. The program processes an input $X \in \mathcal{D}$ as follows:

if T[X] is not defined **then**
  $Y \rightarrow R; T[x] \rightarrow Y$
end if
return T[x]

The answer on any point is random and independent of the answers on other points.

A pseudo-random function is a family of functions, which is a subset of the random function family $Func(\mathcal{D}, \mathcal{R})$, with the property that the input-output behavior of a random instance of this family is "computationally indistinguishable" from that of a random function.

We fix a family of functions $\mathcal{F} : \mathcal{K} \times \mathcal{D} \rightarrow \mathcal{R}$, and assume a two-world setting:

World 0: The function g is drawn at random from $Func(\mathcal{D}, \mathcal{R})$.

World 1: The function g is drawn at random from $\mathcal{F}$. Note $\mathcal{F}$ is a subset of $Func(\mathcal{D}, \mathcal{R})$.

Let $\mathcal{A}$ be an algorithm that takes an oracle from a function $g : \mathcal{D} \rightarrow \mathcal{R}$, to return a bit, 0 or 1, to indicate which family of function the adversary

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

thinks the g is from. We consider the experiments in Experiment 4.

Experiment $Exp_{\mathcal{F}}^{prf-1}(\mathcal{A})$	Experiment $Exp_{\mathcal{F}}^{prf-0}(\mathcal{A})$
$g \xleftarrow{\$} \mathcal{F}$	$g \xleftarrow{\$} Func(\mathcal{D}, \mathcal{R})$
$\mathcal{A}$ queries g	$\mathcal{A}$ queries g
$\mathcal{A}$ outputs b	$\mathcal{A}$ outputs b
return b	return b

Experiment 4: Experiments for pseudo-random function distinguisher

The prf-advantage of $\mathcal{A}$ is defined as

$$Adv_{\mathcal{F}}^{prf}(\mathcal{A}) = |P[Exp_{\mathcal{F}}^{prf-0}(\mathcal{A}) = 1] - P[Exp_{\mathcal{F}}^{prf-1}(\mathcal{A}) = 1]|$$

The definition above measures how good $\mathcal{A}$ can distinguish the two world.

If for any polynomial time algorithm $\mathcal{A}$, its prf-advantage is negligible, then the function family $\mathcal{F} : \mathcal{K} \times \mathcal{D} \rightarrow \mathcal{R}$ is a pseudo-random function family.

The block ciphers such as DES and AES, are modeled as pseudo-random functions (or permutations). That is, let l be the block size, for any polynomial time algorithm $\mathcal{A}$, any polynomial P, for sufficiently large l, there is

$$Adv_{\mathcal{F}}^{prf}(\mathcal{A}) < \frac{1}{P(l)}.$$

The DDH problem we introduced above is also believed to be a pseudo-random function family.

2.5 Cryptographic Scheme Security

In this section, we introduce the cryptographic protocol design approaches. After reviewing the Cryptanalysis-driven design, Shannon Security, we will focus on provable security which is used in our design. For more details, please refer to [2], [8].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.5.1 Cryptanalysis-driven design

Traditionally, cryptographic protocols have been designed by focusing on concrete attacks and how to defeat them. The approach works like this:

1. A cryptographic goal is recognized.
2. A solution is offered.
3. One searches for an attack on the proposed solution.
4. When one feasible attack is found, go back to Step 2 and try to come up with a better solution. The process then continues.

Step 3 is called cryptanalysis. In the classical approach to design cryptographic scheme, cryptanalysis was an essential component of constructing any new design.

There are some difficulties with the approach of cryptanalysis-drive design. The obvious problem is that one never knows if things are right. The process should iterate until one feels "confident" that the solution is adequate. But one has to accept that design errors might come to light at any time.

2.5.2 Shannon Security

A "systematic" approach to cryptography, where proofs and definitions play a visible role, begins in the work of Claude Shannon[19], which measures the secrecy of the information with information theory concepts. We briefly present the idea of Shannon as follows. Let $P : \{0,1\}^n \rightarrow [0,1]$ be a probability distribution on the set of n-bit plaintexts. That is, assume Alice chooses a plaintext m to send with probability $P[m]$. This distribution is known to everyone, including the adversary. Thus, before the ciphertext c is transmitted, all the adversary knows is that any particular message m has probability

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$P[m]$ of being transmitted. Shannon security requires that, the conditional probability that after observing the ciphertext c, to the adversary, the probability that the message is m keeps $P[m]$. That is $P[m|c] = P[m]$. It means that the adversary does not get any information about the plaintext from its ciphertext.

An example of such a scheme, one-time pad, is Shannon-secure. The one-time pad is as follows. Alice and Bob share a random secret key of n bits, $K = k_1 k_2 \cdots k_n$. Alice want to send Bob a message M, also n bits, $M = m_1 m_2 \cdots m_n$. The ciphertext C is the bit-wise XOR of the plaintext and key:

$$C = M \bigoplus K = c_1 c_2 \cdots c_n,$$

where

$$c_1 = k_1 \bigoplus m_1, c_2 = k_2 \bigoplus m_2, \cdots, c_n = k_n \bigoplus m_n.$$

After receiving the ciphertext, Bob can recover the plaintext with the key:

$$M = C \bigoplus K = m_1 m_2 \cdots m_n,$$

where

$$m_1 = k_1 \bigoplus c_1, m_2 = k_2 \bigoplus c_2, \cdots, m_n = k_n \bigoplus c_n.$$

In the one-time pad encryption, the adversary does not get any information about M from C.

Shannon-security however has important limitations. To achieve Shannon security, the key has to be as long as the message. If an encryption scheme is to meet Shannon security, the number of key bits must be at least the total number of plaintext bits we're going to encrypt.

This fact has some fundamental implications. If we want to do practical cryptography, we must be able to use a single short key to encrypt lots of bits. This means that we will not be able to achieve Shannon security. A different paradigm and a different notion of security have to be taken.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.5.3 Provable Security

The modern cryptography introduces a new dimension: the amount of computing power available to an adversary. It seeks to have security as long as adversaries do not have "too much" computing time. Schemes are breakable if the adversary has infinite computing power, but in practice, the attacks are infeasible.

For the security of a scheme, we will want to be making statements like this: Assuming the adversary uses no more than t computing cycles, her probability of breaking the scheme is at most $t = 2^{-200}$. Notice we do not assume how the adversary operates, what algorithm, or technique the adversary uses.

The legitimate parties must be able to efficiently execute the scheme instructions. Their effort should be reasonable. But the task for the adversary must be infeasible.

Cryptographic primitives

The computational nature of modern cryptography means that one must find computationally hard problems, and base the cryptography schemes on them. The basic problems are called cryptographic primitives. They have some "hardness" or "security" properties, but by themselves they do not solve any problem of interest. They must be properly used as building blocks to achieve some useful scheme.

Cryptographic primitives are drawn from two sources: engineered constructs and mathematical problems. In the first class fall standard block ciphers such as the well-known AES algorithm. In the second class falls the DLP, RSA and DDH problems we have introduced above.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The provable-security approach

From the cryptographic primitives, we start to transform them into schemes to solve the practical problems. We will view a cryptographer as an engine for turning the primitives into schemes. That is, we focus on scheme design under the assumption that good primitives exist.

A poorly designed scheme can be insecure even though the underlying primitive is good. The fault is not of the underlying primitive, but that primitive was somehow misused.

In practice, lots of application schemes have been broken, yet the good primitives, like AES and RSA, have never been convincingly broken. We would like to build on the strength of such primitives in such a way that schemes can "inherit" this strength, not lose it. The provable-security paradigm lets us do that.

The provable-security paradigm is as follows. Take some goal, like achieving privacy via symmetric encryption. The first step is to make a formal adversarial model and define what it means for an encryption scheme to be secure. The definition explains exactly when the adversary is successful.

With a definition in hand, a particular scheme, based on some particular primitive, can be put forward. It is then analyzed from the point of view of meeting the definition. The plan is now show security via a reduction. A reduction shows that the only way to defeat the scheme is to break the underlying primitive.

A reduction is a proof that if the cryptographic primitive does the job it is supposed to do, then the scheme we have made does the job that it is supposed to do. Believing this, it is no longer necessary to directly cryptanalyze the scheme. If one found a weakness in the scheme, one would have found the weakness in the underlying primitive. And if we believe the primitive is secure, then without further cryptanalysis of the scheme, we believe the scheme is secure too.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In order to do a reduction one must have a formal notion of what is meant by the security of the underlying cryptographic primitive: what attacks, exactly, does it withstand? For example, we might assume that discrete exponentiation function is a one-way function and can not be inverted.

Here is another way of looking at what reductions do. When a reduction from the onewayness of discrete exponentiation function to the security of the scheme, it is actually giving a transformation with the following property. Suppose the adversary $\mathcal{A}$ is able to break the scheme. The transformation takes $\mathcal{A}$ and turns it into another adversary that breaks discrete logarithm problem. Thus we conclude, as long as we believe no adversary cannot break DLP, there could be no such adversary $\mathcal{A}$ that breaks the scheme. In other words, the scheme is secure.

The concept of using reductions in cryptography is a beautiful and powerful idea. Schemes designed in this way have superior security guarantees. Yet we need to notice that in some ways the term "provable security" is misleading. As the above indicates, what is probably the central step is providing a model and definition of security. The reduction proves the scheme is secure under the security model. Whether the scheme is secure in practice depends on whether the model is defined reasonably.

The scheme we provide in chapter 4 takes the provable-security approach.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 3

Related Works

In this chapter we review a number of previous schemes, trying to show a trace of efforts, progresses and directions in hierarchical access control. This is not a comprehensive list of all works, but to the best of our knowledge, these schemes present distinct ideas in solving the problems, and are typical among similar schemes.

All these schemes consist of two procedures. One is the key assignment. In this procedure a Central Authority (CA) assigns the secret keys and related public parameters to classes. The other is the key derivation. It is a procedure that a class derives the keys of its successors.

These schemes are categorized into 2 groups. One group is called *direct access schemes* because in these schemes, a predecessor can compute the key of any successor without knowing the parameters of other successors between them. The other group is called *indirect access schemes* because in order to compute the key of a successor, a predecessor has to compute the keys of the successors between them.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.1 Some Direct Access Schemes

This group of schemes can also be called RSA-based schemes because the intractability of the RSA problem forms the basis of the security of the schemes.

