New Solutions to the Problem of Access Control in a Hierarchy [1]

Yuliang Zheng
Thomas Hardjono
Jennifer Seberry
The Centre for Computer Security Research
Department of Computer Science
University of Wollongong
Wollongong, NSW 2522, Australia
{yuliang,thomas,jennie}@cs.uow.edu.au

Abstract

The access control problem in a hierarchical organization consists of the management of information among a number of users who are divided into different security classes according to their suitability in accessing the information. Within the scope of cryptography the problem can be reduced to generating a cryptographic key for each security class in such a way that the key of a security class can be used to derive the keys of all lower security classes. This paper presents a new approach to solving the problem, based on pseudo-random function families, universal hash function families and in particular, sibling intractable function families. The approach is illustrated by two types of solutions. The first type of solution allows keys of lower security classes to be obtained indirectly from that of higher security classes through the calculation of the keys of all intermediate security classes, while the second type of solution allows keys of lower security classes to be obtained directly from that of higher security classes without involving other security classes. A formal definition of security for key generation schemes is introduced and the security of the proposed schemes is proven. Issues in key management are also addressed and several possible polices are suggested. The proposed solutions have theoretical significance in that their security relies only on the existence of any one-way function, and they also have practical applications in that they can be easily incorporated into existing information systems.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection — *Access controls; Cryptographic controls; Information flow controls;* H.2.0 [**Database Management**]: General — *Security, integrity and protection.* K.6.5 [**Management of Computing and Information Systems**]: Security and Protection — *Authentication; Insurance;*

General Terms: Algorithms, Security.

Additional Key Words and Phrases: Cryptography, Information Security, One-Way Hash Function, Sibling Intractable Function Family.

[1] This work was supported in part by Telecom Australia under the contract number 7027 and by the Australian Research Council (ARC) under the reference numbers A48830241, A49130102, A9030136, A49131885 and A49232172. A substantial part of the second author's work was completed when he was at the Centre for Computer Security Research.

1 Introduction

Controlling access to information stored in computer systems has been a topic of interest among computer researchers for over two decades. Various techniques and mechanisms have been suggested, among which are those based on cryptography. The simplest cryptography-based solution to control access to *plaintext* information in a computer system is to encipher it using a cryptosystem. A *key* is chosen as a parameter of the cryptosystem which then transforms the plaintext into a corresponding *ciphertext*. The decipherment of the ciphertext can be done using the same or a different (but related) key. The ability of a user to access the information is then expressed in terms of his or her having or knowing the deciphering key.

From this basic idea of controlling access by the encipherment of the required plaintext other problems have been posed by researchers in the field of cryptography and computer security. One important problem which derives from necessities in the real world is that of the organization and maintenance of keys in a hierarchical manner. Hence, the *hierarchical access control problem*, or the *access control problem in a hierarchy*, poses a question as to how users in an organization can access information in a computer system securely in a hierarchical manner, where users placed higher in the hierarchy can access information belonging to others lower down in the hierarchy.

1.1 The Hierarchical Access Control Problem

Using the notations found in [2] a more precise description of the problem is the following. The hierarchical access control problem consists of the management of sensitive information among a number of users in a computer system who are classified according to their suitability in accessing the information. The users of the hierarchy G are divided into $P(n)$ *security classes* $N_1, N_2, \ldots, N_{P(n)}$ and the relation $\geqq$ partially orders the set $S = \{N_1, N_2, \ldots, N_{P(n)}\}$ of security classes, where P is a polynomial and n is an integer called *security parameter*.

The hierarchy G is thus determined by S and $\geqq$, and in the partially ordered set (poset) S the notation $N_i \geqq N_j$ means that the security clearance of the users in class N_i is higher than that of the users in class N_j. Hence in the hierarchy the users in N_i may have access to information held by users in N_j or classified at the security level N_j.

The hierarchy of the poset itself can be viewed as a *Hasse diagram* where classes in S are represented as nodes in a directed graph. Nodes corresponding to the highest security level are located on the top of the graph, and are called *maximal nodes*. A maximal node N_i has the property that there is no other node $N_j \in S$ such that $N_j \geqq N_i$ and $N_j \neq N_i$. Given two nodes N_i and N_j, if $N_i \geqq N_j$ $(i \neq j)$ then N_i is called an *ancestor* of N_j, and if there is no other node $N_k \in S$ with $N_i \geqq N_k \geqq N_j$ then N_i is a *parent* of N_j, and N_j is a *child* of N_i. If N_i is an ancestor of N_j, we say that N_j is is a descendant of N_i.

In the following discussions we will use the term *security class* and *node* interchangeably, and the term $\mathrm{desc}(T)$ where $T \subset S$ is used to denote the *descendants* of T. The subset T induces a sub-poset that consists of the set $\Theta(T)$ and the partial

order relation $\geqq$, where $\Theta(T)$ consists of both the nodes in T and the nodes which are descendants of nodes in T. That is, $\Theta(T) = T \cup \mathrm{desc}(T)$.

Within the scope of cryptography the problem of access control in a hierarchy G can be reduced to that of generating a cryptographic key K_i for each node (security class) N_i in such a way that for any nodes N_i and N_j, the node N_i is able to derive from K_i the key K_j of N_j iff $N_i \geqq N_j$.

1.2 Relationship with Previous Work

The hierarchical access control problem was first addressed in the context of cryptography by Akl and Taylor in [1, 2]. They suggested a solution to the problem based on the Rivest-Shamir-Adleman (RSA) cryptosystem [12]. The heart of the solution was the suitable selection of exponents such that the key of a child node could be easily derived from the key of its parent. This solution was improved further by MacKinnon, Taylor, Meijer and Akl [9] in the form of an optimal algorithm for selecting the suitable exponents.

