PART 2 SECURITY IN DISTRIBUTED SYSTEMS

This page intentionally left blank

Chapter 4

Cover-Free Families and Their Applications

San Ling and Huaxiong Wang

Division of Mathematical Sciences
School of Physical and Mathematical Sciences
Nanyang Technological University, Singapore
email: lingsan, hxwang@ntu.edu.sg

Chaoping Xing

Department of Mathematics
National University of Singapore, Singapore
email: matxcp@nus.edu.sg

Cover-free families are combinatorial objects that have been used in diverse applications such as information theory, communications, group testing, cryptography and information security. In this paper, we survey some mathematical results on cover-free families and present several interesting applications to topics in secure networks and distributed systems.

4.1. Introduction

Cover-free families were first studied in terms of superimposed binary codes by Kautz and Singleton [21] in 1964. These codes are related to retrieval files, data communication and magnetic memories. In 1985 Erdös, Frankl and Füredi [14] studied cover-free families as combinatorial objects, generalising the Sperner systems. Since then, they have been discussed by numerous researchers in the context of information theory, combinatorics, communication and cryptography and information security. In this paper we will present several interesting applications to topics in secure networks and distributed systems.

Definition 4.1. Let X be a set of N elements (points) and let $\mathcal{B}$ be a

set of T subsets (blocks) of X. Then $(X, \mathcal{B})$ is called an (s,t)-*cover-free family* provided that, for any s blocks $B_1, \ldots, B_s$ in $\mathcal{B}$ and t other blocks $B'_1, \ldots, B'_t$ in $\mathcal{B}$, one has

$$\bigcap_{i=1}^{s} B_i \not\subseteq \bigcup_{j=1}^{t} B'_j.$$

In other words, no intersection of s blocks is contained in the union of t other blocks. Sometimes, we will use notation (s,t)-$CFF(N,T)$ to denote an (s,t)-cover-free family $(X, \mathcal{B})$ in which $|X| = N$ and $|\mathcal{B}| = T$. We call $(X, \mathcal{B})$ k-uniform if $|B| = k$ for all $B \in \mathcal{B}$.

Cover-free families have been studied under different names, such as superimposed codes, key distribution patterns, non-adaptive group testing algorithms, etc. For instance, a $(1,t)$-cover-free family is exactly the t-cover-free family studied by Erdös *et al* [14], and a $(2,t)$-cover-free family was introduced, under the name of *key distribution pattern*, by Mitchell and Piper [34] to provide a mechanism for distributing a secret key to each pair of users in a network. For general $s \geq 2$ and $t \geq 2$, (s,t)-cover-free families are relevant to conference key distribution and broadcast encryption [16, 44]. In the following, we show two equivalent objects of cover-free families: *coverings of order-interval hypergraph* [13, 49] and *disjunct systems* [43, 49].

Let l, u, n be integers such that $0 < l < u < n$. Let $[n] = \{1, 2, \ldots, n\}$. Define $P_{n;l,u} = \{X \subseteq [n] : l \leq |X| \leq u\}$, where $0 < l < u < n$. Define a hypergraph $G_{n;l,u} = (P, E)$ as follows. Let the set of points be $P = P_{n;l,u}$, and let the set of edges E be the maximal intervals, *i.e.*,

$$E = \{I = \{C \subseteq [n] : Y_1 \subseteq C \subseteq Y_2\} : |Y_1| = l, |Y_2| = u, y_1, Y_2 \subseteq [n]\}.$$

Definition 4.2. A covering of a hypergraph is a subset of points S such that each edge of the hypergraph contains at least one point of S.

Let $(X, \mathcal{B})$ be a set system, where $X = \{x_1, x_2, \ldots, x_v\}$ and $\mathcal{B} = \{B_1, B_2, \ldots, B_b\}$. The *incidence matrix* of $(X, \mathcal{B})$ is the $b \times v$ matrix $A = (a_{ij})$, where

$$a_{ij} = \begin{cases} 1 & \text{if } x_j \in B_i \\ 0 & \text{if } x_j \notin B_i. \end{cases}$$

Conversely, given an incidence matrix, we can define an associated set system in an obvious way.

Definition 4.3. A set system $(X, \mathcal{B})$ is an (i,j)-*disjunct system* provided that, for any $P, Q \subseteq X$ such that $|P| \leq i$, $|Q| \leq j$ and $P \cap Q = \emptyset$, there

exists a $B \in \mathcal{B}$ such that $P \subseteq B$ and $Q \cap B = \emptyset$. An (i,j)-disjunct system is denoted as an (i,j)-DS(v,b) if $|X| = v$ and $|\mathcal{B}| = b$.

Theorem 4.1. *The following statements are equivalent.*

(i) There exists a covering of $G_{T;i,T-j}$ of size N.
(ii) There exists an (i,j)-DS(T,N).
(iii) There exists an (i,j)-CFF(N,T).

Proof. Firstly, we show that (i) is equivalent to (ii). Observe that S is a covering of $G_{T;i,T-j}$ if and only if for any $Y_1, Y_2 \subseteq [T], Y_1 \subset Y_2, |Y_1| = i, |Y_2| = T - j$, there is a $C \in S$ such that $Y_1 \subseteq C \subseteq Y_2$. This is equivalent to that for any $Y_1, Y_3 \subseteq [T], |Y_1| = i, |Y_3| = T - (T - j) = j, Y_1 \cap Y_3 = \emptyset$, there is some $C \in S$ such that $Y_1 \subseteq C$ and $Y_3 \cap C = \emptyset$, which is equivalent to that $([T], S)$ is an (i,j)-DS(T,N).

Secondly, for the equivalence of (ii) and (iii), it is easy to see that A is an incidence matrix of a disjunct system if and only if A^T, the transpose of A, is an incidence matrix of a cover-free family. □

4.2. Bounds

We start with a trivial construction for (s,t)-cover-free families. For any integers $T \geq s > 0$, define $X = \{x_A : A \subseteq [T], |A| = s\}$. For $1 \leq i \leq T$, define $B_i = \{x_A : i \in A \text{ and } x_A \in X\}$, and $\mathcal{B} = \{B_i : i \in [T]\}$. Then it is easy to see that $(X, \mathcal{B})$ is an (s,t)-CFF$(\binom{T}{s}, T)$ for any $t \leq T - s$. We are interested in (s,t)-CFF(N,T) with better performance than this trivial construction. That is, (s,t)-CFF(N,T) with $N < \binom{T}{s}$. Note that given the values of s and t there is a trade-off between N and T in an (s,t)-CFF(N,T). More precisely, we are interested in (s,t)-CFF(N,T) for which T is as large as possible while s, t and N are given; or equivalently, the value N is small as possible while s, t and T are fixed.