3.1.1 Akl-Taylor scheme

Akl-Taylor scheme [1] is the first scheme that addresses the access control in a poset hierarchy. Let $C_1, C_2, \cdots, C_i, \cdots$ indicate the security classes in the poset. The key assignment are as follows:

choose 2 large secret prime numbers p and q, and publish $N = pq$
choose a secret $s \in \mathbb{Z}_n$
for each class C_i **do**
  assign a distinct prime number p_i
  $P_i = \prod_{C_j \not\leq C_i} p_j$ is assigned to each class as its public parameter
  $k_i = s^{P_i} \mod N$ is assigned to C_i as its secret key
end for

Procedure 5: AKL-Taylor Key assignment

When a class C_i tries to derive the secret key of class $C_j < C_i$, it runs the following key derivation algorithm:

$k_j = k_i^{P_j/P_i}$

Procedure 6: AKL-Taylor Key derivation

The correctness of the key derivation procedure is easy to verify:

$$\begin{aligned} k_i^{P_j/P_i} &= (s^{P_i})^{P_j/P_i} \\ &= s^{P_j} \\ &= k_j \end{aligned}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 3.1: Key Assignment in Akl-Taylor scheme

Security Class	Public Parameter	Secret Key
1	$P_1 = 1$	k_1
2	$P_2 = p_1p_3p_6p_7p_{11}p_{12}$	$k_2 = k_1^{P_2}$
3	$P_3 = p_1p_2$	$k_3 = k_1^{P_3}$
4	$P_4 = p_1p_2p_3p_5p_6p_7p_{11}p_{12}$	$k_4 = k_1^{P_4}$
5	$P_5 = p_1p_2p_3p_4p_6p_7p_8p_{11}p_{12}$	$k_5 = k_1^{P_5}$
6	$P_6 = p_1p_2p_3p_4p_5p_7p_8p_9p_{10}p_{12}$	$k_6 = k_1^{P_6}$
7	$P_7 = p_1p_2p_3p_4p_5p_6p_8p_9p_{10}$	$k_7 = k_1^{P_7}$
8	$P_8 = p_1p_2p_3p_4p_5p_6p_7p_9p_{10}p_{11}p_{12}$	$k_8 = k_1^{P_8}$
9	$P_9 = p_1p_2p_3p_4p_5p_6p_7p_8p_{10}p_{11}p_{12}$	$k_9 = k_1^{P_9}$
10	$P_{10} = p_1p_2p_3p_4p_5p_6p_7p_8p_9p_{11}p_{12}$	$k_{10} = k_1^{P_{10}}$
11	$P_{11} = p_1p_2p_3p_4p_5p_6p_7p_8p_9p_{10}p_{12}$	$k_{11} = k_1^{P_{11}}$
12	$P_{12} = p_1p_2p_3p_4p_5p_6p_7p_8p_9p_{10}p_{11}$	$k_{12} = k_1^{P_{12}}$

The secret analysis is as follows:

If $C_i \geq C_j$, then $P_j | P_i$, and $k_j = k_i^{P_j/P_i}$ is computable by C_i with secret key and the public parameters P_i and P_j.

If $C_j \not\geq C_i$, then $P_j \nmid P_i$, and to compute $k_j = k_i^{P_j/P_i}$ is to solve the RSA problem, which is not feasible.

We give an example for the AKL-Taylor scheme. A hierarchy shown in Figure 1.1 consists of $C_1, \cdots, C_{12}$. Suppose $p_1, \cdots, p_{12}$ are the prime numbers assigned to the classes respectively. Then secret keys and the public parameters of the classes are shown in Table 3.1.

In the case $C_2 \geq C_{10}$, if C_2 is to derive the key of C_{10} , it can compute

$$\begin{aligned} k_{10} &= k_2^{P_{10}/P_2} \mod N \\ &= k_2^{p_2p_4p_5p_8p_9} \mod N \end{aligned}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In the case $C_4 \not\geq C_5$, if C_4 tries to derive k_5, it will have to solve the RSA problem, which is infeasible:

$$\begin{aligned} k_5 &= k_4^{P_5/P_4} \mod N \\ &= k_4^{p_4 p_8/p_5} \mod N \end{aligned}$$

Akl-Taylor scheme is an elegant solution to the access control in a poset hierarchy. But with this scheme, a large amount of storage for the public parameters is required. For example, in a system with n classes, a leaf class (a class without any successor) needs to store the product of n distinct prime numbers. Moreover, once a security class is added to or deleted from the system, the public parameters and keys of all the classes except for its predecessors have to be re-calculated.

Later, in [14], Mackinnon et al. presented an algorithm for prime assignment for Akl-Taylor scheme. With the improved assignment, the primes assigned to the classes do not have to be distinct as in the original Akl-Taylor scheme. This reduces the number of distinct primes in Akl-Taylor scheme, but the number of primes used in a class's public parameter is still the same as in Akl-Taylor scheme.

3.1.2 Harn-Lin scheme

In [10], Harn and Lin proposed an scheme that can be viewed as a "mirror version" of the Akl-Taylor scheme, which shifts the storage load from the lower classes to the upper classes. The key assignment procedure is shown in Procedure 7.

If a class C_i tries to derive the secret key of class C_j, it runs the key derivation algorithm in Procedure 8.

The correctness Harn-Lin key assignment and derivation can be verified as follows:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CA choose 2 large secret prime numbers p and q, and publish $N = pq$
CA choose a public $a \in [2, n-1]$ such that $gcd(a, N) = 1$
for each class C_i **do**
  assign a distinct prime number e_i
  compute $d_i = e_i^{-1} \mod \phi(N)$
end for
for each class C_i **do**
  CA computer $P_i = \prod_{C_j \leq C_i} e_j$
  CA assign P_i to each class as its public parameter
  CA compute $k_i = a^{\prod_{C_j \leq C_i} d_i \mod \phi(N)} \mod N$
  CA assign k_i to C_i as its secret key
end for

Procedure 7: Harn-Lin Key assignment

$k_j = k_i^{P_j/P_i}$

Procedure 8: Harn-Lin Key derivation

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Notice that $e_i d_i = 1 \mod \phi(N)$

$$\begin{aligned} k_i^{P_i/P_j} &= (a^{\prod_{C_k \le C_i} d_k \mod \phi(N)})^{\prod_{C_k \le C_i} e_k / \prod_{C_k \le C_j} e_k} \mod N \\ &= (a^{\prod_{C_k \le C_j} e_k \mod \phi(N)}) \mod N \\ &= k_j \end{aligned}$$

The security of the Harn-Lin scheme is analyzed as follows:

If $C_i \ge C_j$, $P_j | P_i$, then $k_j = k_i^{P_i/P_j}$ is computable by C_i with secret key and the public parameters P_i and P_j.

If $C_j \not\ge C_i$, then $P_j \nmid P_i$, and to compute $K_j = K_i^{P_j/P_i}$ is to solve the RSA problem, which is not feasible. □

Also, we give an example of Harn-Lin scheme. Like the example for the AKL-Taylor scheme, hierarchy shown in Figure 1.1 consists of $C_1, \cdots, C_{12}$. Suppose $p_1, \cdots, p_{12}$ are the prime numbers assigned to the classes respectively. Then secret keys and the public parameters of the classes are listed in Table 3.2

In the case $C_2 \ge C_{10}$, if C_2 is to derive the key of C_{10} , it can compute

$$\begin{aligned} k_{10} &= k_2^{P_2/P_{10}} \mod N \\ &= k_2^{p_2 p_4 p_5 p_8 p_9} \mod N \end{aligned}$$

In the case $C_4 \not\ge C_5$, if C_4 tries to derive k_5, it will have to solve the RSA problem, which is infeasible:

$$\begin{aligned} k_5 &= k_4^{P_4/P_5} \mod N \\ &= k_4^{e_4 e_8/e_5} \mod N \end{aligned}$$

With the Harn-Lin scheme, the higher a class is in the hierarchy, the larger storage it requires. In a hierarchy with n classes, the leaf classes need to store only 1 prime as its public parameter, but a root class (a class that is the predecessor of all other classes) needs to store the product of n

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 3.2: Key assignment in Harn-Lin scheme

Security Class	Public Parameter	Secret Key
1	$P_1 = e_1e_2e_3e_4e_5e_6e_7e_8e_9e_{10}e_{11}e_{12}$	$k_1 = a^{P_1^{-1} \bmod \phi(N)} \bmod N$
2	$P_2 = e_2e_4e_5e_8e_9e_{10}$	$k_2 = a^{P_2^{-1} \bmod \phi(N)} \bmod N$
3	$P_3 = e_3e_4e_6e_7e_8e_9e_{10}e_{11}e_{12}$	$k_3 = a^{P_3^{-1} \bmod \phi(N)} \bmod N$
4	$P_4 = e_4e_8e_9e_{10}$	$k_4 = a^{P_4^{-1} \bmod \phi(N)} \bmod N$
5	$P_5 = e_5e_9e_{10}$	$k_5 = a^{P_5^{-1} \bmod \phi(N)} \bmod N$
6	$P_6 = e_6e_{11}$	$k_6 = a^{P_6^{-1} \bmod \phi(N)} \bmod N$
7	$P_7 = e_7e_{11}e_{12}$	$k_7 = a^{P_7^{-1} \bmod \phi(N)} \bmod N$
8	$P_8 = e_8$	$k_8 = a^{P_8^{-1} \bmod \phi(N)} \bmod N$
9	$P_9 = e_9$	$k_9 = a^{P_9^{-1} \bmod \phi(N)} \bmod N$
10	$P_{10} = e_{10}$	$k_{10} = a^{P_{10}^{-1} \bmod \phi(N)} \bmod N$
11	$P_{11} = e_{11}$	$k_{11} = a^{P_{11}^{-1} \bmod \phi(N)} \bmod N$
12	$P_{12} = e_{12}$	$k_{12} = a^{P_{12}^{-1} \bmod \phi(N)} \bmod N$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

primes. This is reverse to the Akl-Taylor scheme. Considering there are more lower classes than upper classes in a practical hierarchy, the Harn-Lin scheme achieves improvement in storage space consumed by all users. However, in view of the greatest storage space required for a single user, the Harn-Lin scheme is the same as the Akl-Taylor scheme.