However, one remaining problem with the key generation scheme in [2] was the difficulty in adding new nodes (security classes) to the hierarchy. More specifically, when a new node N_j is to be added as a child of node N_i ($N_i \geqq N_j$) then new keys must also be generated for all the ancestor nodes of N_i who by definition have access to N_j. Another problem with the solution of Akl and Taylor was that its security was based on a particular cryptographic assumption, that is the (supposed) infeasibility of breaking the RSA cryptosystem. Furthermore, it made heavy use of the underlying algebraic properties of the crypto-function. In fact, these problems are common to many subsequently proposed solutions such as those in [4, 6, 11].

A key generation scheme for the special case of a *tree*-like hierarchy was suggested by Sandhu [14]. This solution was based on the use of a different one-way function to generate the key of each child of a node in the hierarchy. The selection of the one-way function for a child node is based on the name or identity of that child. In this way, the addition of a new child node N_j to an existing node N_i ($N_i \geqq N_j$) necessitates only the selection of an identity ID_j of node N_j and a one-way function f_{ID_j}. The key K_j for that new node N_j is then a function of the key K_i of its parent node. Unfortunately, Sandhu did not give any suggestion for the general case of a poset.

In this paper we show a number of solutions to the access control problem in a general hierarchy based on the use of pseudo-random function families, universal hash function families, and in particular, on the use of sibling intractable function families (SIFF) which was first introduced by Zheng, Hardjono and Pieprzyk in [16]. Note that both pseudo-random function families and sibling intractable function families can be constructed from one-way functions (see Sections 2.1 and 2.3). The solutions are in the same spirit as that given by Sandhu [14] in that each node has an identity or name and a single key. There are two types of solutions which will be presented, the first for indirect access to nodes and the second for direct access to nodes. The first type follows the same approach by previous authors in that access to a node further down in the hierarchy requires the traversal of the intermediate nodes, while the second type represents an improvement on this approach. Access to a node in

the hierarchy can be done directly without the need to involve other nodes.

The organization of this paper is as follows. Section 2 presents a brief review of one-way functions, pseudo-random function families, universal hash function families and sibling intractable function families. Section 3 makes use of the material reviewed in Section 2 and presents the key generation schemes for a hierarchy. In Section 4 we will discuss some issues related to the management and maintenance of keys in the hierarchy in the context of sibling intractable function families. Finally, Section 5 ends with some conclusions and remarks.

2 Background in Cryptography

Since the time of the initial solution based on cryptography given by Akl and Taylor in [2], research in cryptography has advanced considerably, particularly in the theoretical areas. The work in this paper is built upon some of these important theoretical results, in particular on pseudo-random function families. A formal definition of this concept, together with a definition of one-way functions will be reviewed in Sections 2.1. This is followed by the definition of universal hash function families in Section 2.2. Section 2.3 reviews the definition and construction of sibling intractable function families (SIFF) which will be an important primitive of our solutions to the access control problem in a hierarchy.

2.1 One-Way Functions and Pseudo-Random Function Families

Denote by $\mathbb{N}$ the set of all positive integers, n the security parameter, Σ the alphabet $\{0,1\}$ and $\sharp S$ the number of elements in a set S. By $x \in_R S$ we mean that x is chosen randomly and uniformly from the set S. The composition of two functions f and g is defined as $f \circ g(x) = f(g(x))$. Throughout the paper ℓ and m will be used to denote polynomials over $\mathbb{N}$. The formal definition of one-way functions is presented in the following.

Definition 1 *Let $f : D \rightarrow R$ be a polynomial time computable function, where $D = \bigcup_{n\in\mathbb{N}} \Sigma^{\ell(n)}$ and $R = \bigcup_{n\in\mathbb{N}} \Sigma^{m(n)}$. f is a* one-way function *if for each probabilistic polynomial time algorithm M, for each polynomial Q and for all sufficiently large n, $\Pr\{f_n(x) = f_n(M(f_n(x)))\} < 1/Q(n)$, where $x \in_R D_n$ and f_n denotes the restriction of f on $\Sigma^{\ell(n)}$.*

Let $F = \bigcup_{n\in\mathbb{N}} F_n$ be an infinite family of functions, where $F_n = \{f | f : \Sigma^{\ell(n)} \rightarrow \Sigma^{m(n)}\}$. We call F a function family mapping $\ell(n)$-bit input to $m(n)$-bit output strings. F is *polynomial time computable* if there is a polynomial time algorithm (in n) computing all $f \in F$, and *samplable* if there is a probabilistic polynomial time algorithm that on input $n \in \mathbb{N}$ outputs uniformly at random a description of $f \in F_n$.

Next, we review the definition of pseudo-random functions [5] which will be applied in Section 3. Intuitively, $F = \bigcup_{n\in\mathbb{N}} F_n$ is a pseudo-random function family if to a

probabilistic polynomial time algorithm, the output of a function f chosen randomly and uniformly from F_n, whose description is unknown to the algorithm, appears to be totally uncorrelated to the input of f, even if the algorithm can choose input for f. The formal definition is described in terms of *(uniform) statistical tests for functions.* A (uniform) statistical test for functions is a probabilistic polynomial time algorithm A that, given n as input and access to an oracle O_f for a function $f : \Sigma^{\ell(n)} \to \Sigma^{m(n)}$, outputs a bit 0 or 1. A can query the oracle only by writing on a special tape some $y \in \Sigma^{\ell(n)}$ and will read the oracle answer $f(y)$ on a separate answer-tape. The oracle prints its answer in one step.

Definition 2 *Let $F = \bigcup_{n\in\mathbb{N}} F_n$ be an infinite family of functions, where $F_n = \{f | f : \Sigma^{\ell(n)} \to \Sigma^{m(n)}\}$. Assume that F is both polynomial time computable and samplable. F is a* pseudo-random function family *iff for any statistical test A, for any polynomial Q, and for all sufficiently large n,*

$$|p_n^f - p_n^r| < 1/Q(n),$$

where p_n^f denotes the probability that A outputs 1 on input n and access to an oracle O_f for $f \in_R F_n$ and p_n^r the probability that A outputs 1 on input n and access to an oracle O_r for a function r chosen randomly and uniformly from the set of all functions from $\Sigma^{\ell(n)}$ to $\Sigma^{m(n)}$. The probabilities are computed over all the possible choices of f, r and the internal coin tosses of A.