Let $N((s,t),T)$ denote the minimum value of N in an (s,t)-CFF(N,T). It is desirable to find the value of $N((s,t),T)$. Unfortunately, as shown in [4, 49], computing the value of $N((s,t),T)$ turns out to be rather hard.

Theorem 4.2. *Given integers s, t, T and k, the problem of deciding $N((s,t),T) \leq k$ is* **NP***-complete.*

Proof. For given Let $G_{n;l,u}$ be a hypergraph defined in Section 4.1, and let

$$\tau(G_{n;l,u}) = \min\{|S| : S \text{ is a covering of } G_{n;l,u}\}.$$

It has been proved in [4] that for any given integers n, l, u and k, the problem of deciding $\tau(G_{n;l,u}) \leq k$ is **NP**-complete. The result follows from Theorem 4.1 immediately. □

Theorem 4.3. *We have the following known bounds on cover-free families:*

(a) (Erdös et al [14]) In a k-uniform $(1,t)$-CFF(N,T)

$$T \leq \binom{N}{\lceil \frac{k}{t} \rceil} \Big/ \binom{k-1}{\lceil \frac{k}{t} \rceil - 1}.$$

(b) (Dýachkov and Rykov [11], Füredi [18] and Ruszinkó [38]) For any $t \geq 2$, it holds that for any $(1,t)$-CFF(N,T)

$$N \geq c \frac{t^2}{\log t} \log T,$$

where the constant c is shown to be approximately $1/2$ in Dýachkov and Rykov [11], approximately $1/4$ in Füredi [18], approximately $1/8$ in Ruszinkó [38].

(c) (Dyer et al [12])

$$N((s,t),T) \geq t(s \log T - \log t - s \log s).$$

(d) (Engel [13])

$$N((s,t),T) \geq \binom{s+t-1}{s} \log(T - t - s + 2).$$

(e) (Engel [13]) For any $\epsilon > 0$, it holds that

$$N((s,t),T) \geq (1-\epsilon) \frac{(s+t-2)^{s+t-2}}{(s-1)^{s-1}(t-1)^{t-1}} \log(T - t - s + 2)$$

for all sufficiently large T.

(f) (Stinson et al [47]) For $s, t \geq 1$ and $T \geq s + t > 2$, we have

$$N((s,t),T) \geq 2c \frac{\binom{s+t}{t}}{\log(s+t)} \log T,$$

where the constant c is the same as in (b).

(g) (Stinson et al [47], Ma and Wei [29]) For any integers $s, t \geq 1$ and $T \geq \max\{\lfloor (s+t+1)/2 \rfloor^2, 5\}$,

$$N((s,t),T) \geq 0.7c \frac{\binom{s+t}{s}(s+t)}{\log \binom{s+t}{s}} \log T,$$

where the constant c is the same as in (b).

(h) (Stinson and Wei [46]) For positive integers s, t, T,

$$N((s,t),T) \leq \min\left\{\left\lceil \frac{(s+t)\log T}{-\log p}\right\rceil, \left\lceil \frac{(s+t-1)\log 2T}{-\log p}\right\rceil\right\}$$

where $p = 1 - \frac{s^s t^t}{(s+t)^{s+t}}$.

Since it is hard to compute the exact value of $N((s,t),T)$ for larger values of T, some authors have considered another measurement on the efficiency on CFFs, called the (performance) *rate*, defined as

$$R(X,\mathcal{B}) = \frac{\log_2 T}{N},$$

for a cover-free-family $(X,\mathcal{B})$. We are interested in the asymptotic behavior of the rate.

Definition 4.4. For fixed s and t, we define the *asymptotic rate* of (s,t)-*CFFs* as

$$R(s,t) = \lim_{T\to\infty} \frac{\log_2 T}{N((s,t),T)}.$$

The following theorem was proved in [26].

Theorem 4.4. *For any integers s and t, we have*

$$R(s,t) \leq \min_{0<x<s}\min_{0<y<t} \frac{R(s-x,t-y)}{R(s-x,t-y)+(x+y)^{x+y}/(x^x y^y)}.$$

Table 4.1 (taken from [23]) lists some numerical values of the upper bounds for the asymptotic rate $R(s,t)$ which are the best results among several choices in Theorem 4.4.

Table 4.1. **Numerical Values of Upper bounds for the Rate** $R(s,t)$

(s,t)	(2, 3)	(2, 4)	(2, 5)	(2, 6)	(3, 4)
$R(s,t) \leq$	0.07488	0.045522	0.028677	0.020385	0.018282
(s,t)	(3, 5)	(3, 6)	(4, 4)	(4, 5)	(4, 6)
$R(s,t) \leq$	0.010915	0.0066989	0.0095784	0.0045496	0.0025677
(s,t)	(5, 5)	(5, 6)	(6, 6)		
$R(s,t) \leq$	0.0023889	0.0011361	0.0005969		

Definition 4.5. An (s,t)-*CFF*(N,T) is said to be *optimal* if $N = N((s,t),T)$.

Although computing the value of $N((s,t),T)$ is hard, some cover-free families with small parameters are known to be optimal. There are cases that the trivial solution given at the beginning of this section results in the optimal solutions. For example, from [13] and [23], we know that whenever $T \leq s+t+t/s$ or $T \leq \frac{(t+1)s}{s-1} - \sqrt{\frac{36t}{s-1}}$, then $N((s,t),T) = \binom{T}{s}$, which implies that the trivial solution is an optimal solution. We list some of these optimal families in Table 4.2 (taken from [22, 23]).

Table 4.2. **Optimal** (s,t)**-CFF**(N,T)**.**

$T=$	5	6	7	8	9	10	11-12	16 -20
$N((1,2),T)=$	5	6	7	8	9	9	9	
$N((1,3),T)=$	5	6	7	8	9	10	11-12	16
$N((2,2),T)=$	10	14	14	14	18	18-20	20-22	22-26
$N((2,3),T)=$	10	15	21	24-28	26-30	30	33-45	45-48

4.3. Constructions

4.3.1. *Constructions from error-correcting codes*

A nice construction for cover-free families is to use error-correcting codes ([14, 43]). Let Y be an alphabet of q elements. An (n,T,d,q) *code* is a set $\mathcal{C}$ of T vectors in Y^n such that the Hamming distance between any two distinct vectors in $\mathcal{C}$ is at least d.