3.1.3 Huang-Yang Scheme

In [11], Huang and Yang proposed a scheme based on Akl-Taylor scheme to reduce the number of primes consumed in Akl-Taylor scheme. In Akl-Taylor scheme, each class is assigned with a distinct prime. In Huang-Yang scheme, a combination of primes is assigned to a class. For example, it assigns $\binom{10}{2} = 45$ pairs of primes to 45 classes, instead of assigning 10 distinct primes to 10 classes respectively. By reducing the number of primes, it is hoped that the storage space for the key materials for a class will be reduced.

The key assignment in Huang-Yang scheme is shown in Procedure 9. In the procedure, f is a one-way hash function.

If a class C_i tries to derive the secret key of class C_j, it runs the following key derivation algorithm as shown in Procedure 10.

Although the scheme is carefully designed with several attacking possibility in mind, [21] shows that it is insecure against the collusion attack whereby some security classes conspire to derive the secret keys of other leaf security classes. Here we show that some leaf security classes in a leaf group can conspire to derive secrets of other classes in the same leaf group.

Assume the leaf group $\{C_{i,1}, C_{i,2}, \cdots, C_{i,t}\}$ with secret keys $\{k_{i,1}, k_{i,2}, \cdots, k_{i,t}\}$, respectively, has a common ancestor C_j. Without loss of generality, we assume that $\{C_{i,1}, C_{i,2}, \cdots, C_{i,t-1}\}$ collude.

Denote $L = lcm(P_{i,1}, P_{i,2}, \cdots, P_{i,t})$. We can represent $K_{i,l}$ as follows. For

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

choose 2 large secret prime numbers p and q, and publish $N = pq$
choose a public $k_0 \in [2, n-1]$ such that $gcd(k_0, n) = 1$
for each class C_i **do**
  if C_i is not a leaf class **then**
    assign a distinct prime number e_i
    compute $d_i = e_i^{-1} \mod \phi(N)$
  else
    assign a distinct set of prime number $z_i = \{e_{i,1}, \cdots, e_{i,k}\}$
    compute $z_i' = \{d_{i,1}, \cdots, d_{i,k}\}$ where $d_{i,j} = e_{i,j}^{-1} \mod \phi(N)$, $j \in [1, k]$
  end if
end for
for each class C_i **do**
  if C_i is not a leaf class **then**
    computer P_i, the product of the distinct primes assigned to C_j where $C_j \leq C_i$
    CA assign P_i to C_i as its public parameter
    compute $k_i = a^{P_i^{-1} \mod \phi(N)} \mod N$
    assign k_i to C_i as its secret key
  else
    computer $P_i = \prod_{e_j \in z_i} e_j$
    assign P_i to each class as its public parameter
    compute $k_i = k_0^{f(C_i) \cdot \prod_{d_j \in z_i'} d_j \mod \phi(N)} \mod N$
    assign k_i to C_i as its secret key

  end if
end for

Procedure 9: Huang-Yang Key assignment

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$k_i = \begin{cases} k_j^{(P_j/P_i)f(C_i)} \mod N & \text{if } C_i \text{ is a leaf class,} \\ k_j^{P_j/P_i} \mod N & \text{if } C_i \text{ is a leaf class,} \end{cases}$$

Procedure 10: Huang-Yang Key derivation

$l \in [1, t]$,

$$k_{i,l} = k_j^{(P_j/P_{i,l})f(C_{i,l})} \mod N = (k'_j)^{Lf(C_{i,j})/P_{i,l}} \mod N$$

where $k'_j = k_j^{P_j/L}$.

By extended Euclidean algorithm, we can find $t-1$ integers $v_{i,l}$ for $l \in [1, t-1]$, such that

$$\sum_{t \in [1,t-1]} \frac{L}{P_{i,l}} f(C_{i,l}) v_{i,l} = \lambda$$

where

$$\lambda = \gcd(\frac{L}{P_{i,l}} f(C_{i,l}) : l \in [1, t-1])$$

Then we have

$$\prod_{l \in [1,t-1]} K_{i,l}^{v_{i,l}} = \prod_{l \in [1,t-1]} (K'_j)^{\frac{L}{P_{i,j}} f(C_{i,j}) v_{i,j}} \mod N = (K'_j)^{\lambda} \mod N$$

If $\lambda | (\frac{L}{P_{i,t} f(C_{i,t})})$, then $C_{i,1}, C_{i,2}, \cdots, , C_{i,t-1}$ can conspire to deduce $k_{i,t}$ as follows:

$$k_{i,t} = (k'_j)^{\frac{L}{P_{i,t}} f(C_{i,t})} = (k'_j)^{\lambda d} = (\prod_{l \in [1,t-1]} k_{i,l}^{v_{i,l}})^d$$

where

$$d = \frac{L \cdot f(C_{i,t})}{P_{i,t} \cdot \lambda}$$

The research in [21] shows that the probability of $\lambda | (\frac{L}{P_{i,t} f(C_{i,t})})$ is rather high. For example, in a leaf group of 10 members, each assigned with 2 primes out of 5 primes. If the output length of f is 48 bit, then the probability that a class can be attacked by others is greater that 90%.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2 Some Indirect Access Schemes

3.2.1 Sandhu Scheme

In [18], Sandhu proposed an access control scheme for *tree hierarchy* based on parameterized family of one-way functions constructed from encryption primitives such as DES. The tree hierarchy is a special case of a poset hierarchy where each class has at most one immediate predecessor. The key for a class is generated with its identity (ID) and the key of its immediate predecessor through a one-way function. In the scheme, no public parameters are needed for key derivation except for the ID of the classes.

A well known method to construct a one-way function is to encrypt some fixed and public known constant c using x as the key, i.e. $f(x) = \mathcal{E}_x(c)$ where $\mathcal{E}$ is the encryption algorithm of a block cipher. This can be generalized to obtain a family of one-way functions by replacing the constant c by a parameter p, that is $f_p(x) = \mathcal{E}_x(p)$. Now computing the inverse of $f_p(x)$ amounts to computing the key x given that p encrypted as $f(x)$. So this is a known plaintext attack which is infeasible for secure cryptosystems. Hence $f_p(x)$ is a one-way function for every p. The collection of functions $f_p(x)$ is called a parameterized family of one-way functions.

The key assignment procedure in Sandhu scheme is shown in Procedure 11.

assign an arbitrary key to the root security class.
for each class C_i **do**
  if C_j is an immediate successor of C_i **then**
    assign $k_j = f_{name(C_j)}(k_i) = \mathcal{E}_{k_i}(name(C_j))$ to C_j as its key.
  end if
end for

Procedure 11: Sandhu key assignment procedure

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Each class C_j can derive the key of its immediate successor. If $C_i \leq C_j$ but C_i is not an immediate successor of C_j, C_j needs to run the derivation procedure iteratively for each C_l that $C_i \leq C_l \leq C_j$ and finally derives k_i. The procedure is shown in Procedure 12. When a C_j is to derive the key of its successor C_i, it runs the procedure in Procedure 12.

if C_i is C_j's immediate successor **then**
  $k_i = \mathcal{E}_{k_j}(name(C_i))$
else
  C_j compute all keys in the path from C_j to C_i downwards until k_i is obtained
end if

Procedure 12: Sandhu key derivation procedure

Since the family of one-way functions is publicly known and the names of the security classes are public, a class can easily compute the key k_j for all security classes C_j covered by C_i; However it is computationally infeasible to compute k_j for a security class $C_j > C_i$; since this amounts to the inversion of one or more one-way functions.

Finally it should be computationally infeasible to compute k_j from k_i for C_j incomparable with C_i. To see what this entails consider the simple case where C_i and C_j are immediate successor of C_k. Then

$$k_i = \mathcal{E}_{k_k}(name(C_i))$$

$$k_j = \mathcal{E}_{k_k}(name(C_j))$$

By the assumed security of the $\mathcal{E}$ it is infeasible to compute k_j from k_i by solving the known plaintext problem of the former equation to derive k_k and then using the latter equation to compute k_j. For a strong cryptosystem we believe it can be safely assumed that there will also be no other tractable method of computing k_j from k_i in this situation. Moreover even if we know

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the keys for a large number of siblings it will be infeasible to compute the keys for a sibling outside the known set. That is collusion among the siblings is infeasible. Similar considerations apply to incomparable classes which are not siblings.

Compared with the direct access schemes, the required storage space in Sandhu scheme is reduced tremendously. However, this scheme can only be implemented for a tree hierarchy. The solution for the general case of an arbitrary poset was not given.

3.2.2 Zhong Scheme

In [24] Zhong proposed a solution that supports poset while inheriting the advantages of Sandhu scheme. This scheme is based on an ideal hash function $h : \mathcal{R} \times \mathcal{S} \rightarrow \mathcal{R}$ where $\mathcal{R}$ can be considered as a set of keys, $\mathcal{S}$ can be regarded as a set of class IDs. The hash function must be collision-free and modelled as a random oracle.

The key assignment procedure in Zhong's scheme is shown in Procedure 13[1].

When a class $C_j \geq C_i$ needs to derive the k_i, it runs the key derivation procedure as described in Procedure 14.

The security analysis is given as follows.