In [5], it has been shown that pseudo-random function families can be constructed from any pseudo-random string generator. By the result of [8, 7], the existence of any one-way function is sufficient for the construction of pseudo-random function families.

2.2 Universal Hash Function Families

Universal hash function families, which were first introduced in [3] and further developed in [15], have played an essential role in many recent major results in cryptography and theoretical computer science (see for example [7, 8, 13]). Let $U = \bigcup_{n\in\mathbb{N}} U_n$ be a family of functions mapping $\ell(n)$-bit input into $m(n)$-bit output strings. For two strings $x, y \in \Sigma^{\ell(n)}$ with $x \neq y$, we say that x and y collide with each other under $u \in U_n$ or x and y are siblings under $u \in U_n$, if $u(x) = u(y)$.

Definition 3 *Let $U = \bigcup_{n\in\mathbb{N}} U_n$ be a family of functions that is polynomial time computable, samplable and maps $\ell(n)$-bit input into $m(n)$-bit output strings. Let k be a fixed positive integer. U is a* (strong) k-universal hash function family *if for all n, for all k (distinct) strings $x_1, x_2, \ldots, x_k \in \Sigma^{\ell(n)}$ and all k strings $y_1, y_2, \ldots, y_k \in \Sigma^{m(n)}$, there are $\sharp U_n / 2^{km(n)}$ functions in U_n that map x_1 to y_1, x_2 to y_2, ..., x_k to y_k.*

The following definition of the *collision accessibility property* is presented due to its importance and useful role in universal hash function families, and later in the definition and construction of sibling intractable function families.

Definition 4 *Let $U = \bigcup_{n\in\mathbb{N}} U_n$ be a family of functions that is polynomial time computable, samplable and maps $\ell(n)$-bit input into $m(n)$-bit output strings. Let k be a fixed positive integer. U has the* collision accessibility property *if for all n and for all $1 \leqq j \leqq k$, given any set $X = \{x_1, x_2, \ldots, x_j\}$ of j initial strings in $\Sigma^{\ell(n)}$, it is possible in probabilistic polynomial time to select randomly and uniformly functions from U_n^X, where $U_n^X \subset U_n$ is the set of all functions in U_n that map x_1, x_2, ..., and x_j to the same strings in $\Sigma^{m(n)}$.*

Strong k-universal hash function families with the collision accessibility property can be obtained from polynomials over finite fields [3, 15]. We denote by P_n the collection of all polynomials over $GF(2^{\ell(n)})$ with degree less than k. That is,

$$P_n = \{a_0 + a_1 x + \cdots + a_{k-1}x^{k-1} | a_0, a_1, \ldots, a_{k-1} \in GF(2^{\ell(n)})\}.$$

For each $p \in P_n$, let u_p be the function obtained from p by chopping the first $\ell(n) - m(n)$ bits of the output of p whenever $\ell(n) \geqq m(n)$, or by appending a fixed $m(n) - \ell(n)$-bit string to the output of p whenever $\ell(n) < m(n)$. Let $U_n = \{u_p | p \in P_n\}$, and $U = \bigcup_{n\in\mathbb{N}} U_n$. Then U is a strong k-universal hash function family, which maps $\ell(n)$-bit input into $m(n)$-bit output strings and has the collision accessibility property.

2.3 Sibling Intractable Function Families

The notion of the sibling intractable function family was first introduced by Zheng *et al.* in [16]. In this section we will provide a definition of the sibling intractable function family and explain briefly some aspects relevant to the current work. The reader is directed to [16] for a more comprehensive explanation of the sibling intractable function family.

Let $H = \bigcup_{n\in\mathbb{N}} H_n$, where $H_n = \{h | h : \Sigma^{\ell(n)} \to \Sigma^{m(n)}\}$, be an infinite family of functions that is polynomial time computable, samplable and has the collision accessibility property. Also let k be a fixed integer, $X = \{x_1, x_2, \ldots, x_k\}$ a set of k initial strings in $\Sigma^{\ell(n)}$ and h a function in H_n that maps $x_1, x_2, \ldots, x_k$ to the same string. Let F, called a *sibling finder*, be a probabilistic polynomial time algorithm that on input X and h, outputs either "?" ("I cannot find") or a string $x' \in \Sigma^{\ell(n)}$ such that $x' \notin X$ and $h(x') = h(x_1) = h(x_2) = \cdots = h(x_k)$. Informally, H is a $(k, 1)$-*sibling intractable function family*, or $(k, 1)$-SIFF for short, if for any sibling finder F, the probability that F outputs an x' is negligible. In other words, H is a $(k, 1)$-SIFF if it is infeasible for any probabilistic polynomial time algorithm to find a string x' so that x' collides with the strings $x_1, x_2, \ldots, x_k$, which all collide with one another under h. More precisely:

Definition 5 *Let H be a family of functions that is polynomial time computable, samplable, has the collision accessibility property and maps $\ell(n)$-bit input into $m(n)$-bit output strings. Let $X = \{x_1, x_2, \ldots, x_k\}$ be any set of k initial strings. H is a* $(k, 1)$-sibling intractable function family*, or simply $(k, 1)$-SIFF, if for each sibling finder F, for each polynomial Q, and for all sufficiently large n,*

$$\Pr\{F(X, h) \neq ?\} < 1/Q(n),$$

where h is chosen randomly and uniformly from $H_n^X \subset H_n$, the set of all functions in H_n that map x_1, x_2, ..., and x_k to the same strings in $\Sigma^{m(n)}$, and the probability $\Pr\{F(X,h) \neq ?\}$ is computed over H_n^X and the sample space of all finite strings of coin flips that F could have tossed.