Consider an (n,T,d,q) code $\mathcal{C}$. We write each codeword as $c_i = (c_{i1},\ldots,c_{in})$ with $c_{ij} \in Y$, where $1 \leq i \leq T, 1 \leq j \leq n$. Set $X = [n] \times Y$ and $\mathcal{B} = \{B_i : 1 \leq i \leq T\}$, where for each $1 \leq i \leq T$ we define $B_i = \{(j, c_{ij}) : 1 \leq j \leq n\}$. It is easy to see that $|X| = nq$, $|\mathcal{B}| = T$ and $|B_i| = n$. For each choice of $i \neq k$, we have $|B_i \cap B_k| = |\{(j,c_{ij}) : 1 \leq j \leq n\} \cap \{(j,c_{kj}) : 1 \leq j \leq n\}| = |\{j : c_{ij} = c_{kj}\}| \leq n-d$.

It is straightforward to show that $(X,\mathcal{B})$ is a $(1,t)$-CFF(nq,T) if the condition $t < \frac{n}{n-d}$ holds. We thus obtain the following theorem.

Theorem 4.5. *If there is an* (n,T,d,q) *code, then there exists a* $(1,t)$-CFF(nq,T) *provided that* $t < \frac{n}{n-d}$.

Now if we apply the above coding construction to algebraic-geometry codes, we immediately obtain the following corollary.

Corollary 4.1 ([37]). *For any integers* g,l,n *with* $l \leq g \leq l < n$, *there exists a* $(1, \lfloor (n-1)/l \rfloor)$-CFF$(ng, g^{l-g+1})$.

Let us look at the asymptotic behavior of cover-free families in Corollary 4.1. From [36], we know that for a fixed $0 \leq \delta < 1$ and a square prime power q, there exists a sequence of (n_i, T_i, d_i, q)-codes such that $n_i \to \infty$ as $i \to \infty$ and

$$\lim_{i\to\infty} \frac{d_i}{n_i} = \delta, \quad \lim_{i\to\infty} \frac{\log_q T_i}{n_i} \geq 1 - \delta - \frac{1}{\sqrt{q}-1}.$$

By Theorem 4.5, we have the following asymptotic result.

Corollary 4.2. *For a fixed $t \geq 1$ and a square prime power q with $t < \sqrt{q} - 1$, there exists a sequence of $(1,t)$-CFF$(n_i q, q^{l_i - g + 1})$ such that*

$$\lim_{i\to\infty} \frac{\log q^{l_i - g+1}}{n_i q} = \frac{\log q}{q} \cdot (\frac{1}{t} - \frac{1}{\sqrt{q}-1}). \tag{4.1}$$

Corollary 4.2 give examples of infinite cover-free families with positive rates and they can be constructed explicitly. In other words, for any fixed t there are infinite families of $(1,t)$-CFF(N,T) in which $N = O(\log T)$ with explicit constructions.

Next we describe the concatenated construction given in [23, 47], which is a powerful method in constructing a larger cover-free family from small cover-free families.

Definition 4.6. A matrix $C = (c_{uv})_{N\times T}$ with entries from $[q]$ is called an (s,t) *separating matrix* of size (N,T,q), if, for any pair of sets $I, J \subset [T]$ such that $|I| = s, |J| = t$ and $I \cap J = \emptyset$, there exists an integer $x \in [N]$ such that the sets $\{c_{xi}, i \in I\}$ and $\{c_{xj}, j \in J\}$ are disjoint.

The notion of the separating matrix is equivalent to those of the separate code [22] and the separating hash family [47]. Let C be a (s,t) separating matrix of size (N_0, T_0, q) and $A = (a_{ij})_{N_1\times q}$ be the transpose of the incidence matrix of an (s,t)-DS(q, N_1). Denote by $b_1, b_2, \ldots, b_q$ the columns of A. We construct an $N_0N_1 \times T_0$ matrix $B = C \diamond A$ by substituting the element i in C by b_i. It can be verified that the resulting matrix B is the transpose of the incidence matrix of an (s,t)-CFF(N_0N_1, T_0) (see [45]).

Theorem 4.6. *If there exist an (s,t) separating matrix of size (N,T,q) and an (s,t)-CFF(N_0,q), then there exists an (s,t)-CFF(NN_0,T).*

Separate matrices can be constructed from error-correcting codes, therefore making another link between cover-free families and error-correcting codes.

Theorem 4.7 ([47]). *If there exists an (N, T, d, q) code, then there exists an (s, t) separating matrix of size (N, T, q) provided that*

$$\frac{d}{N} > 1 - \frac{1}{st}.$$

Proof. Let C be an $N \times T$ matrix such that each column is a codeword in an (N, T, d, q) code. Since the minimum distance of the code is d, we know any two codewords have at most $N - d$ elements in common. It is easy to see that the matrix C is an (s, t)-separate matrix if $N > st(N - d)$, proving the desired result. □

4.3.2. *Constructions from perfect hash families*

Let n and m be integers such that $2 \leq m \leq n$. Let A be a set of size n and let B be a set of size m. A *hash function* is a function h from A to B. We say a hash function $h : A \to B$ is *perfect* on a subset $X \subseteq A$ if h is injective when restricted to X. Let w be an integer such that $2 \leq w \leq m$ and let $\mathcal{H} \subseteq \{h : A \to B\}$.

Definition 4.7. We say $\mathcal{H}$ is an (n, m, w)-*perfect hash family* if for any $X \subseteq A$ with $|X| = w$ there exists at least one function $h \in \mathcal{H}$ such that h is perfect on X. We use $PHF(N; n, m, w)$ to denote an (n, m, w)-perfect hash family with $|\mathcal{H}| = N$.

The terminology of "perfect hash family" is motivated by the fact that we have a family of hash functions with the property that if at most w elements are to be hashed, then at least one function in the family yields no collisions when applied to the given w inputs. Obviously, one can take all functions from A to B, which yield a perfect hash family with $|\mathcal{H}| = m^n$. What makes perfect hash families interesting is that by careful design the number of hash functions can achieve $O(\log n)$, instead of $O(2^n)$ from the above trivial solution.

There have been different definitions and representations of perfect hash families in the literature. For example, a $PHF(N; n, m, w)$ can be depicted as an $N \times n$ array of m symbols, where each row of the array corresponds to one of the functions in the family. This array has the property that, for any subset of w columns, there exists at least one row such that the entries in the w given columns of that row are distinct. A $PHF(N; n, m, w)$ can also be treated as a family of N partitions of an n-set A such that each partition π has at most m parts and such that for all $X \subseteq A$ with $|X| = w$,

there exists a partition π for which the elements in X are in distinct parts of π.

Perfect hash families originally arose as part of compiler design; see Mehlhorn [32] for a summary of the early results in this area. They have applications to operating systems, language translation systems, hypertext, hypermedia, file managers and information retrieval systems; see the survey article of Czech, Havas and Majewski [7]. More recently, they have found numerous applications to cryptography [35, 45].