This scheme prevents classes from illegal derivation of keys. That is, a class can never derive a key that does not belong to any successor. In general, consider the class C_i. Suppose that C_i wants to derive k_h, where $C_h \not\leq C_i$. Therefore, C_i has to compute k_h from k_i. For each common predecessor C_j of these two security classes, these two secret parameters can be expanded

[1]In Zhong's scheme, the key assigned to C_i is $k_i \bigoplus P_i$ where P_i is picked by C_i. Here we simplify the description.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

for each class C_i **do**
  if C_i has no predecessor **then**
    CA picks $k_i \in \mathcal{R}$ uniformly at random
  else if C_i has one immediate precessor C_j **then**
    CA picks $a_{j,i} \in \mathcal{S}$
    $k_i \leftarrow h(k_j, a_{j,i})$
    publish $a_{j,i}$
  else
    {comment: C_i has more than one immediate predecessors $C_{j1}, \cdots, C_{jk}$}
    CA picks $a_{j1,i}, \cdots, a_{jk,i} \in \mathcal{S}$
    $k_i \leftarrow h(k_{j1}, a_{j1,i})$
    $O_{j2,i} \leftarrow k_i \bigoplus h(k_{j2,a_{j2,i}})$
    $\cdots$
    $O_{jk,i} \leftarrow k_i \bigoplus h(k_{j2,a_{j2,i}})$
    publish $a_{j1,i}, \cdots, a_{jk,i}$
    publish $O_{j2,i}, \cdots, O_{jk,i}$
  end if
end for

Procedure 13: Zhong key assignment procedure

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

if C_j is the single immediate predecessor of C_i **then**
  $k_i = hk_j, a_{j,i}$
else if C_j is C_i's immediate predecessor C_{j1} **then**
  $k_i \leftarrow h(k_{j1}, a_{j1,i})$
else if C_j is C_i's immediate predecessor $C_{jp}, p > 1$ **then**
  $k_i = O_{jp,i} \bigoplus h(k_{jp,a_{jp,i}})$
else
  C_j compute all keys in the path from C_j to C_i downwards until k_i is obtained
end if

Procedure 14: Zhong key derivation

according to the paths from C_j to them:

$$k_i = h(\ldots h(k_j, \ldots)\ldots);$$

$$k_h = h(\ldots h(k_j, \ldots)\ldots).$$

However, by the property of random oracle, $h(L, a)$ is independent of $h(L, a')$ if $a \neq a'$. Because $C_h \leq C_i$, the paths from C_j to C_h must diverge from the path from C_j to C_i at some point. Therefore, k_h must be independent of k_i. In other words, C_i cannot compute k_h from k_i. □

3.2.3 Zheng-Hardjono-Pieprzyk Scheme

In [23] Zheng et al. proposed a solution that supports poset while inheriting the advantages of Sandhu scheme. What is more important, in this proposal, the security of the scheme is analyzed based on a comprehensive security model instead of some ad hoc attacking scenarios. The security definition is as follows:

Definition 3.1 *Let* $\mathbb{C}$ *be the set of classes in a hierarchical organization.* $\mathbb{S}' \subset \mathbb{C}$, $\Theta(\mathbb{S}')$ *denotes the set of classes in* $\mathbb{S}'$ *and all the successors of* $\mathbb{S}'$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Let P be a polynomial, l an integer. Assume $|\mathbb{C}| = P(n)$. A key generation scheme for a hierarchical organization is secure if for any $\mathbb{S}' \subset \mathbb{C}$, for any class $C_i \notin \Theta(\mathbb{S}')$, for any polynomial Q and for all sufficiently large l, the probability that the classes in $\mathbb{S}'$ are able to find by collaboration the key k_i of the class C_i whenever C_i has no successor, or to simulate C_i 's procedure for generating the key of a successor of C_i whenever C_i is an internal class or the root class, is less than $\frac{1}{Q(l)}$.

Pseudo-random function families and sibling intractable function families (SIFF) are employed in this scheme. k-SIFF is a generalization of the universal one-way hash function family we have introduced. Similar to the universal one-way hash function, the k-SIFF has the following properties:

1. let $s = \lceil \log_2(k) \rceil$, k-SIFF maps $\{0,1\}^l$ to $\{0,1\}^{l-s}$
2. Given distinct $x_1, \cdots, x_j \in \{0,1\}^l, j \leq k$, it is easy to find $f \in k - SIFF$ such that $f(x_1) = f(x_2) = \cdots = f(x_j)$.
3. Given distinct $x_1, \cdots, x_j \in \{0,1\}^l, j < k$, and the $f \in k - SIFF$ such that $f(x_1) = f(x_2) = \cdots = f(x_j)$, it is infeasible to find a x' such that $f(x') = f(x_1) = f(x_2) = \cdots = f(x_j)$.

We define the notes for the key assignment and derivation procedures. Denote by ID_i the identity of the class C_i . Assume that every ID_i can be described by an $m(l)$-bit string, where m is a polynomial. Let $\mathcal{F} = \{\mathcal{F}_l\}$ be a pseudo-random function family, where $\mathcal{F}_l = \{f_K | f_K : \{0,1\}^{n(l)} \rightarrow \{0,1\}^l, K \in \{0,1\}^n\}$ and each function $f_K \in \mathcal{F}_l$ is specified by an l-bit string K. Let $\mathcal{H} = \bigcup_l \mathcal{H}_l$ be a k-SIFF mapping l-bit to l-bit output strings. Also assume that k is sufficiently large so that no nodes could have more than k parents. The key assignment procedure is described in Procedure 15.

When a class $C_j \geq C_i$ needs to derive the k_i, it runs the key derivation procedure as described in Procedure 16.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A random string $k_0 \in \{0,1\}^l$ is chosen for the root class
for each class C_i without a key **do**
  if the class C_i has a single immediate predecessor C_j **then**
    $k_i = f_{k_j}(ID_i)$
  else
    {comment: C_i has p immediate predecessors $C_{j_1}, C_{j_2}, \cdots, C_{j_p}$}
    a random $k_i \in \{0,1\}^l$ is chosen for C_i
    choose from $\mathcal{H}_l$ a function h_i such that
    $h_i(f_{k_{j_1}}(ID_i)) = h_i f_{k_{j_2}}(ID_i) = \cdots = h_i(f_{k_{j_p}}(ID_i)) = k_i$
    publish h_i
  end if
end for

Procedure 15: Zheng-Hardjono-Pieprzyk key assignment

if C_j is the single immediate predecessor of C_i **then**
  $k_i = f_{k_j}(ID_i)$
else if C_j is one of the immediate predecessors of C_i **then**
  $k_i = h_i(f_{k_j}(ID_i))$
else
  Compute all keys in the path from C_j to C_i downwards until k_i is obtained
end if

Procedure 16: Zheng-Hardjono-Pieprzyk key derivation

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

For the security of the scheme, [23] gives a proof sketch as follows:

Assume $\mathbb{S}' \subset \mathbb{C}$, $C_i \notin \Theta(\mathbb{S}')$. According to the definition for security, the following two cases are to be considered:

Case 1 : C_i has no successor and $\mathbb{S}'$ can directly find the key k_i of C_i .

Case 2 : C_i has one or more successors and $\mathbb{S}'$ can simulate C_i 's procedure for generating the key k_j of some successor C_j of C_i.

First discuss Case 1 where C_i has no successor. Note that the key k_i of C_i is derived from the key(s) of the predecessor(s) of C_i by the use of the pseudo-random function family. Therefore, obtaining k_i by $\mathbb{S}'$ implies that $\mathbb{S}'$ is able to predict the output of the pseudo-random function family, which is a contradiction.

Now consider Case 2 where C_i is an internal class or the root class, and $\mathbb{S}'$ can simulate C_i 's procedure for generating the key k_j of some successor C_j of C_i . Note that C_j may or may not be a member of $\Theta(S')$. For the key generation scheme, being able to simulate C_i 's procedure for generating the key C_j of the successor C_j of C_i implies being able to get either k_i when C_i is the single predecessor of C_j , or $f_{k_i}(ID_j)$ when C_j has other predecessor than C_i . Also note that getting k_i or $f_{k_i}(ID_j)$ means getting the keys of all the descendants of C_j besides the key k_j of C_j . Thus there are only two situations to be considered when $\mathbb{S}'$ is able to get k_i or $f_{k_i}(ID_j)$ but fails to mimic any of the immediate predecessors of C_i . These two situations are:

Situation 1 : C_i is an predecessor of some class(s) in $\Theta(\mathbb{S}')$.

Situation 2 : C_i is not the predecessor of any class in $\Theta(\mathbb{S}')$.

Consider Situation 1 first. Since C_i is an predecessor of a class in $\Theta(\mathbb{S}')$, there is a path from C_i to the class in $\Theta(S')$. C_i can derive the key of the class in $\Theta(\mathbb{S}')$ by evaluating the pseudo-random function family and (instances of)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the sibling intractable function family which appear in the path. Therefore, getting the key k_i of C_i or $f_{k_i}(ID_j)$ by $\mathbb{S}'$ implies that $\mathbb{S}'$ can do at least one of the following three actions: invert the pseudo-random function family, find a collision string for (instances of) the sibling intractable function family (appearing in the path from C_i to the class in $\Theta(\mathbb{S}')$, or invert (instances of) the sibling intractable function family. The success of any of these actions with a high probability is a contradiction. Comparing to Situation 1, Situation 2 is easier to analyze. Since C_i is not the predecessor of any class in $\Theta(\mathbb{S}')$, there is pass from a class in $\Theta(\mathbb{S}')$ to C_i . Thus getting k_i or $f_{k_i}(IDj)$ by $\Theta(\mathbb{S}')$ implies that $\Theta(\mathbb{S}')$ can predict the output of the pseudo-random function family. This is also a contradiction.

The construction of k-SIFF is similar to the construction of universal one-way hash function we presented in section 2.4.3. An example is as follows.

Let $s \geq 2^k$. On the finite field F_{2^m},

$$\mathcal{G}_m = \{g_{a,b} | g_{a,b}(x) = chop(a_0 + a_1 x + \cdots + a_{k-1}x^{x-1}), a_0, \cdots, a_{k-1} \in F_{2^m}\}$$

where all computation are in F_{2^m} and the function

$$chop : \{0,1\}^m \mapsto \{0,1\}^{m-s}$$

chops the last s bits.

Let f be an one-way permutation. Define $\mathcal{H}_m = \{h = g \circ f | g \in \mathcal{G}_m)\}$. Then $\bigcup_m(\mathcal{H}_m)$ is a k-universal one-way hash function.

[12] and [24] state that there are problems in implementation of Zheng's scheme in practice. But from the above example, we think the implementation is practical.

The security model in the Zheng-Hardjono-Pieprzyk scheme is actually equivalent to our security model in Definition 1.1. Yet a formal and rigorous proof can not be obtained directly from the above proof sketch. The argument in the proof is more "statement" than "proof". Detailed proof is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

still open there. For example, we consider the following simple scenario in Figure 3.1. Suppose $\mathbb{S}' = \{C_2, C_4\}$, the class they are going to attack is C_3.