In [16], an explicit construction of SIFF from any one-way function was given. The construction begins with the universal one-way hash function family introduced in [10], which can be constructed from any one-way function [13], and gives a method for transforming any universal one-way hash function family into a $(2^s - 1, 1)$-SIFF for any $s = O(\log n)$. The transformation method is presented here for completeness: *Let ℓ, m' and m be polynomials with $m'(n) - m(n) = O(\log n)$. Let $k = 2^{m'(n)-m(n)} - 1$. Assume that $H' = \bigcup_{n\in\mathbb{N}} H'_n$ is a $(1,1)$-SIFF mapping $\ell(n)$-bit input to $m'(n)$-bit output strings, and $U = \bigcup_{n\in\mathbb{N}} U_n$ a $(k+1)$-universal hash function family that has the collision accessibility property and maps $m'(n)$-bit input to $m(n)$-bit output strings. Let*

$$H_n = \{u \circ h' | h' \in H'_n, u \in U_n\},$$

and let $H = \bigcup_{n\in\mathbb{N}} H_n$. Then H is a $(k,1)$-SIFF mapping $\ell(n)$-bit input into $m(n)$-bit output strings.

The significance of a SIFF which distinguishes it from a universal hash function family is precisely its sibling intractable property. Informally, consider the case when a collection of input strings are to be mapped to the same output string under a function which is an instance of a SIFF. The word *sibling* in this case refers specifically to the collection of input strings to that instance of SIFF. Hence, one input string is said to be a sibling of another input string, both of which belong to the collection. The sibling intractable property ensures that the knowledge of several input strings from the collection will not allow the discovery of one or more siblings of those input strings. The sibling intractable property represents a new concept in cryptography, one that will be used to provide a simple solution to the problem of access control in a hierarchy.

3 Key Generation Schemes for a Hierarchy

The solution to the problem of access control in a hierarchy based on cryptographic techniques can be expressed in terms of providing a cryptographic key generation and management scheme which would allow easy access to information classified at nodes and easy maintenance of keys associated with the nodes in the case of addition or deletion of nodes from the hierarchy.

From the previous work by other authors we can summarize the following points concerning the keys in a hierarchy:

1. The generation of a key for a node is based on the assumption that the illegal key re-calculation by unauthorized users is equivalent to solving an intractable problem. This notion was realized in [2] by the use of encipherment procedures which rely on the infeasibility of factorization (more specifically the RSA

cryptosystem) and in [14] by the use of one-way functions. Note that the work in [14] treated only the special case of tree structures.

2. There is a relationship between the key assigned to a node and those assigned to its children which is established through the key generation scheme being employed. In [2] this relationship was realized in the selection of exponents of a key to satisfy certain criterion which mathematically allowed the key of a child node to be derived from the key of its parent, while in [14] the relationship was established through the use of one-way functions selected on the basis of the name (identity) of the child node.

In this section we will discuss two types of key generation schemes. The first type allows nodes high in the hierarchy (having high security classification) to access *indirectly* information in nodes lower down in the hierarchy (having lower security classification). The second type allows nodes of high security classification to access *directly* information in nodes of lower security classification. One important assumption regarding the key generation schemes is that a trusted party or the maximal node(s) must perform securely all the tasks associated with the key generation schemes. This includes the determination of certain security parameters and functions, and selection and distribution of keys. However, before proceeding with a description of our key generation schemes it would be useful to define the notion of security in the context of access control in a hierarchy.

3.1 The Definition of Security

Recall that $\mathrm{desc}(T)$ for $T \subset S$ denotes the descendants of the nodes in T, and recall that T induces a sub-poset which consists of the set $\Theta(T)$ and the partial order relation $\geqq$, where $\Theta(T) = T \cup \mathrm{desc}(T)$.

Informally, a key generation scheme is said to be *secure* if it is computationally infeasible for users in a subset T of S to find (individually or by collaboration) the key K_i of a node N_i not in $\Theta(T)$. In a formal definition to be stated below, the computational power of nodes in a hierarchy (organization) is assumed to be bounded by probabilistic polynomial time. As in most standard formal definitions for security, our definition is also presented in terms of asymptotic behavior. In other words, we consider the asymptotic performance of a key generation scheme, when the number of nodes in a hierarchy increases as the security parameter n. The formal definition is as follows.

Definition 6 *Let G be a hierarchy with $P(n)$ nodes. Denote by S the set of the $P(n)$ nodes. A key generation scheme for a hierarchy is* secure *if for any $T \subset S$, for any node $N_i \notin \Theta(T)$, for any polynomial Q and for all sufficiently large n, the probability that the nodes in T are able to* find *by collaboration together the key K_i of the node N_i is less than $1/Q(n)$.*

3.2 Key Generation for Indirect Access to Nodes

In the same manner as Sandhu [14], our first key generation scheme begins with the selection of a name or identity ID_j for each node N_j in the hierarchy and we assume that every ID_j can be described by an $\ell(n)$-bit string, where ℓ is a polynomial. Next, we let $F = \bigcup_{n\in\mathbb{N}} F_n$ be a pseudo-random function family, where $F_n = \{f_K | f_K : \Sigma^{\ell(n)} \rightarrow \Sigma^n, K \in \Sigma^n\}$ and each function $f_K \in F_n$ is specified by an n-bit string K. In addition, let $U = \bigcup_{n\in\mathbb{N}} U_n$ be a k-universal hash function family that maps n-bit input to n-bit output strings. The integer k is assumed to be **larger than** the total number of parents of any node in the hierarchy.

Scheme 1 *A secure scheme for indirect node access.* Firstly keys for maximal nodes are generated, which is done by simply selecting random strings of length n as their keys. For the remaining nodes, there are two methods to generate the keys. The first method is applied to nodes which have only a single parent node, while the second method is used for nodes which have more than one parent node.