Let $N(n, m, w)$ denote the minimum N for which a $PHF(N; n, m, w)$ exists.

Theorem 4.8 ([32]). *For any integers $n \geq m \geq w \geq 2$, we have*

(i) $N(n, m, w) \geq \frac{\log n}{\log m}$.
(ii) $N(n, m, w) \leq \lceil we^{w^2/m} \log n \rceil$.

It follows from Theorem 4.8 that for fixed m and w, $N(n, m, w) = O(\log n)$.

Next, we give two constructions of cover-free families from perfect hash families. The first construction is a direct construction from perfect hash families and works only for $(1, t)$-CFF. Assume that $\mathcal{H}$ is a $PHF(N; T, m, t + 1)$ from A to B. Let $A = \{1, 2, \ldots, T\}$ and $B = \{1, 2, \ldots, m\}$. We define

$$X = \mathcal{H} \times B = \{(h, j) \ : \ h \in \mathcal{H}, j \in B\}.$$

For each $1 \leq i \leq T$, we define a subset (block) B_i of X by

$$B_i = \{(h, h(i)) : \ h \in \mathcal{H}\},$$

and $\mathcal{B} = \{B_i : \ 1 \leq i \leq T\}$. Then $(X, \mathcal{B})$ is a $(1, t)$-CFF(Nm, T). Indeed, $|X| = Nm$ and $|\mathcal{B}| = T$. For any $t + 1$ blocks $B_{i_1}, \ldots, B_{i_t}, B_j$, since $\mathcal{H}$ is a $PHF(N; T, m, t + 1)$, there exists a hash function $h \in \mathcal{H}$ such that h restricted to $\{i_1, \ldots, i_t, j\}$ is one-to-one. It follows that $h(i_1), \ldots, h(i_t), h(j)$ are $t + 1$ distinct elements in B, which also implies that $(h, h(i_1)), \ldots, (h, h(i_t)), (h, h(j))$ are $t + 1$ distinct elements in $B_{i_1}, \ldots, B_{i_t}, B_j$, respectively. Hence the union of any t blocks in $\mathcal{B}$ cannot cover any remaining block. Thus, we have shown the following result.

Theorem 4.9. *If there exists a $PHF(N; T, m, t + 1)$, then there exists a* $(1, t)$-CFF(Nm, T).

The second construction from perfect hash families ([44]) provides a method of building a larger cover-free family from small cover-free families. It has a similar flavor as the coding construction in subsection 4.3.1.

The construction works as follows. Let $(X_0, \mathcal{B}_0)$ be an (s,t)-CFF(N_0, T_0) and let $\mathcal{H} = \{h_1, \ldots, h_N\}$ be a $PHF(N; T, T_0, s+t)$. Consider N copies of $(X_0, \mathcal{B}_0)$, denoted by $(X_1, \mathcal{B}_1), \ldots, (X_N, \mathcal{B}_N)$, where X_i and X_j are disjoint sets, i.e. $X_i \cap X_j = \emptyset$, for all $i \neq j$. For each $1 \leq j \leq N$, denote $X_j = \{x_1^{(j)}, \ldots, x_{N_0}^{(j)}\}$ and $\mathcal{B}_j = \{B_1^{(j)}, \ldots, B_{T_0}^{(j)}\}$. Then $(X_j, \mathcal{B}_j)$ is an (s,t)-CFF(N_0, T_0). We construct a pair $(X, \mathcal{B})$ with

$$X = X_1 \cup \cdots \cup X_N \quad \text{and} \quad \mathcal{B} = \{B_1, \ldots, B_n\},$$

where $B_i = B_{h_1(i)}^{(1)} \cup \cdots \cup B_{h_N(i)}^{(N)} = \cup_{j=1}^{N} B_{h_j(i)}^{(j)}$ for $1 \leq i \leq T$. That is, an element of $\mathcal{B}$ is a union of elements of $\mathcal{B}_j$, $1 \leq j \leq N$, chosen through the application of the perfect hash family. We show that $(X, \mathcal{B})$ is an (s,t)-CFF(T, NN_0). Clearly, $|X| = NN_0$ and $|\mathcal{B}| = T$. For any $s+t$ blocks $B_{i_1}, \ldots, B_{i_s}, B_{j_1}, \ldots, B_{j_t}$, there exists at least one hash function $h_k \in \mathcal{H}$ which is one-to-one on $\{i_1, \cdots, i_s, j_1, \ldots, j_t\}$. Since $(X_k, \mathcal{B}_k)$ is an (s,t)-CFF(N_0, T_0), we have

$$\begin{aligned}\left|\bigcap_{u=1}^{s} B_{i_u} \setminus \bigcup_{v=1}^{t} B_{j_u}\right| &\geq \left|\bigcap_{u=1}^{s} B_{h_k(i_u)}^{(k)} \setminus \bigcup_{v=1}^{t} B_{h_k(j_u)}^{(k)}\right| \\ &\geq 1,\end{aligned}$$

proving the desired result. Thus, we have the following result.

Theorem 4.10. *Suppose that there exist an* (s,t)-CFF(N_0, T_0) *and a* $PHF(N; T, T_0, s+t)$. *Then there exists an* (s,t)-CFF(NT_0, T).

4.3.3. *Constructions from designs*

Let Y be a set of v elements (called *points*), and let $\mathcal{A} = \{A_1, A_2, \ldots, A_\beta\}$ be a family of k-subsets of Y (called *blocks*). We say that $(Y, \mathcal{A})$ is a $t-(v,k,\lambda)$ *design* if every subset of t points occurs in exactly λ blocks. It can be shown by elementary counting that a $t-(v,k,\lambda)$ design is also a $t'-(v,k,\lambda')$ design for $1 \leq t' \leq t$, where

$$\lambda' = \frac{\lambda \binom{v-t'}{t-t'}}{\binom{k-t'}{t-t'}}.$$

Theorem 4.11 ([44]). *If there exist an* $(s+1)-(n,k,\lambda)$ *design, then there exists an* (s,t)-CFF$(\lambda\binom{n}{s}/\binom{k}{s},n)$ *provided*

$$t \leq \frac{n-s}{k-s}.$$

Proof. Let $(Y,\mathcal{A})$ be an $(s+1)-(n,k,\lambda)$ design, where $Y = \{y_1,y_2,\ldots,y_n\}$ and $\mathcal{A} = \{A_1,A_2,\ldots,A_\beta\}$. We consider the dual of $(Y,\mathcal{A})$, $(X,\mathcal{B})$, defined by $X = \{A_1,A_2,\ldots,A_\beta\}$ and $B_i = \{A_r \mid A_r \in X, y_i \in A_r\}$. We show that $(X,\mathcal{B})$ is an (s,t)-CFF.