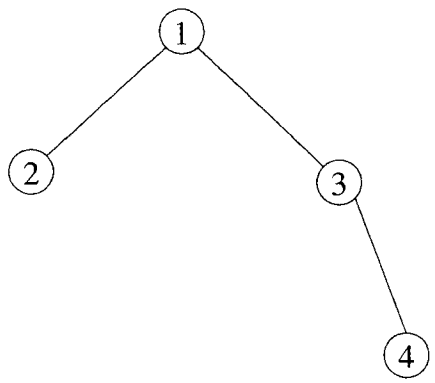

Figure 3.1: A simple attacking scenario: class 2 and class 4 conspire to attack class 3

This case falls into Case-2, Situation-1 in the proof sketch. According to the proof argument, $\mathbb{S}'$ has to invert the pseudo-random function family. We know if C_4 itself intends to attack C_3, it is safe to say that C_4 has to invert the pseudo-random function family. But now with the help of C_2, it is not obvious that C_4 has to invert the pseudo-random function family. What we need to prove here, is that with all the information held by C_2 and C_4, to compute k_3 is infeasible. The result can not be obtained directly from the argument in the proof sketch.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.3 Summary

In this chapter we reviewed some typical hierarchical access schemes. More schemes with variance including [5] [6] [7] [16] [22] . In view of the four requirements to the schemes, Zheng-Hardjono-Pieprzyk scheme is a outstanding one. The most significant part in this scheme is that it provides a security model that generalized all possible attack scenarios natural to the schemes. Also its performance in storage and dynamics is at least as good as others. Yet we think its security proof should be more formal and rigorous, thus is more persuasive and clear to be verified. For a scheme, a proof in the flavor the provable security, reducing some standard cryptographic primitives to the scheme, with each step firmly based on clear reasoning, would be more favorable. In the next parts, we are going to present a new scheme with same performance and security property as Zheng-Hardjono-Pieprzyk scheme, but with more rigorous formal security proof.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 4

Proposed Scheme

In this chapter, we propose a new hierarchical access control scheme.

4.1 Poset Representation

First we define how the poset is represented. For a given hierarchy structure, its corresponding poset $(\mathbb{C}, \leq)$ can be represented by a Hasse diagram, which is a graph whose vertices are classes of $\mathbb{C}$ and the edges correspond to the $\leq$ relation. An edge from $C_j \in \mathbb{C}$ to $C_i \in \mathbb{C}$ is present if $C_i < C_j$ and there is no $C_k \in \mathbb{C}$ such that $C_i < C_k$ and $C_k < C_j$. If $C_i < C_j$, then C_j is drawn higher than C_i. Because of that, the direction of the edges is not indicated in a Hasse diagram. Figure 1.1 shows an example of poset represented as a Hasse diagram.

4.2 Auxiliary Function

We introduce a function that will be used in our scheme below. Let $p = 2q+1$ where p, q are all odd primes. Let $\mathbb{G}$ be the subgroup of $\mathbb{Z}_p^*$ of order q. We

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

define a function $f : \mathbb{G} \rightarrow [1, q]$ as follows:

$$f(x) = \begin{cases} x; & x \leq q \\ p - x; & x > q \end{cases} \tag{4.1}$$

For any $x \in \mathbb{Z}_p^*$, if $x \in \mathbb{G}$, then $-x \notin \mathbb{G}$. So the above function is a bijection. If x is a random variable uniformly distributed on $\mathbb{G}$, $f(x)$ is uniformly distributed on $[1, q]$.

4.3 Key Management

The key management of the scheme consists of two procedures: the key assignment and the key derivation.

4.3.1 Key Assignment

The CA runs Procedure 17 to assign each class C_i its public parameters g_i, $h_{i,j}$ and a secret key k_i. The function f in the procedure is the auxiliary function presented above in (4.1).

For example, the classes in Figure 1.1 will be assigned with the secret key and public parameters as shown in Table 4.1.

4.3.2 Key Derivation

When a class C_j needs to compute the key of one successor C_i, it finds a path from itself to the successor in the Hasse diagram of the hierarchy. Starting from its immediate successor in the path, the class go through the path, and computes key of every successor along the path. The procedure of derivation is shown in Procedure 18.

For example, in Figure 1.1, class 1 is to derive the key of class 10. It finds the path $1 \rightarrow 3 \rightarrow 10$, and does the following computations:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 4.1: Example of key assignment

Node ID	secret key	public parameters
1	k_1	-
2	$k_2 = f(g_2^{k_1})$	g_2
3	$k_3 = f(g_3^{k_1})$	g_3
4	$k_4 = f(g_4^{k_2 k_3})$	$h_{4,2} = g_4^{k_3}, h_{4,3} = g_4^{k_2}$
5	$k_5 = f(g_5^{k_2})$	g_5
6	$k_6 = f(g_6^{k_3})$	g_6
7	$k_7 = f(g_7^{k_3})$	g_7
8	$k_8 = f(g_8^{k_4})$	g_8
9	$k_9 = f(g_9^{k_4 k_5})$	$h_{9,4} = g_9^{k_5}, h_{9,5} = g_9^{k_4}$
10	$k_{10} = f(g_{10}^{k_3 k_4 k_5})$	$h_{10,3} = g_{10}^{k_4 k_5}, h_{10,4} = g_{10}^{k_3 k_5}, h_{10,5} = g_{10}^{k_3 k_4}$
11	$k_{11} = f(g_{11}^{k_6 k_7})$	$h_{11,6} = g_{11}^{k_7}, h_{11,7} = g_{11}^{k_6}$
12	$k_{12} = f(g_{12}^{k_7})$	g_{12}

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CA chooses a group $\mathbb{Z}_p^*$, where $p = 2q + 1$, p and q are both large primes.
CA chooses $\mathbb{G}$, the subgroup of $\mathbb{Z}_p^*$ of order q
CA traverses the Hasse diagram from the root class with width-first algorithm, and
for each C_i **do**
  set g_i to be a unique generator of $\mathbb{G}$
  if C_i does not have any immediate predecessor **then**
    set k_i to be a number chosen from $[1, q]$ at random
  else if C_i has only one immediate predecessor C_j **then**
    $k_i = f(g_i^{k_j})$
  else
    {comment: C_i has more than one immediate predecessors}
    let $\mathcal{X}$ be the set of keys of C_i's immediate predecessors
    $x = \prod_{x_i \in \mathcal{X}} x_i$
    $k_i = f(g_i^x)$
    for all $x_j \in \mathcal{X}$ **do**
      $h_{i,j} = g_i^{x/x_j}$
    end for
  end if
end for

Procedure 17: Key Assignment

$k_3 = f(g_3^{k_1})$

$k_{10} = f(h_{10,3}^{k_3})$

4.3.3 Add a class

Let C_k be a new security class to be added into the hierarchy. The procedure to add the class is shown in Procedure 19.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

if C_j is the only one immediate predecessor of C_i **then**
  $k_i = f(g_i^{k_j})$
else if C_j is one of immediate predecessors of C_i **then**
  $k_i = f(h_{i,j}^{k_j})$
else
  {comment: C_j is not a immediate predecessor of C_i}
  compute all keys in the path from C_j to C_i downwards until k_i is obtained
end if

Procedure 18: Key Derivation

Example 4.1 *Let class C_{13} be the new class to be added in the hierarchy, serving as the immediate predecessor of C_4 and immediate successor of C_1, as shown in Figure 4.1. After the update, key of C_1 is set, and the keys and public parameters of the classes that are successors of C_{13}, including C_4, C_8, C_9 and C_{10}, are updated.*

4.3.4 Delete a classe

Let C_k be the class to be deleted from into the hierarchy. The procedure to delete the class is shown in Procedure 20.

Example 4.2 *Let class C_5 be the class to be deleted from the hierarchy. as shown in Figure 4.2. After the update, the keys and the public parameters of C_9 and C_{10} are changed.*

4.3.5 Add relation

A relationship between C_a and C_b added to the hierarchy so that $C_a \geq C_b$ is added. The procedure to add the relation is shown in Procedure 21.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CA randomly selects $g_k \in \mathbb{G}/\{1\}$ for C_i
if C_i does not have any immediate predecessor **then**
  set k_i to be a number chosen from $[1, q]$ at random
else if C_i has only one immediate predecessor C_j **then**
  $k_i = f(g_i^{k_j})$
else
  {comment: C_i has more than one immediate predecessors}
  let $\mathcal{X}$ be the set of keys of C_i's immediate predecessors
  $x = \prod_{x_i \in \mathcal{X}} x_i$
  $k_i = f(g_i^x)$
  for all $x_j \in \mathcal{X}$ **do**
    $h_{i,j} = g_i^{x/x_j}$
  end for
end if
for all C_i that is the successor of C_k **do**
  if C_i has only one immediate predecessor C_j **then**
    $k_i = f(g_i^{k_j})$
  else
    {comment: C_i has more than one immediate predecessors}
    let $\mathcal{X}$ be the set of keys of C_i's immediate predecessors
    $x = \prod_{x_i \in \mathcal{X}} x_i$
    $k_i = f(g_i^x)$
    for all $x_j \in \mathcal{X}$ **do**
      $h_{i,j} = g_i^{x/x_j}$
    end for
  end if
end for

Procedure 19: Add a new class

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

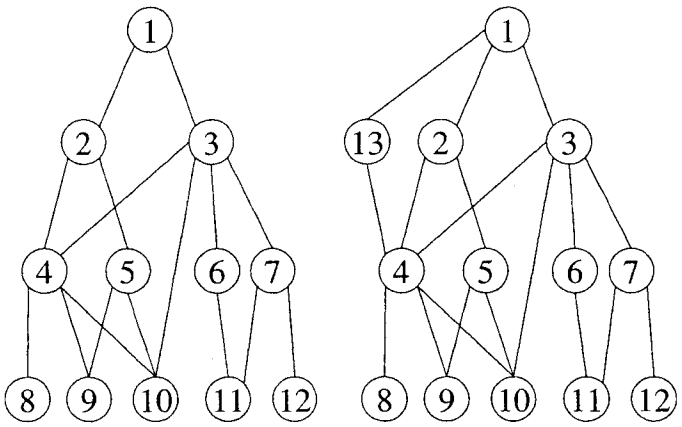

Figure 4.1: Add a class: class 13 is added.

Example 4.3 *Let edge from C_2 to C_6 is added to the hierarchy, as shown in Figure 4.3). After the update, the keys and the public parameters of C_6, C_{11} are changed.*

4.3.6 Delete relation

A relationship between C_a and C_b is delete from the hierarchy so that $C_a \geq C_b$ is deleted. The procedure to delete the relation is shown in Procedure 22.