1. *Nodes with one parent*
   Given a node N_j with its parent N_i which has already been assigned a key K_i, the key to be assigned to N_j is the n-bit string

$$K_j = f_{K_i}(ID_j) \tag{1}$$

2. *Nodes with two or more parents*
   Given a node N_j with all its p ($p \leqq k$) parents N_{i_1}, N_{i_2}, ..., N_{i_p} having been assigned keys K_{i_1}, K_{i_2}, ..., K_{i_p}, the key K_j for N_j is chosen as a random string $K_j \in_R \Sigma^n$.

   From U_n an instance (function) u_j is chosen randomly and uniformly such that $f_{K_{i_1}}(ID_j)$, $f_{K_{i_2}}(ID_j)$, ..., $f_{K_{i_p}}(ID_j)$ are mapped to K_j. That is,

$$u_j(f_{K_{i_1}}(ID_j)) = u_j(f_{K_{i_2}}(ID_j)) = \cdots = u_j(f_{K_{i_p}}(ID_j)) = K_j \tag{2}$$

   The function u_j is then made public together with the fact that it is associated with node N_j.

Now we prove that this scheme based on a pseudo-random function family and a universal hash function family is secure.

Theorem 1 *Scheme 1 is secure.*

Proof. Assume for contradiction that there is an infinite subset $\mathbb{N}'$ of $\mathbb{N}$ and a polynomial Q, such that for each $n \in \mathbb{N}'$ there is a subset $T \subset S$ which can find with probability at least $1/Q(n)$ the key K_i of a node $N_i \notin \Theta(T)$. Since obtaining K_i means obtaining the keys of all the descendants of N_i, there are only two situations to be considered when T fails to find the key of any parent of N_i. These two cases are:

Case-1 N_i is an ancestor of some node(s) in $\Theta(T)$.

Case-2 N_i is not the ancestor of any node in $\Theta(T)$.

Note that since T contains only polynomially many nodes and the computational power of nodes is bounded by probabilistic polynomial time, the whole computational activities of T during the finding of K_i can be simulated by a probabilistic polynomial time algorithm. Thus, Case-1 implies that there is an probabilistic polynomial time algorithm that can invert with probability at least $1/Q(n)$ the pseudo-random function family for all $n \in \mathbb{N}'$, while Case-2 implies that there is an probabilistic polynomial time algorithm that can predict with probability at least $1/Q(n)$ the output of the pseudo-random function family, also for all $n \in \mathbb{N}'$. These are contradictions. □

3.3 Key Generation for Direct Access to Nodes

In the scheme presented by Akl and Taylor in [2] and in its subsequent improvements the key for a node was related only to the key of its parent. This is also true in the case of the key generation scheme for indirect access to nodes presented in Section 3.2. However, one important impracticality stems precisely from this limited relationship between a node and its parent(s). A user at a node, which by definition has access to information stored at all its descendant nodes, must traverse (and thus compute the keys of) all the intermediate nodes between his or her node and the desired (non-child) descendant node at which the information he or she wishes to access is stored. This may prove highly impractical in many situations.

In order to remedy this impracticality it would be desirable for a user at a node to have the ability to access information at any descendants of his or her current node in a minimum number of steps. This can be achieved by improving the previous key generation scheme as will be shown in the following.

Let $U = \bigcup_{n\in\mathbb{N}} U_n$ be a k-universal hash function family that maps n-bit input to n-bit output strings. Here k is assumed to be greater than the total number of ancestors of any node in the hierarchy. In addition, let $H = \bigcup_{n\in\mathbb{N}} H_n$ be a $(k,1)$-SIFF mapping n-bit input to n-bit output strings. As in Scheme 1, assume that $F = \bigcup_{n\in\mathbb{N}} F_n$ is a pseudo-random function family, where $F_n = \{f_K | f_K : \Sigma^{\ell(n)} \rightarrow \Sigma^n, K \in \Sigma^n\}$ and each function $f_K \in F_n$ is specified by an n-bit string K. We present three possible key generation schemes which allow direct access to information at descendants nodes. The three key generation schemes for direct access to nodes consist of two general phases, the first is common to all three key generation schemes, while the second is unique to each scheme. These two phases are:

- The selection of n-bit keys uniformly and randomly for each node in the structure.
- The establishment of relationships or "links" among nodes in the structure.

Assume that the set of nodes N_{i_1}, N_{i_2}, ..., N_{i_p} are the p immediate parents of a node N_j. Assume further that the m nodes $N_{i_{p+1}}$, $N_{i_{p+2}}$, ..., $N_{i_{p+m}}$ are the

other ancestors of N_j. Note that $p + m \leqq k$. The first phase simply requires that the keys $K_{i_1}, \ldots, K_{i_p}, K_{i_{p+1}}, \ldots, K_{i_{p+m}}$ be chosen uniformly and randomly for the corresponding nodes. Of more interest is the second phase in which relationships (or "links") between nodes are established. The links to be created are not only between a given node and its parents, but also between it and each of its (non-parent) ancestors. Based on universal hash function families, sibling intractable function families and pseudo-random function families the three key generation schemes for the second phase are as follows.

Scheme 2 *A secure scheme for direct node access.* Choose uniformly and randomly from U_n an instance u_j such that:

$$u_j(f_{K_{i_1}}(ID_j)) = u_j(f_{K_{i_2}}(ID_j)) = \cdots = u_j(f_{K_{i_p}}(ID_j)) = u_j(f_{K_{i_{p+1}}}(ID_j)) = u_j(f_{K_{i_{p+2}}}(ID_j)) = \cdots = u_j(f_{K_{i_{p+m}}}(ID_j)) = K_j \quad (3)$$

The function u_j is made public together with the fact that it is associated with node N_j.

Like Scheme 1, Scheme 2 is also secure. Note that both schemes rely on the use of pseudo-random function families and universal one way hash function families. As in Scheme 1, it is computationally infeasible for a user at a node to obtain the key of any non-descendants of that node. If the user can find the key of one of his or her ancestors then the user is able to invert the pseudo-random function family. On the other hand, if the user is able to obtain the key of a node which is neither an ancestor nor a descendant of that user's node then he or she is also able to predict the result of the pseudo-random function family. Hence, these two actions are contradictory to the definition of pseudo-random function families. Thus we have the following theorem.