For each s-subset Δ of Y, there are exactly $\lambda(n-s)/(k-s)$ elements (blocks) from $\mathcal{A}$ that contain Δ. For any given t-subset $\Lambda \subseteq Y$ and $\Delta \cap \Lambda = \emptyset$, and for each $y \in \Lambda$, there are λ blocks that contain $\Delta \cup \{y\}$. Thus, the number of blocks from $\mathcal{A}$ that contain Δ and at least one member from Λ is at most λt. Since $\lambda t < \lambda(v-s)/(k-s)$, it follows that there exists a block from $\mathcal{A}$ that contains Δ such that $\mathcal{A} \cap \Lambda = \emptyset$. It is then easy to verify that $(X,\mathcal{B})$ is indeed an (s,t)-CFF$(\lambda\binom{n}{s}/\binom{k}{s},n)$ □

Corollary 4.3. *An* $(s+1)-(n,k,1)$ *design gives rise to an* (s,t)*-em CFF*(N,T)*, where*

$$N = \frac{\binom{n}{s+1}}{\binom{k}{s+1}} = \frac{(n-s)\binom{n}{s}}{(k-s)\binom{k}{s}},\ T = n, \quad \textit{and} \quad t < \frac{n-s}{k-s}.$$

From [34, 44], we know that an inversive plane is a $3-(q^2+1,q+1,1)$ design. Such a design is known to exist whenever q is a prime power. Applying Corollary 4.3 we know there exists a $(2,q)$-CFF$(q(q^2+1),q^2+1)$. Taking $q=3$, we obtain a $(2,3)$-CFF$(30,10)$, which is optimal since $N((2,3),10) = 30$ ([22]). Note that the codewords of weight 4 in the binary extended Hamming $[8,4,4]$ code form a $3-(8,4,1)$ design. It follows that there is a $(2,2)$-CFF$(8,14)$, which is optimal as well [22].

The concept of super-simple t-design was introduced by Gronau and Mullin [20]. The construction of cover-free families from super-simple designs is proposed by Kim and Lebedev [22].

Definition 4.8. A *super-simple* $t-(v,k,\lambda)$ design is a $t-(v,k,\lambda)$ design with $\lambda > 1$ in which the intersection of any two blocks has at most t elements.

Theorem 4.12 ([22]). *A super-simple* $s-(n,k,\lambda)$ *design gives rise to an* $(s,\lambda-1)$-CFF$(\lambda\binom{n}{s}/\binom{k}{s},n)$.

Proof. Let $(Y, \mathcal{A})$ be a super-simple $s-(n,k,\lambda)$ design, where $Y = \{y_1, y_2, \ldots, y_n\}$ and $\mathcal{A} = \{A_1, A_2, \ldots, A_\beta\}$. As in the proof of Theorem 4.11, let $(X, \mathcal{B})$ be the dual of $(Y, \mathcal{A})$, where $X = \{A_1, A_2, \ldots, A_\beta\}$ and $B_i = \{A_r \mid A_r \in X, y_i \in A_r\}$. We show that $(X, \mathcal{B})$ is an $(s, \lambda-1)$-CFF$(\lambda\binom{n}{s}/\binom{k}{s}, n)$.

For each point y, we denote by $S_y \subseteq \mathcal{A}$ the collection of blocks that contain y. For any s points $y_{i_1}, \ldots, y_{i_s} \in Y$, there are exactly λ blocks from $\mathcal{A}$ that contain these t points. That is,

$$|B_{i_1} \cap B_{i_2} \cap \cdots \cap B_{i_s}| = \lambda.$$

Consider any other t points $y_{j_1}, \ldots, y_{j_t}$, where $t = \lambda - 1$. Since no two (or more) blocks of a super-simple s design can have more than s common points, for any ℓ with $1 \le \ell \le t$, we have

$$|B_{i_1} \cap B_{i_2} \cap \cdots \cap B_{i_s} \cap B_{j_\ell}| \le 1.$$

It follows that

$$\left|B_{i_1} \cap B_{i_2} \cap \cdots \cap B_{i_s} \cap \left(\cup_{\ell=1}^{t} B_{j_\ell}\right)\right| \le t < \lambda.$$

We then have

$$B_{i_1} \cap B_{i_2} \cap \cdots \cap B_{i_s} \not\subseteq \cup_{\ell=1}^{t} B_{j_\ell},$$

for otherwise, we would have $B_{i_1} \cap B_{i_2} \cap \cdots \cap B_{i_s} \subseteq \cup_{\ell=1}^{t} B_{j_\ell}$ which implies that

$$\left|B_{i_1} \cap B_{i_2} \cap \cdots \cap B_{i_s} \cap \left(\cup_{\ell=1}^{t} B_{j_\ell}\right)\right| = |B_{i_1} \cap B_{i_2} \cap \cdots \cap B_{i_s}| = \lambda,$$

a contradiction. This shows that $(X, \mathcal{B})$ is an cover-free family with the desired parameters. □

Note that it is easy to see that an $(s+1)-(n,k,1)$ design is a super-simple $s-(n,k,(n-s)/(k-s))$ design. Therefore, in this case Theorem 4.12 implies in Theorem 4.11.

4.4. Applications

In this section, we present several interesting applications of cover-free families to topics in secure distributed systems.

4.4.1. *Key distribution in networks*

Key management for secure communication in the general network model has been widely studied in recent years. There are typically three approaches to the general key management problem: *public-key infrastructure* (PKI); *trusted-server* and *key predistribution*. The *PKI* schemes depend on asymmetric cryptographic primitives, such as RSA encryption or authenticated Diffie-Hellman key agreement; such schemes suffer expensive computational cost and storage constraints in each node of the network. The *trusted-server* schemes, such as Kerberos, rely on a trusted server to generate keys between nodes; they require a trusted infrastructure, which may not exist or be maintained in many modern network environments, such as Ad-Hoc networks. The *key predistribution* scheme distributes information about the keys among all nodes prior to deployment. Over the past few years, key predistribution schemes have attracted much attention because of their suitability in many current-day network infrastructures such as Ad Hoc networks, wireless networks, etc.