Example 4.4 *Let edge from C_3 to C_4 is deleted from the hierarchy, as shown in Figure 4.4. After the update, the keys and the public parameters of C_4, C_8, C_9 and C_{10} are changed.*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Traverse the classes of C_k's successors with width-first algorithm, and
for each C_i that is C_k's successor **do**
  if C_i is an immediate successor of C_k **then**
    CA assigns a new generator g_i to C_i
  end if
  if C_i does not have any immediate predecessor other than C_k **then**
    set k_i to be a number chosen from $[1, q]$ at random
  else if C_i has only one immediate predecessor C_j other than C_k **then**
    $k_i = f(g_i^{k_j})$
  else
    {comment: C_i has more than one immediate predecessors other than C_k}
    let $\mathcal{X}$ be the set of keys of C_i's immediate predecessors (not include C_k)
    $x = \prod_{x_i \in \mathcal{X}} x_i$
    $k_i = f(g_i^x)$
    for all $x_j \in \mathcal{X}$ **do**
      $h_{i,j} = g_i^{x/x_j}$
    end for
  end if
end for
Delete C_k and and the edges connected to C_k

Procedure 20: Delete a class

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

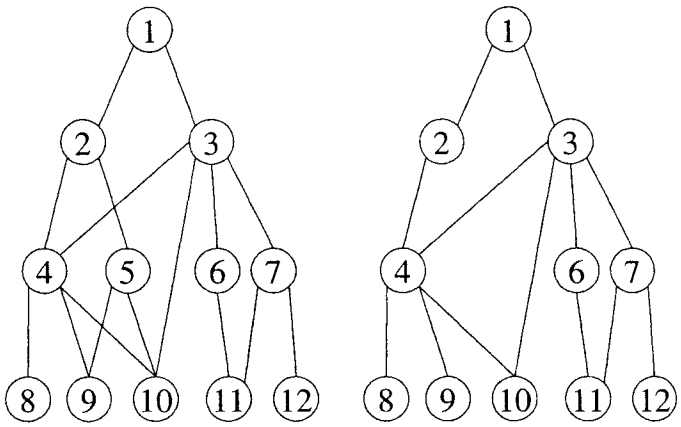

Figure 4.2: Delete a class: class 5 is deleted.

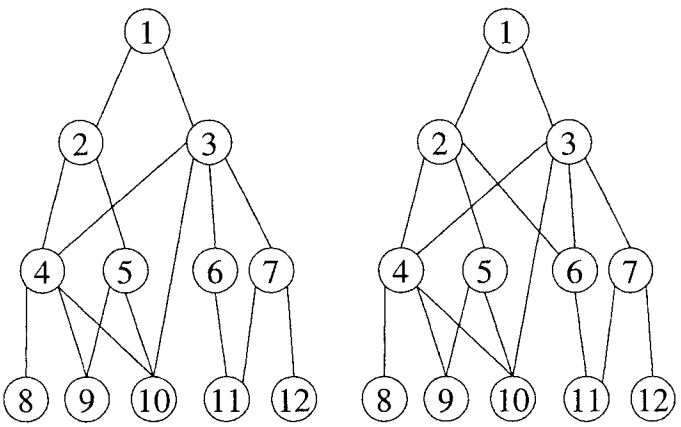

Figure 4.3: Add a relation: class 2 > class 6 is added.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Add an edge for the new raltion
Traverse the sub-graph consists of the class C_b and its successors with width-first algorithm, and
for each C_i traversed **do**
  if C_i has only one immediate predecessor C_j **then**
    $k_i = f(g_i^{k_j})$
  else
    {comment: C_i has more than one immediate predecessors}
    let $\mathcal{X}$ be the set of keys of C_i's immediate predecessors
    $x = \prod_{x_i \in \mathcal{X}} x_i$
    $k_i = f(g_i^x)$
    for all $x_j \in \mathcal{X}$ **do**
      $h_{i,j} = g_i^{x/x_j}$
    end for
  end if
end for

Procedure 21: Add a relation

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the CA assigns C_b with a new generator
Traverse the sub-graph consists of the class C_b and its successors with width-first algorithm, and run the following algorithm for each class:
if C_i has only one immediate predecessor C_j **then**
  $k_i = f(g_i^{k_j})$
else
  {comment: C_i has more than one immediate predecessors}
  let $\mathcal{X}$ be the set of keys of C_i's immediate predecessors
  $x = \prod_{x_i \in \mathcal{X}} x_i$
  $k_i = f(g_i^x)$
  for all $x_j \in \mathcal{X}$ **do**
    $h_{i,j} = g_i^{x/x_j}$
  end for
end if
Delete the edge for the relation

Procedure 22: Delete a relation

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

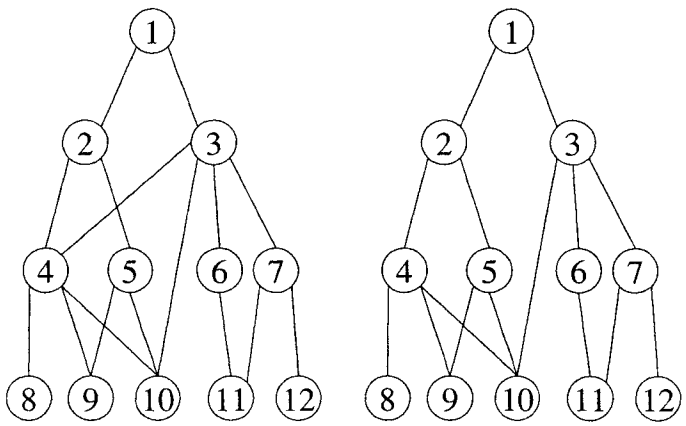

Figure 4.4: Delete a relation: class 3 > class 4 is deleted.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.4 Security Analysis

4.4.1 Standard Cryptographic Assumptions

On the group $\mathbb{G}$ used in our scheme, two standard assumptions, the discrete logarithm (DL) assumption and decisional Diffie-Hellman (DDH) assumption are believed to hold [3]. Another assumption, named group decisional Diffie-Hellman (GDDH) assumption is proved to hold on $\mathbb{G}$ too [20, 4]. To be concrete, let g be a generater of $\mathbb{G}$, a, b, c be random variables uniform on $[1, q]$, $\mathcal{X}$ be a set of random variables uniform on $[1, q]$, l be the binary length of q. Suppose $|\mathcal{X}|$ is polynomially bounded by l. Let $\prod(S)$ indicate the product of all elements in the set S. For any probabilistic polynomial time (in l) algorithms $\mathcal{A}$, any polynomial Q, for l large enough, the three assumptions are formally expressed as follows:

DL assumption:

$$P_r[\mathcal{A}(g, g^a) = a] < \frac{1}{Q(l)}$$

DDH assumption:

$$|P_r[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1] - P_r[\mathcal{A}(g, g^a, g^b, g^c) = 1]| < \frac{1}{Q(l)}$$

For convenience, we use the notation from [20] to simplify the expression. We say that the probabilistic distributions (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c) are polynomially indistinguishable, and denote them as

$$(g, g^a, g^b, g^{ab}) \approx_{poly} (g, g^a, g^b, g^c)$$

GDDH assumption:

$$|P_r[\mathcal{A}(g, g^{\prod(\mathcal{X})}, g^{\prod(S)} | S \subset \mathcal{X}) = 1] - P_r[\mathcal{A}(g, g^c, g^{\prod(S)} | S \subset \mathcal{X}) = 1]| < \frac{1}{Q(l)}$$

or denoted as

$$(g, g^{\prod(\mathcal{X})}, g^{\prod(S)} | S \subset \mathcal{X}) \approx_{poly} (g, g^c, g^{\prod(S)} | S \subset \mathcal{X})$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.4.2 Security Proof

The security of our scheme is based on the above three assumptions. In the following parts, we prove the scheme is secure under Definition 1.1. We suppose the number of classes in $\mathbb{C}$ is polynomially bounded by l (the binary length of $|\mathbb{G}|$), and all the algorithms considered below are polynomial time (in l) algorithms.

We choose an arbitrary class $C_t \in \mathbb{C}$ and suppose its secret key is k_t. Let $\mathbb{A}$ be the set of predecessors of C_t. We need to prove that, even when all the classes in $\mathbb{C} - \mathbb{A} - \{C_t\}$ conspire, it is computationally intractable for them to derive k_t.

We group the set $\mathbb{C} - \mathbb{A} - \{C_t\}$ into three subsets: $\mathbb{B}$ the set of classes in $\mathbb{C} - \mathbb{A}$ which do not have predecessors in $\mathbb{C} - \mathbb{A}$, and which is not C_t; $\mathbb{D}$ the set of classes that are immediate successors of C_t; $\mathbb{R} = \mathbb{C} - \mathbb{A} - \{C_t\} - \mathbb{B} - \mathbb{D}$. The followings relations between $\mathbb{B}$, $\mathbb{D}$ and $\mathbb{R}$ are direct from their definitions:

- $\mathbb{B} \cup \mathbb{D} \cup \mathbb{R} = \mathbb{C} - \mathbb{A} - \{C_t\}$
- $\mathbb{B} \cap \mathbb{D} = \emptyset$, $\mathbb{R} \cap \mathbb{B} = \emptyset$ and $\mathbb{R} \cap \mathbb{D} = \emptyset$
- the classes in $\mathbb{R}$ are successors of the classes in $\mathbb{B}$, or $\mathbb{D}$, or both

An example of the above partition is as follows: in Figure 3.1, suppose class 4 is the one we choose as the class C_t, then $\mathbb{A} = \{1, 2, 3\}, \mathbb{B} = \{5, 6, 7\}, \mathbb{D} = \{8, 9, 10\}, \mathbb{R} = \{11, 12\}$.

First we consider when all classes in $\mathbb{B}$ conspire, what information about k_t they can learn. Suppose the generator assigned to class C_t is g_t, $\mathcal{X}$ is the set of secret keys of the immediate predecessors of class C_t. Let $\prod(S)$ be the product of all elements in the set S. Let $x = \prod(\mathcal{X})$, then $k_t = g_t^x$. The public parameters of C_t are

$$\{g_t, g_t^{\prod(S)} | S \subset \mathcal{X} \text{ and } |S| = |\mathcal{X}| - 1\}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The classes $b_i \in \mathbb{B}$ with generators g_{b_i}, $i \in [1,n]$ may share the same predecessors with class C_t, thus may hold a subset of $\{g_{b_i}^{\prod(S)} | S \subseteq \mathcal{X}\}$ as their public parameters or secret keys. We assume that

$$\{g_{b_i}, g_{b_i}^{\prod(S)} | S \subseteq \mathcal{X}, i \in [1,n]\}$$

is all the information possibly held by classes in $\mathbb{B}$ that is related to k_t. So the public parameters of C_t, plus the information pertaining to k_t held by $\mathbb{B}$ is a subset of

$$\{g_t, g_t^{\prod(S)} | \mathcal{S} \subset \mathcal{X}\} \cup \{g_{b_i}, g_{b_i}^{\prod(S)} | \mathcal{S} \subseteq \mathcal{X}, i \in [1,n]\}$$

We have the following result showing that even all classes in $\mathbb{B}$ conspire, with the above information, they can not distinguish k_t from a random number on $[1,q]$. For convenient expression, the following theorem and its proof follow the notation style similar to that in [20].