Theorem 2 *Scheme 2 is secure.*

Scheme 3 *A more secure scheme for direct node access.* Choose uniformly and randomly from H_n an instance h_j such that:

$$h_j(f_{K_{i_1}}(ID_j)) = h_j(f_{K_{i_2}}(ID_j)) = \cdots = h_j(f_{K_{i_p}}(ID_j)) = h_j(f_{K_{i_{p+1}}}(ID_j)) = h_j(f_{K_{i_{p+2}}}(ID_j)) = \cdots = h_j(f_{K_{i_{p+m}}}(ID_j)) = K_j \quad (4)$$

The instance h_j of the $(k, 1)$-SIFF h_j is made public together with the fact that it is associated with node N_j.

Here, instead of a universal one-way hash function family, a sibling intractable function family is employed. By a similar argument to that for Scheme 2, we have the next theorem.

Theorem 3 *Scheme 3 is secure.*

Note that a universal hash function family is used in Scheme 2. In general a universal hash function family is *not* one-way. Let N_i, N_j and N_k be three nodes in a hierarchy, where both N_j and N_k are ancestors of N_i, but neither is N_j an ancestor of N_k nor is N_k an ancestor of N_j. Although it is provably infeasible for N_j to obtain the key K_k of N_k, it is easy for N_j to obtain $f_{K_k}(ID_i)$. This may prove to be a problem in certain circumstances, where the ability to obtain $f_{K_k}(ID_i)$ may represent a kind of privilege of N_k which should not be available to any non-ancestor of N_k.

This problem does not exist in Scheme 3. Therefore Scheme 3 represents an improvement in security over both Scheme 1 and Scheme 2. The argument necessary is similar to the previous two schemes. The ability of a user at a node to calculate the key of another non-descendant or non-ancestor node is equivalent to predicting outputs of a pseudo-random function family, and thus represents a contradiction to the definition of pseudo-random function families. Furthermore, if a user is able to obtain $f_{K_i}(ID_j)$ of an ancestor node N_i, then this represents an ability to invert or to find a collision string for the sibling intractable function family, both cases of which have a negligible probability of occurring. If N_i is not an ancestor of the user then the user has the ability to predict the outputs of a pseudo-random function family, something that was concluded to be a contradiction. Thus we have shown that indeed Scheme 3 has higher security than Scheme 2.

Scheme 4 *A pragmatic scheme for direct node access.* Choose uniformly and randomly from H_n an instance h_j such that:

$$h_j(K_{i_1}) = h_j(K_{i_2}) = \cdots = h_j(K_{i_p}) = \\ h_j(K_{i_{p+1}}) = h_j(K_{i_{p+2}}) = \cdots = h_j(K_{i_{p+m}}) = K_j \tag{5}$$

The instance h_j of the $(k, 1)$-SIFF is made public together with the fact that it is associated with node N_j.

The security level attained in Scheme 4 is difficult to be shown for the following reason. Unlike in the previous schemes, in Scheme 4 the key of a node is used directly as input to the instances of SIFF that are associated with the descendants of that node. This direct application of the node's key results in the difficulty in measuring the amount of information concerning that node that is obtainable by its descendants in the case when these descendants are collaborating against that node. As a consequence, this difficulty in measuring the obtainable information has led to difficulty in providing a formal proof of the security of this scheme. However, this scheme represents an improvement from the previous three schemes in terms of practicalities since only a sibling intractable function family is employed without the use of a pseudo-random function family. Thus, key generation and access to nodes is comparatively faster than in the previous three schemes.

By using any one of these last three key generations schemes the m $(m + p \leqq k)$ ancestor nodes $N_{i_{p+1}}$, $N_{i_{p+2}}$, ..., $N_{i_{p+m}}$ of node N_j have access directly to node N_j. This is shown in a simplified manner in Figure 1 where the broken lines indicates direct paths from a node to its descendants.

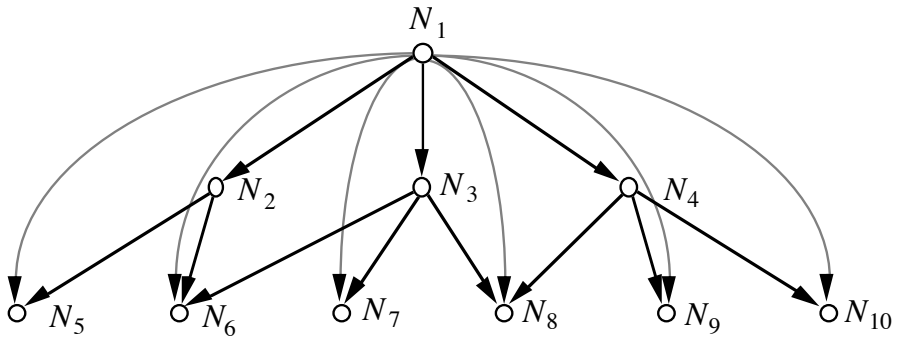

Figure 1: Direct access to nodes by ancestors

Note that in all the schemes presented in this paper all nodes are assumed to be assigned a $(k, 1)$-SIFF or a k-universal hash function family where k is assumed to be larger than the number of parents of any node in the indirect access schemes, and to be larger than the number of ancestors of any node in the direct access schemes. This uniform size of k simplifies the description of the instances of SIFF or the universal hash function family, but it may require more computer memory for their maintenance. Furthermore, this uniformity may represent an unwise usage of computer resources since the number of ancestors of a node located at a higher level in the hierarchy is small. Hence, an alternative strategy would be to use instances of a $(k_i, 1)$-SIFF or a k_i-universal hash function family for node N_i where k_i is the precise number of parents in the indirect access schemes, and the precise number of ancestors in the direct access schemes.