The key predistribution problem was first considered by Blom [2], Mitchell and Piper [34], and Gong and Wheeler [19]. Assume that a *key distribution center* (KDC) enables a secure communication between any pair of nodes by issuing a unique cryptographic key to each pair of nodes. If there are T nodes in a network, then there are $\binom{T}{2}$ possible pairs, and so about $\frac{1}{2}T^2$ keys need to be generated by the KDC and all of them need to be distributed to the nodes secretly, which may be impractical when T is large. To address this problem, Mitchell and Piper [34] proposed a solution in which the KDC generates a set X of N keys and issues each node P_i $(i = 1, 2, \ldots, T)$ a subset B_i of these keys, so $B_i \subset X$. If nodes P_i and P_j wish to securely communicate with each other, they can use a key constructed from the set of keys contained in $B_i \cap B_j$ (normally, this would be the XOR of all keys in this intersection). By imposing the condition $B_i \cap B_j \not\subseteq B_r$, for all $r \notin \{i, j\}$, a secure communication between P_i and P_j against each node P_r is guaranteed. This approach can be further extended to networks that are resilient against collusion attacks. If $B_i \cap B_j$ is not contained in the union of any other t-tuple of subsets, then secure communication between P_i and P_j is possible even if an adversary can corrupt t nodes in the network. Thus, a $(2, t)$-cover-free family can be used for key distribution with a significant reduction on keys.

One of the major drawbacks with the key distribution pattern schemes is that *efficient* constructions are typically probabilistic and result in small t,

which means that the maximum number of nodes that can be compromised by an adversary cannot be too large. This makes them impractical for many network applications such as Ad Hoc networks and distributed senor networks (where robustness is the main concern).

There are several extensions on the cover-free family approach. Eschenauer and Gligor propose a random key predistribution in [15]. In their scheme, each node receives a random subset of keys from a large key pool. To agree on a key for communication, two nodes find one common key within their subsets and use that key as their shared key. The original Eschenauer-Gligor scheme does not consider an attack by a collusion of compromised nodes in the network. Since then, many extensions of the Eschenauer-Gligor scheme have been proposed. In [6], Chan, Perrig, and Song propose a scheme that has l common keys for any two nodes, instead of a single key. It can then be shown that, by increasing the value of l, the network resilience against collusion attacks is improved.

In [27], Lee and Stinson give two deterministic key predistribution schemes using strongly regular graphs as network graphs. One scheme applies public, one-way functions to reduce the required key storage and another scheme combines Blom's scheme with strongly regular graphs, yielding a tradeoff between the connectivity of the network and the resilience. Similar approaches have been developed by Du, Deng, Han, and Varsheney [10], by Liu and Ning [28], and by other authors.

4.4.2. *Antijamming systems*

Traditional antijamming systems [48] use spread spectrum techniques to increase availability. In these systems a transmitter wants to broadcast a signal to a single receiver such that the enemy cannot jam the transmission. In the classical communication scenario, a message modulates a carrier frequency f which is known to the transmitter and receiver and so the receiver can receive the message. However if f is publicly known, an outsider can send a strong noise signal on the same frequency and hence completely jam the reception. To protect against jamming, the transmitter and the receiver can keep their shared frequency secret and use new frequencies after every v seconds, where v is the minimum time required for the enemy to find f. Pseudorandom generators [33] are often used to decide the new frequency. This is the so-called frequency-hopping spread spectrum system.

Spread-spectrum systems have been used for Wireless LAN, or WLAN [5, 41]. A WLAN is a flexible data communication system that provides an

attractive alternative to wired LAN within a building or where wires cannot go. A PC with a wireless adapter can connect to a wired LAN equipped with a transmitter/receiver device, called an *access point*, or can have a peer-to-peer connection with a set of PCs with wireless adapters. Traditional spread spectrum systems are for providing security and reliability between the two ends of a single communication channel. Using spread spectrum in group communication requires a careful adaptation of the traditional model.

If a group member wants to broadcast a message to the rest of the group, one possible solution is to give the transmitter's frequency list and frequency update table, to all the receivers. This would allow the receivers to synchronize their receiving equipment and follow the transmitter's frequency 'hopping'. However the system would be completely vulnerable to jamming by a receiver simply because receivers know the secret frequencies and can use this knowledge to jam the transmitter. That is, when more than one receiver is considered, the attack is not limited to the outsiders who do not know the frequencies but also could be launched by insiders with some privileged information. In other words, using the above simplistic approach means that the system only works if the receivers are assumed trusted. This is not a reasonable assumption in an open environment.

In [9], an antijamming system was suggested. In this model, the transmitter sends the same signal on a number of frequencies such that each receiver only knows a subset of these frequencies. Suppose that each receiver P_i is given a set of secret frequencies. When the number of different frequencies the transmitter wants to use is less than the number of receivers, some frequencies will have to be shared. Therefore, some frequencies will be assigned to at least two receivers.

A transmitter A will secretly choose N frequencies out of a total of M frequencies and will send the message simultaneously over these N frequencies. Knowing the frequency of a channel, a receiver may use it either to receive the messages sent by A, or else to send noise on that frequency with the purpose of jamming the reception of other receivers who are listening to the same channel.

A uses a public channel allocation table. For simplicity we number the channels from 1 to N. Each receiver P_i is assigned a collection of channels or, in other words, a subset $B_i \subseteq \{1, \ldots, N\}$. This allocation is public and is displayed in the table. Any receiver is also given secret information which specifies the correspondence between the allocated channels and the actual frequency, i.e., if a receiver is assigned the channels $i_1, \ldots, i_k$, it will receive

the associated frequencies $f_{i_1}, \ldots, f_{i_k}$.

We assume that there are T receivers, $P_1, \ldots, P_T$, and a group of up to t receivers might collude against a receiver P_j. In this case they can send noise on all their allocated channels (frequencies). If P_j's allocated frequencies are all among the frequencies of the colluders, then it cannot receive any message and its reception will be jammed. We assume the channels are authenticated, that is the transmitter can be uniquely determined. This means that if a receiver is left with even one un-jammed channel, it is able to receive messages sent by the transmitter. It is easy to see that an $(1, t)$-CFF gives rise to the channel allocation for an antijamming system against up to t colluders.

4.4.3. *Secure multicast*

Multicast, or *one-to-many* communication is the basic form of transmission in group communication applications and forms the main primitive for a range of advanced telecommunication services including video broadcasting, multi-party teleconferencing, stock quote distribution, and updating software, etc. Multicast security has been intensively studied in recent years (see, for example [16, 24, 44]. Secure communication in multicast environment is much more challenging than traditional point-to-point communication and raises numerous new security problems. Examples are controlling access to the encrypted data, and efficient management of dynamic groups where new members join or existing members need to be evicted.