Theorem 4.5 *Suppose DDH and GDDH assumptions hold on the group $\mathbb{G}$. Let c be a random variable uniform on $[1,q]$, $x = \prod(\mathcal{X})$. The two distributions*

$$V_{b_n} = \left(g_t^x, \{g_t, g_t^{\prod(S)} | \mathcal{S} \subset \mathcal{X}\}, \{g_{b_i}, g_{b_i}^{\prod(S)} | \mathcal{S} \subseteq \mathcal{X}, i \in [1,n]\}\right)$$

and

$$V'_{b_n} = \left(g_t^c, \{g_t, g_t^{\prod(S)} | \mathcal{S} \subset \mathcal{X}\}, \{g_{b_i}, g_{b_i}^{\prod(S)} | \mathcal{S} \subseteq \mathcal{X}, i \in [1,n]\}\right)$$

are indistinguishable.

Proof. From GDDH assumption we have

$$\left(g_t^x, \{g_t, g_t^{\prod(S)} | \mathcal{S} \subset \mathcal{X}\}\right) \approx_{poly} \left(g_t^c, \{g_t, g_t^{\prod(S)} | \mathcal{S} \subset \mathcal{X}\}\right)$$

A polynomial time algorithm can choose z uniformly from $[1,q]$ at random, and reduce the above GDDH distribution pair to

$$V_b = \left(g_t^x, \{g_t, g_t^{\prod(S)} | S \subset \mathcal{X}\}, g_t^z, (g_t^z)^x, \{(g_t^z)^{\prod(S)} | S \subset \mathcal{X}\}\right)$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$V'_{im} = \left(g_t^c, \{g_t, g_t^{\prod(S)} | S \subset \mathcal{X}\}, g_t^z, (g_t^z)^c, \{(g_t^z)^{\prod(S)} | S \subset \mathcal{X}\}\right)$$

respectively. It follows that

$$V_b \approx_{poly} V'_{im}. \tag{4.2}$$

Let c_1 be a random variable uniform on $[1, q]$. Since zc_1 is independent of z and c, from DDH, we have

$$(g_t, g_t^z, g_t^c, g_t^{zc}) \approx_{poly} (g_t, g_t^z, g_t^c, g_t^{zc_1})$$

A polynomial time (in l) algorithm can choose $\mathcal{X}$ that is a set of random variables uniform on $[1, q]$, and whose order is polynomially bounded by l, and reduce the above DDH distribution pair to

$$V'_{im} = \left(g_t^c, \{g_t, g_t^{\prod(S)} | S \subset \mathcal{X}\}, g_t^z, (g_t^z)^c, \{(g_t^z)^{\prod(S)} | S \subset \mathcal{X}\}\right)$$

$$V''_{im} = \left(g_t^c, \{g_t, g_t^{\prod(S)} | S \subset \mathcal{X}\}, g_t^z, (g_t^z)^{c_1}, \{(g_t^z)^{\prod(S)} | S \subset \mathcal{X}\}\right)$$

respectively. It follows that

$$V'_{im} \approx_{poly} V''_{im} \tag{4.3}$$

Similarly, by choosing z and c uniformly from $[1, q]$ at random, a polynomial time (in l) algorithm can reduce the GDDH distribution pair

$$\left(g_t^{c_1}, \{g_t, g_t^{\prod(\mathcal{S})} | \mathcal{S} \subset \mathcal{X}\}\right) \approx_{poly} \left(g_t^x, \{g_t, g_t^{\prod(\mathcal{S})} | \mathcal{S} \subset \mathcal{X}\}\right).$$

to

$$V''_{im} = \left(g_t^c, \{g_t, g_t^{\prod(S)} | S \subset \mathcal{X}\}, g_t^z, (g_t^z)^{c_1}, \{(g_t^z)^{\prod(S)} | S \subset \mathcal{X}\}\right)$$

$$V'_b = \left(g_t^c, \{g_t, g_t^{\prod(S)} | S \subset \mathcal{X}\}, g_t^z, (g_t^z)^x, \{(g_t^z)^{\prod(S)} | S \subset \mathcal{X}\}\right).$$

respectively. It follows that

$$V''_{im} \approx_{poly} V'_b \tag{4.4}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

From (4.2), (4.3) and (4.4), We conclude

$$V_b \approx_{poly} V_b'$$

i.e.,

$$\begin{aligned} & \left(g_t^x, \{g_t, g_t^{\Pi(S)} | S \subset \mathcal{X}\}, g_t^z, \{(g_t^z)^{\Pi(S)} | S \subseteq \mathcal{X}\}\right) \\ \approx_{poly} & \left(g_t^c, \{g_t, g_t^{\Pi(S)} | \mathcal{S} \subseteq \mathcal{X}\}, g_t^z, \{(g_t^z)^{\Pi(S)} | \mathcal{S} \subseteq \mathcal{X}\}\right). \end{aligned}$$

By choosing z_i, $i \in [1, n]$ uniformly from $[1, q]$ at random, a polynomial time algorithm can reduce V_b and V_b' to

$$\left(g_t^x, \{g_t, g_t^{\Pi(S)} | \mathcal{S} \subset \mathcal{X}\}, \{g_t^{zz_i}, (g_t^{zz_i})^{\Pi(S)} | \mathcal{S} \subseteq \mathcal{X}, i \in [1, n]\}\right)$$

$$\left(g_t^c, \{g_t, g_t^{\Pi(S)} | \mathcal{S} \subset \mathcal{X}, \{g_t^{zz_i}, (g_t^{zz_i})^{\Pi(S)} | \mathcal{S} \subseteq \mathcal{X}, i \in [1, n]\}\right)$$

It follows that

$$V_{b_n} \approx_{poly} V_{b_n}'.$$

This completes our proof. □

Then we consider when the classes in $\mathbb{B}$ and $\mathbb{D}$ conspire, what information about k_t they can learn. The classes $d_i \in \mathbb{D}$ assigned with generator g_{d_i}, $i \in [1, m]$ may hold a subset of the following information pertaining to k_t:

$$\{g_{d_i}, g_{d_i}^{k_t} | i \in [1, m]\}.$$

The following theorem shows that even all classes in $\mathbb{B}$ and $\mathbb{D}$ conspire, they can not derive k_t:

Theorem 4.6 *It is intractable for any polynomial time (in l) algorithm to derive g_t^x from*

$$\mathcal{I} = \{g_t, g_t^{\Pi(S)} | \mathcal{S} \subset \mathcal{X}\} \cup \{g_{b_i}, g_{b_i}^{\Pi(S)} | \mathcal{S} \subseteq \mathcal{X}, i \in [1, n]\} \cup \{g_{d_i}, g_{d_i}^{f(g_t^x)} | i \in [1, m]\},$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

i.e., for any polynomial time (in l) algorithm $\mathcal{A}$, any polynomial Q, if l is sufficiently large, then

$$P_r\left[\mathcal{A}\left(\mathcal{I}\right) = f(g_t^x)\right] < \frac{1}{Q(l)}.$$

Proof. For convenience, let

$$\mathcal{V} = \{g_t, g_t^{\prod(\mathcal{S})} | \mathcal{S} \subset \mathcal{X}\} \cup \{g_{b_i}, g_{b_i}^{\prod(\mathcal{S})} | \mathcal{S} \subseteq \mathcal{X}, i \in [1,n]\}.$$

Step 1. Assume that there exist a polynomial time (in l) algorithm B, a polynomial Q_1 and a number L, for $l > L$

$$P_r[B(\mathcal{V}, g_d, g_d^{f(g_t^x)}) = f(g_t^x)] \geq \frac{1}{Q_1(l)} \tag{4.5}$$

where g_d is a generator of $\mathbb{G}$.

Let c be a random variable uniform on $[1,q]$, $Q_2(l) = 2Q_1(l)$. Suppose l is large enough. We consider the following two cases

- **Case 1**: $P_r[B(\mathcal{V}, g_d, g_d^{f(g_t^c)}) = f(g_t^c)] \geq \frac{1}{Q_2(l)}$

  Notice that c is a random variable independent of $\mathcal{V}$. Let $z \in [1,q]$, we define the following algorithm $C(g_d, g_d^z)$:

choose a generator of $\mathbb{G}$ as g_t
choose a set of n distinct generators of $\mathbb{G}$ as $\mathbb{B}$
choose a set of random variables uniform on $[1,q]$ as $\mathcal{X}$
compute $\mathcal{V}$ with g_t, $\mathbb{B}$ and $\mathcal{X}$
return $B(\mathcal{V}, g_d, g_d^z)$

Algorithm 23: $C(g_d, g_d^z)$

The algorithm C is a polynomial time (in l) algorithm. Since $z = f(g_t^c)$ for some $c \in [1,q]$ (though we do not know c), we have

$$\begin{aligned} P_r[C(g_b, g_b^z) = z] &= P_r[B(\mathcal{V}, g_d, g_d^{f(g_t^c)}) = f(g_t^c)] \\ &\geq \frac{1}{Q_2(l)}. \end{aligned}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

This contradicts the DL assumption.