4 Issues in Key Management Policies

From the description of the key generating schemes for direct access to nodes it is evident that any non-parent ancestor of a node may have direct access to that node, expressed in terms of the inclusion of that ancestor's key in the key generation for the node. However, further implications arise. The key generation schemes for direct access to a node can be tailored to allow any other node in the hierarchy to access that node. In one sense this apparent freedom in providing access to nodes seems contradictory to the notion of a hierarchy with partial order on its nodes, and hence we feel some attention must be given to possible policies that should be adopted in the management and maintenance of the keys of the nodes in the hierarchy.

With the freedom attained in "linking" one node to another through the direct access key generation schemes a number of issues concerning the addition and deletion of nodes, and the granting and removal of access rights to a node must be addressed. In the following sections these issues will be considered and some points are suggested concerning the maintenance of the keys. Recall that access by a user at a node to information at another node is expressed in terms of the "links" from the user's node to the information node. Hence a *link* from node N_i to node N_j means that users at node N_i have access to information at node N_j.

4.1 Periodic Key Updates

As have been pointed out by a number of authors working in the expanding area of cryptography the problem of key management includes the issue of updates of the keys of the users. The need to have periodic key updates does not necessarily reflect the quality or security of the cryptosystem being employed. Rather, it is a precautionary measure against the failures on the part of the human users who may have accidentally or otherwise disclosed the key of a node to other unauthorized users.

The issue of periodic key updates for the hierarchy becomes important in our solution to the access control problem due to the freedom it provides in linking nodes to one another. We perceive that the establishment of new links between nodes and the removal of existing links is best done during the time of the periodic key update of every node. Similarly, the movement of users from one node (security class) to another requires that the migrating users not have unlawful access to their previous nodes. Hence the migration of users should occur also during the periodic key updates.

4.2 Temporary Key Updates

In reality often the establishment and removal of links between nodes may have to be done in the time between the periodic key updates. This requirement usually has the effect of a large number of key changes due to a change of one key of a node located higher in the hierarchy. More specifically, the tight relationship between a node and its descendants results in the need to modify the keys of all the descendants of that node if the key of that node was updated. This disadvantage occurs in all the previous solutions by other authors due to the key generation procedure for a node which involves some parameters belonging to the node's parents. Hence, it is thought that a policy must govern the changes to the keys in the hierarchy.

Such a policy should take into consideration the total number of nodes that will be affected by the temporary key update. Thus, for example, if only 10 percent of all the nodes will be affected then the temporary update can be carried out. However, if more than 50 percent of all the nodes will be affected by the temporary update due to links between nodes, then it should be postponed until the next periodic update or the next periodic update can be advanced forward in time to the present moment. The policy should also consider the life time of the temporary key update. Those that will be undone before the next periodic key update may be deferred until such time.

In the following we will discuss four broad kinds of temporary key updates that may arise in a hierarchy. It is assumed that the updates are temporary in that they will be undone before or during the next periodic key update. Alternatively they can be left to be made permanent in the next periodic update. With respect to the discussions in the following sections concerning the maintenance of keys we assume that Scheme 3 for direct access to nodes (described in Section 3.3) will be employed.

4.2.1 The Addition of Nodes

The addition of a new node to a hierarchy is a natural occurrence that is to be expected. In general there are two cases of the addition of nodes which can be considered. The first case involves the addition of a new node as a child of an existing node. This new node has no descendants of its own. The second is when the new node is to be "inserted" between a node and its parent.

The first case is simply solved by generating for it a new key using Scheme 3 described in Section 3.3. This also means that a new instance of SIFF must be assigned to the new node.

The second case requires that the new node be assigned a new key and an instance of SIFF, similar to the first case. However, in addition, links must be created between the new node and its adopted descendants. In this situation the keys of the descendant nodes need not be replaced or updated. Instead, a new instance of SIFF for each of these descendant nodes must be selected that would map to the existing key of that node all the previous input strings plus the new key from the new node.

4.2.2 The Deletion of Nodes

The deletion of a node is trivial if the node is a leaf node. However, if the deleted node is positioned internally in the hierarchy then all the descendants of the deleted node become the descendants of the parent(s) of the deleted node.

Recall that in the key generation schemes for direct access to nodes any ancestor of a node has the ability to access information in that node in one step. Thus, the deletion of a node does not require any action to be taken to the other nodes in the hierarchy unless users from the deleted node are to be migrated (or re-classified) to other nodes.

However, one important difficulty exists in the case of the deletion of nodes, namely the *ex-member problem*. Informally, the ex-member problem arises from users who have migrated to another node and who have retained a copy of the key from their previous (not necessarily deleted) node, thus having unlawful access to the descendants of that previous node. The most straightforward solution to this ex-member problem would be the selection of new keys and new instances of SIFF for all the descendants of the (previous or deleted) node and for the other nodes accessible from it through the links emanating from it. In this way the illegal copies of the key of the node is rendered useless.

Another possible solution relies on the ability to maintain the key of a node as a secret from the users in the hierarchy. The key can be made simultaneously secret and usable by employing tamper-proof devices such as smartcards. However, such an approach is beyond the scope of this paper.

4.2.3 The Addition of Links

The granting of permission to users at one node to access information at another (previously restricted) node is also a realistic requirement in a hierarchy. The granting of access to a node can be viewed as providing a link from a *source* node to a *sink* node

which may reside in different sub-posets in the hierarchy. One important point about the addition of a link from a source node N_i to a sink node N_j is the accessibility of the descendants of the sink node by the source node. By definition, if a source node has access to a given sink node, then it should also have access to the descendants of the sink node and all the other non-descendant nodes which have links from the sink node. What remains, however, is the question of how the establishment of a link from the source node to all the descendants of the sink node can be achieved economically.

One approach that is immediately clear is to simply establish links from the source node to every individual descendant node of the sink node, regardless of the required computing resource. This can be achieved by selecting uniformly and randomly a new instance of SIFF for each of these affected descendant nodes. The new instance of SIFF must take as input the existing input strings and the string corresponding to the link. The keys of each of these nodes may or may not be updated depending on the availability of computing resources. The effort in establishing all these links is rewarded in that the access to each of the sink's descendant nodes by the source node requires only a single step. However, this approach will require much computing resource and thus should only be pursued when the number of the affected nodes is small.