A simple solution to providing secure communication in a group is by employing conventional point-to-point cryptographic protocols. For example, secure group communication can be achieved by giving each user a pair of public and secret keys which can be used to encrypt messages. However this is very inefficient: a user who wants to encrypt a message for the group must encrypt it for each group member individually and then broadcast the concatenation of the encrypted parts. A second solution is to share a common key among the group members and use the key to perform the cryptographic operation. This raises the question of how to efficiently add new members to, or remove members from, the group such that security of previous and future communication is guaranteed. When a new user joins the group, the common key can be sent to the new user using secure unicast. However this means that the new user can read all previous encrypted messages. To keep the previous communication secret from the new user, a new common key can be generated and sent to the old group

members encrypted with the old common key, and to the new user using secure unicast. Removing users is a more difficult problem. When users leave the group it is essential to change the group key in order to conceal future communication from the evicted users. This is known as the *user revocation* or *blacklisting* problem.

A simple solution to user revocation problem exists when each user in the group shares an individual secret key with a KDC which controls the group. When a user is to be deleted from the group, the KDC chooses a new common key to be used for encrypting future group messages, encrypts it with the secret key of each user and sends it to them.

In this system the group controller is the trust and communication bottleneck of the system. The controller knows all the keys used by the group members and its compromise results in the complete loss of system security. It is also communication bottleneck of the system and any user revocation requires its participation.

In [25], a secure multicast scheme to deal with the revocation problem without the need for a trusted group controller is proposed. It allows any member of the group to remove a subgroup of members and obtain a shared key with the remaining group members. This can be used to establish conferences within arbitrary subgroup, and initiated by a group member.

The scheme works as follows. Assume there are T users $P_1, \ldots, P_T$, and let $\mathcal{P} = \{P_1, \ldots, P_T\}$. The KDC distributes keys to each user during the system setup. At a later time the users in the group can broadcast messages such that only some designated users can decrypt the messages.

Let $(X, \mathcal{B})$ be a $(2,t)$-CFF(N,T). In the system setup, the KDC randomly selects a set of N keys $k_1, \ldots, k_N$ and for each user P_i gives him a subset $\mathcal{K}_i = \{k_r \mid \text{if } x_r \in B_i\}$ of keys. Assume that a user P_i wants to establish a session key SK with other users of the group except t users, say $P_{\ell_1}, \ldots, P_{\ell_t}$. The user P_i encrypts the session key SK with all his keys except those keys incident to $P_{\ell_1}, \ldots, P_{\ell_t}$, and broadcasts the encrypted message. That is, P_i broadcasts $\{E_{k_r}(SK) \mid r \in B_i \setminus (B_{\ell_1} \cup \cdots \cup B_{\ell_t})\}$, where $E_k(\cdot)$ denotes a symmetric key encryption with key k. From the definition of $(2,t)$-CFF we know that every $P_j \in \mathcal{P} \setminus \{P_{\ell_1}, \ldots, P_{\ell_t}\}$ has at least one key k_r where $x_r \in B_i \setminus (B_{\ell_1} \cup \cdots \cup B_{\ell_t})$, and so he can decrypt $E_{k_r}(SK)$ to obtain SK, while every $P_{\ell_j}, 1 \leq j \leq t$ cannot decrypt the message since he does not have any of the keys used for encryption.

It is not hard to see that the above scheme based on a $(2,t)$-CFF(N,T) can remove up to t users from a group of N users. It requires each user to store fewer than N keys, and the maximum number of transmissions is N.

The system is secure against collusion of t malicious users.

The results on cover-free families in Section 4.2 show that the number of keys that need to be stored by each user is of the order $O(\log T)$ and that the length of the message needed for updating the session key and removing t users is $O(\log T)$, where the message is the concatenation of the session key encrypted with a number of keys.

4.4.4. *Broadcast authentication*

One fundamental goal in cryptography is to ensure integrity of sensitive data, which simply means providing assurance about the content and origin of the communicated and/or stored data. Data integrity is accomplished by means such as digital signature schemes and message authentication codes. In a *digital signature scheme* the signature is generated using the secret key of the signer, and the authenticity is verified by a public verification algorithm. The security of signature schemes relies on some assumed computational complexity of problems such as the discrete logarithm and factorisation problems. A *message authentication code* (or MAC), on the other hand, is a private-key based cryptosystem, requiring to share a secret key between a sender and a receiver ahead of the communication. A typical example of a MAC is constructed by using block ciphers (e.g., DES or AES) in the cipher block chaining (CBC) mode. The MACs based on block ciphers are generally much faster than digital signature schemes, but there is no known proof of security, not even one based on a plausible computational assumption. However, it is possible to construct MACs that can be proved secure, without any computational assumptions. Such MACs are usually called *unconditionally secure authentication codes*.

Conventional authentication systems deal with *point-to-point* message authentication in which the sender and the receiver share a secret key and are both assumed honest. Multi-receiver (or broadcast) authentication systems are an extension of the point-to-point authentication model in which there are multiple receivers who cannot all be trusted. The sender broadcasts a message to all the receivers who can individually verify the authenticity of the message using their secret key information. There are malicious groups of receivers who use their secret keys and all the previous communication in the system to construct fraudulent messages. They succeed in their attack as soon as a single receiver accepts the message as being authentic. In a (t, T) multi-receiver authentication system there are T receivers such that the coalition of any t receivers cannot cheat other

receivers.

A authentication code (or A-code) is a code where the *source state* (i.e. plaintext) is concatenated with an *authenticator* (or a *tag*) to obtain a *message* which is sent through the channel. Such a code is a triple $(\mathcal{S}, \mathcal{E}, \mathcal{T})$ of nonempty finite sets together with an authentication mapping $f : \mathcal{S} \times \mathcal{E} \to \mathcal{T}$. Here $\mathcal{S}$ is the set of source states, $\mathcal{E}$ is the set of keys and $\mathcal{T}$ is the set of authenticators. When the transmitter wants to send the information $s \in \mathcal{S}$ using a key $e \in \mathcal{E}$, which is secretly shared with the receiver, he transmits the message $m = (s, t)$, where $s \in \mathcal{S}$ and $t = f(s, e) \in \mathcal{T}$. When the receiver gets a message $m = (s, t)$, she checks the authenticity by verifying whether $t = f(s, e)$ or not, using the secret key $e \in \mathcal{E}$.

Obviously, a multi-receiver authentication system can be constructed from a conventional authentication code by allowing the sender to use T authentication keys for the T receivers and broadcast a codeword that is simply a concatenation of the codewords for each receiver. The length of the combined authentication tag is T times the length of the individual receiver's authentication tag, and the sender's key is T times the size of a receiver's key. This is a very uneconomical method of authenticating a message as such a system can prevent attacks by even $T-1$ colluding receivers, while it is reasonably realistic to assume that an (t, T) multi-receiver authentication system is sufficient to satisfy the security requirements. That is, we assume that in every group of $t+1$ receivers there is at least one honest receiver. In [39, 40], it has been shown that cover-free families can play a role to improve the above trivial construction of broadcast authentication systems.