- **Case 2:** $P_r[B(\mathcal{V}, g_d, g_d^{f(g_t^c)}) = f(g_t^c)] < \frac{1}{Q_2(l)}$

  From this inequality and (4.5), we have

$$\begin{aligned} & P_r[B(\mathcal{V}, g_d, g_d^{f(g_t^x)}) = f(g_t^x)] - P_r[B(\mathcal{V}, g_d, g_d^{f(g_t^c)}) = f(g_t^c)] \\ \geq & \frac{1}{Q_1(l)} - \frac{1}{Q_2(l)} \\ = & \frac{1}{Q_2(l)} \end{aligned} \tag{4.6}$$

  Let $z \in \mathbb{G}$, we define the algorithm $D(\mathcal{V}, z)$ in Algorithm 24.

```
choose a generator of G as g_b
if B(V, g_d, g_d^{f(z)}) = f(z) then
  return 1
else
  return 0
end if
```

Algorithm 24: $D(\mathcal{V}, z)$

D is a polynomial time (in l) algorithm. From (4.6), we have

$$\begin{aligned} & P_r[D(\mathcal{V}, g_t^x) = 1] - P_r[D(\mathcal{V}, g_t^c) = 1] \\ = & P_r[B(\mathcal{V}, g_d, g_d^{f(g_t^x)}) = f(g_t^x)] - P_r[B(\mathcal{V}, g_d, g_d^{f(g_t^c)}) = f(g_t^c)] \\ \geq & \frac{1}{Q_2(l)}. \end{aligned}$$

That means D can distinguish the two distributions:

$$(\mathcal{V}, g_t^x) \text{ and } (\mathcal{V}, g_t^c).$$

This contradicts to Theorem 4.5.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Combining Case 1 and Case 2, we conclude that for any polynomial time (in l) algorithm B, any polynomial Q, for sufficiently large l,

$$P_r\left[B(\mathcal{V}, g_d, g_d^{f(g_t^x)}) = f(g_t^x)\right] < \frac{1}{Q(l)} \tag{4.7}$$

Step 2. Assume there exist a polynomial time (in l) algorithm $\mathcal{A}$, a polynomial Q and a number L such that for $l > L$,

$$P_r\left[\mathcal{A}\left(\mathcal{V}, \{g_{d_i}, g_{d_i}^{f(g_t^x)} | i \in [1,m]\}\right) = f(g_t^x)\right] \geq \frac{1}{Q(l)}.$$

Let $B(\mathcal{V}, g_d, g_d^{f(g_t^x)}) = \mathcal{A}(\mathcal{V}, \{g_d^{z_i}, g_d^{z_i f(g_t^x)} | i \in [1,m]\})$ where $z_1, \cdots, z_m$ are random variables uniform on $[1,q]$, and m is polynomially bounded by l. We have

$$\begin{aligned} P_r\left[B(\mathcal{V}, g_d, g_d^{f(g_t^x)}) = f(g_t^x)\right] &= P_r\left[\mathcal{A}(\mathcal{V}, \{g_d^{z_i}, (g_d^{z_i})^{f(g_t^x)} | i \in [1,m]\} = f(g_t^x)\right] \\ &\geq \frac{1}{Q(l)} \end{aligned}$$

This contradicts (4.7). Therefore for any polynomial time (in l) algorithm $\mathcal{A}$, any polynomial Q, for sufficiently large l,

$$P_r\left[\mathcal{A}\left(\mathcal{V}, \{g_{d_i}, g_{d_i}^{f(g_t^x)} | i \in [1,m]\}\right) = f(g_t^x)\right] < \frac{1}{Q(l)},$$

i.e.,

$$P_r\left[\mathcal{A}\left(\mathcal{I}\right) = f(g_t^x)\right] < \frac{1}{Q(l)}.$$

This completes our proof. □

Finally, we consider when all the classes in $\mathbb{B}$, $\mathbb{D}$, and $\mathbb{R}$ conspire, whether they are able to derive k_p. Since all the classes in $\mathbb{R}$ are successors of $\mathbb{B}$ or $\mathbb{D}$ or both, the information held by $\mathbb{R}$ can be derived by a polynomial time (in l) algorithm from the information held by $\mathbb{B}$ and $\mathbb{D}$. Thus if $\mathbb{B} \cup \mathbb{D} \cup \mathbb{R}$ can derive k_p, then $\mathbb{B} \cup \mathbb{D}$ can derive k_p. This contradicts to Theorem (4.6). Therefore we conclude that the scheme is secure under the security model defined in Definition (1.1).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 5

Summary

In this chapter, we briefly summarize the schemes we have reviewed and proposed. We set four criteria to evaluate the schemes. Because all these schemes except for Sandhu's scheme [18] support poset structure, we only compare these schemes in the other three criteria, i.e., storage requirement, dynamics and security.

5.1 Storage Requirement

Our scheme is an indirect access scheme, and has similar storage requirement with other indirect schemes. In a hierarchy with N classes where each class has at most M predecessors, the storage space required for a single class is about M for our scheme and other indirect schemes. For the direct schemes, to store the public information of one class, the maximum storage is about N numbers, or the product of the N numbers. In a real situation, N would be much greater than M, and N will increase as the scale of the hierarchy increases, while M usually keeps constant. So the indirect schemes achieves require less storage than the direct schemes.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.2 Dynamics

As an indirect hierarchical access scheme, the operation of adding, deleting a class or link in our scheme is similar to other indirect access schemes. When a class is added or deleted, or a link is added to or deleted from a class, only its successors are impacted, i.e., the secret key and public parameters of those classes need to be updated. The direct schemes are quite different. For example, in Akl-Taylor scheme, when a class is added or deleted, all the classes except for its successors have to update their secret keys and public parameters. In Harn-Lin scheme, when a class is added or deleted, all its predecessors will be impacted. In addition, for these two schemes, to prevent a deleted class to access its former successors, the keys of these successors have to be changed too. In a practical hierarchy, there are much more low level classes than high level classes, and it is more likely that the low level classes will change. Therefore in an indirect scheme, less classes are impacted than in a direct scheme when the hierarchy structure changes. The indirect schemes are more suitable than direct schemes for a dynamic hierarchy.

5.3 Security

Security is essential to a cryptographic scheme. The hierarchical access control schemes must withstand the cryptographic attacks. Every scheme shows that a class C_i cannot derive the key of another class C_j if C_i is not C_j's predecessor. Some of the schemes further analyzed that in some collusion attacks, the schemes are still secure. In these scheme a list of collusion attacking scenarios are listed and discussed. However, such lists do not elaborate all possible attacks. The first comprehensive security model is presented in Zheng-Hardjono-Pieprzyk scheme [23]. This security model covers all possible collusion attacks. In their scheme, Zheng et al. gave a proof sketch to

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

show that their scheme is secure under this security model. However, the proof sketch is not a very rigorous mathematic proof. Some statement in the sketch is not obviously tenable and need more detailed proof to make it persuasive. In our scheme, we take the provable security approach, and strictly proved that our scheme is secure as long as the standard DDH assumption holds. We would say so far our scheme is the first hierarchical access control scheme that is provable security under the comprehensive security model. Also the techniques used in the security proof in our scheme are helpful in analyzing the security of other schemes.

5.4 Conclusion

In this thesis we reviewed previous hierarchical access control schemes, proposed a new access control scheme for poset hierarchy. The new scheme supports any arbitrary poset, achieves the best performance of previous schemes, and provides a formal security proof under a comprehensive security model. None of the previous schemes achieved the properties as fully as ours does. Also our work provides useful techniques that facilitate the analysis of other schemes.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

References

[1] Selim G. Akl and Peter D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Trans. Comput. Syst.*, 1(3):239–248, 1983.

[2] Mihir Bellare and Phillip Rogaway. *Introduction to Modern Cryptography.* http://www-cse.ucsd.edu/users/mihir/cse207/classnotes.html.

[3] Dan Boneh. The decision diffie-hellman problem. In *ANTS-III: Proceedings of the Third International Symposium on Algorithmic Number Theory*, pages 48–63. Springer-Verlag, 1998.

[4] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. The group diffie-hellman problems. In *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*, pages 325–338. Springer, 2003.

[5] Brian J. Cacic. Indirect key derivation schemes for key management of access heirarchies. Master's thesis, Lakehead University, Thunder Bay, ON, Canada, 2004.

[6] Tzer-Shyong Chen, Yu-Fang Chung, and Chang-Sin Tian. A novel key management scheme for dynamic access control in a user hierarchy. In *COMPSAC*, pages 396–397, 2004.

[7] Gerald C. Chick and Stafford E. Tavares. Flexible access control with master keys. In *CRYPTO '89: Proceedings on Advances in cryptology*, pages 316–322. Springer-Verlag New York, Inc., 1989.

[8] Oded Goldreich. *Foundations of cryptography.* Cambridge University Press, Cambridge, UK.;New York, 2001.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[9] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[10] L. Harn and H.-Y. Lin. A cryptographic key generation scheme for multilevel data security. *Comput. Secur.*, 9(6):539–546, 1990.

[11] Min-Shiang Hwang and Wei-Pang Yang. Controlling access in large partially ordered hierarchies using cryptographic keys. *Journal of Systems and Software*, 67(2):99–107, 2003.

[12] Narn-Yih Lee and Tzonelih Hwang. A pseudo-key scheme for dynamic access control in a hierarchy. *J. Inf. Sci. Eng.*, 11(4):601–610, 1994.

[13] M Yung M Naor. Universal one-way hash functions and their cryptographic application. In *Proceedings of the 21st Annual ACM Symposium on the Theory of Computing*, pages 33–43, 1989.

[14] Stephen J. MacKinnon, Peter D. Taylor, Henk Meijer, and Selim G. Akl. An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. *IEEE Trans. Comput.*, 34(9):797–802, 1985.

[15] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[16] Indrakshi Ray, Indrajit Ray, and Natu Narasimhamurthi. A cryptographic solution to implement access control in a hierarchy and more. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 65–73. ACM Press, 2002.

[17] J Rompel. One-way functions are necessary and sufficient for digital signatures. In *Proceeding of 22nd Annual ACM Symposium on the Theory of Computing*, pages 387–394, 1990.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[18] Ravi S. Sandhu. Cryptographic implementation of a tree hierarchy for access control. *Inf. Process. Lett.*, 27(2):95–98, 1988.

[19] Claude E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949.

[20] Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-hellman key distribution extended to group communication. In *CCS '96: Proceedings of the 3rd ACM conference on Computer and communications security*, pages 31–37. ACM Press, 1996.

[21] Shyh-Yih Wang and Chi-Sung Laih. Cryptanalysis of hwang-yang scheme for controlling access in large partially ordered hierarchies, journal of systems and software. *Journal of Systems and Software*, 75(1-2):189–192, 2005.

[22] Cungang Yang and Celia Li. Access control in a hierarchy using one-way hash functions. *Computers & Security*, 23:659–664, 2004.

[23] Y. Zheng, T. Hardjono, and J. Pieprzyk. The sibling intractable function family (siff): notion, construction and applications. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science.*, E76-A(1):4–13, 1993.

[24] Sheng Zhong. A practical key management scheme for access control in a user hierarchy. *Computers & Security*, 21(8):750–759, 2002.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.