An alternative approach is to allow the source node N_i to access the descendants of the sink node N_j in two steps. That is, the source node must pass through the sink node before it can proceed to access any of the descendants of the sink node. Thus, only the instance of SIFF h_j associated with the sink node N_j need to be replaced in order to create a link from the source node N_i. The source node N_i must then establish the key K_j of the sink node N_j before using the key to access any of the descendants of the sink node. This approach can be used even in the case when the number of the affected nodes is large.

In the creation of links from a source node to a sink node one important criteria must be satisfied, namely that the position of the source node in the hierarchy must be higher than that of the sink node. This criteria should be observed by the trusted party which performs the selection of keys and instances of SIFF for the nodes in the hierarchy. This criteria ensures that the hierarchy remains what is was defined to be.

4.2.4 The Deletion of Links

The realization of the denial of access of a user in a source node to information at a sink node can only take the form of the selection of a new key and a new instance of SIFF for the sink node. However, the policy that is employed must determine whether the deletion of a link to a sink node necessitates the deletion of the links from the source node to all descendants of the sink node. If that is the case, then such a deletion of a link to a sink node will require the selection of new keys and new instances of SIFF for all the descendants of the sink node and for other nodes accessible from the sink node and its descendants. If such a deletion affects too many nodes, it is more advantageous to postpone it until the time of the next periodic key update of the hierarchy.

In dealing with the deletion of links there is a concept which needs to be made clear

in the key management policy, namely that of the "minimum" shape of the hierarchy. In other words, the policy must ensure that a node is not completely severed from its parent(s), which would then be equivalent to the deletion of that node. Hence the deletion of links must be guarded so as not to result in the deletion of a node, and that each node must be linked to at least one parent.

5 Conclusion

In this paper we have presented new solutions to the problem of access control in a hierarchy which are based on cryptography. The concepts in cryptography that are necessary for the solutions have been presented, and they include one-way functions, pseudo-random function families and universal hash function families. In addition, the new concept of sibling intractable function families was briefly reviewed due to its importance to the proposed solutions.

The solutions to the problem of access control in a hierarchy took the form of key generation schemes for the nodes or security classes in the hierarchy. Each node in the hierarchy is assigned a key and an instance of a universal hash function family or of a sibling intractable function family. Using the key, a user at a node (security class) can access information stored at another node (security class) lower down in the hierarchy. Access to nodes can be be *indirect*, as is the case with previous solutions to the problem, or it can be *direct*, which represents an improvement by the proposed solutions over the previous solutions. The security of the key generation schemes was examined based on a formal definition of security in the context of access control in a hierarchy.

Some issues concerning the management of the keys of the nodes were also discussed. These concentrated mainly on the addition and deletion of nodes and links between nodes in the hierarchy. The work in this paper represents a new approach in pursuit of solutions to the problem of access control in a hierarchy, and it serves also as an illustration of the importance of the new concept of the sibling intractable function family.

References

[1] Akl, S. G., and Taylor, P. D. Cryptographic solution to a multilevel security problem. In *Advances in Cryptology - Proceedings of Crypto'82* (Santa Barbara, August 1982), D. Chaum, R. L. Rivest, and A. T. Sherman, Eds., Plenum Press, NY, pp. 237–250.

[2] Akl, S. G., and Taylor, P. D. Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems 1*, 3 (1983), 239–248.

[3] Carter, J., and Wegman, M. Universal classes of hash functions. *Journal of Computer and System Sciences 18* (1979), 143–154.

[4] Chick, G. C., and Tavares, S. E. Flexible access control with master keys. In *Advances in Cryptology - Proceedings of Crypto'89,* Lecture Notes in Computer Science, Vol. 435 (1990), G. Brassard, Ed., Springer-Verlag, pp. 316–322.

[5] Goldreich, O., Goldwasser, S., and Micali, S. How to construct random functions. *Journal of ACM 33*, 4 (1986), 792–807.

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

[7] Håstad, J. Pseudo-random generation under uniform assumptions. In *Proceedings of the 22-nd ACM Symposium on Theory of Computing* (1990), pp. 395–404.

[8] Impagliazzo, R., Levin, L., and Luby, M. Pseudo-random generation from one-way functions. In *Proceedings of the 21-st ACM Symposium on Theory of Computing* (1989), pp. 12–24.

[9] MacKinnon, S. J., Taylor, P. D., Meijer, H., and Akl, S. G. An optimal algorithm for assigning cryptographic keys to access control in a hierarchy. *IEEE Transactions on Computers C-34*, 9 (1985), 797–802.

[10] Naor, M., and Yung, M. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the 21-st ACM Symposium on Theory of Computing* (1989), pp. 33–43.

[11] Ohta, K., Okamoto, T., and Koyama, K. Membership authentication for hierarchical multigroup using the extended Fiat-Shamir scheme. In *Advances in Cryptology - Proceedings of EuroCrypt'90,* Lecture Notes in Computer Science, Vol. 473 (1991), I. B. Damgård, Ed., Springer-Verlag.

[12] Rivest, R. L., Shamir, A., and Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21*, 2 (1978), 120–128.

[13] Rompel, J. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the 22-nd ACM Symposium on Theory of Computing* (1990), pp. 387–394.

[14] Sandhu, R. S. Cryptographic implementation of a tree hierarchy for access control. *Information Processing Letters 27*, 2 (1988), 95–98.

[15] Wegman, M., and Carter, J. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences 22* (1981), 265–279.

[16] Zheng, Y., Hardjono, T., and Pieprzyk, J. Sibling intractable function families and their applications. In *Advances in Cryptology - Proceedings of AsiaCrypt'91,* Lecture Notes in Computer Science (1992), Springer-Verlag. (to appear).