Assume that $(X, \mathcal{B})$ is a $(1, t)$-CFF(N, T) with $X = \{x_1, \ldots, x_N\}$ and $\mathcal{B} = \{B_1, \ldots, B_T\}$, and assume that $(\mathcal{S}, \mathcal{E}, \mathcal{T})$, together with the authentication mapping $f : \mathcal{S} \times \mathcal{E} \to \mathcal{T}$, is an authentication code*. We construct a (t, T) multi-receiver authentication system with T receivers $P_1, \ldots, P_T$ as follows. The sender randomly chooses a t-tuple of keys $(e_1, \ldots, e_N) \in \mathcal{E}^N$ and privately sends e_i to every receiver P_j for all j with $x_i \in B_j$, $1 \leq i \leq t$. To authenticate a source message $s \in \mathcal{S}$, the sender computes $a_i = f(s, e_i)$ for all $1 \leq i \leq N$ and broadcasts $(s, a_1, \ldots, a_N)$ to all the receivers. Since the receiver P_j holds the keys e_i for all i with $x_i \in B_j$, P_j accepts $(s, a_1, \ldots, a_N)$ as authentic if $a_i = f(s, e_i)$ for all i satisfying $x_i \in B_j$.

It was proved in [39] that this construction gives rise to a (t, T) broadcast authentication system where both the sizes of the key for the sender and

*$(\mathcal{S}, \mathcal{E}, \mathcal{T})$ can be either an unconditionally secure A-code or a computationally secure MAC.

of the broadcasting message are N times of the underlying point-to-point authentication code, in contrast to the T times increase for the trivial construction. Similarly, in [40] a $(2,t)$-CFF was applied to construct broadcast authentication with dynamic senders.

4.4.5. *Secret sharing schemes*

We describe a variant of cover-free families and its applications in secret sharing schemes. Strong cover-free families were first considered in [8] providing solution to the problem of redistribution of shares in secret sharing schemes. They have been used later [31] as a mechanism for implementing shared encryption and decryption for block ciphers.

Definition 4.9. Let X be a set of N elements (points) and let $\mathcal{B}$ be a set of T subsets (blocks) of X. Then $(X,\mathcal{B})$ is called a *t-strong-cover-free family* provided that, for any $\Delta,\Lambda\subseteq\{1,\ldots,T\}$ with $|\Delta|=t$ and $|\Lambda|=t-1$:

$$\left|\bigcup_{i\in\Delta}B_i\right| > \left|\bigcup_{j\in\Lambda}B_j\right|.$$

Sometimes, we will use the notation t-$SCFF(N,T)$ to denote a t-strong-cover-free family $(X,\mathcal{B})$ in which $|X|=N$ and $|\mathcal{B}|=T$.

A *secret sharing scheme* is a method of protecting a *secret* among a group of *participants* in such a way that only certain specified subsets of the participants (those belonging to the *access structure*) can reconstruct the secret. Secret sharing schemes were first proposed for cryptographic applications in which the secret is a highly sensitive piece of data, and the secret sharing scheme is used to control access to this data by requiring certain subsets of participants to cooperate in order to retrieve the data. Examples of applications include controlling access to a bank vault, installation of high level cryptographic master keys and enabling a nuclear missile.

A secret sharing scheme is normally initialised by an external trusted *dealer* who securely transfers a piece of information relating to the secret, called a *share*, to each participant in the scheme. The first secret sharing schemes proposed by Shamir [42] and Blakley [1] were (t,T)-*threshold schemes* where the access structure consists of all subsets of at least t (out of a total of T) participants. Secret sharing schemes, and in particular threshold schemes, have become an indispensable basic cryptographic tool

in any security environment where active entities are groups rather than individuals.

In certain applications, the set of participants might change or a new threshold value may be necessary. These changes could result from changes in the structure of the organization or level of sensitivity of the secret. A natural question is if it is possible to re-distribute the shares so that the new requirements can be met. More specifically, given an (ℓ, N) threshold scheme, whether it is possible to redistribute the shares in an efficient way such that a new (t, T') threshold scheme can be obtained. The straightforward solution would be to redesign the secret sharing system for the new threshold structure. This is an expensive approach. We see how to use strong cover-free families to achieve this goal without the need to generate new shares, but only redistribute existing shares among the new participants.

Let $(X, \mathcal{B})$ be a t-SCFF(N, T) and

$$\max_{\Lambda}\{|\cup_{i\in\Lambda} B_i|\} < \ell \leq \min_{\Delta}\{|\cup_{j\in\Delta} B_j|\},$$

where Λ and Δ run through all the $(t-1)$-subsets and t-subsets of $\{1, 2, \ldots, T\}$, respectively. Let X be the N shares of an (ℓ, N) threshold scheme. We assign a subset of X to each of the T participants according to the t-SCFF(N, T); that is, participant P_i has a subset of shares B_i. It is easy to see that the new share distribution gives rise to a (t, T) secret sharing scheme.

Cover-free families and their variants are also used to construct other threshold cryptosystems, such threshold block ciphers [31] and threshold MAC [30].

4.5. Conclusions

We have surveyed some known bounds and constructions for cover-free families. We have also presented several interesting applications of cover-free families to topics in secure distributed systems. Cover-free families and their generalisations have been used for many other cryptographic problems such as frameproof codes and traceability schemes [17], multiple time signature schemes [37] and blacklisting problems [24] etc.

References

[1] G. R. Blakley, "Safeguarding cryptographic keys", Proceedings of AFIPS 1979 National Computer Conference, 48, 313–317 (1979).
[2] R. Blom, "An optimal class of symmetric key generation systems", In Proc. Eurocrypt '84, Lecture Notes in Computer Science, Vol. 209, 335–338 (1985).
[3] D. Boneh and J. Shaw, "Collision-secure fingerprinting for digital data", IEEE Trans. Inform. Theory, Vol. 44, 1897–1905 (1998).
[4] I. Bouchemakh and K. Engel, "The order-interval hypergraph of a finite poset and th König property"', Discrete Math, Vol. 170, 51–61 (1997).
[5] B. Bruegge, B. Bennington, "Applications of Mobile Computing and Communication", IEEE Personal Communication, Vol 3, No 1, 64-71 (1996)
[6] H. Chan, A. Perrig, and D. Song, "Random key predistribution for sensor networks", Proceedings of IEEE Symposium on Security and Privacy, 197–213 (2003).
[7] Z. J. Czech, G. Havas and B. S. Majewski, "Perfect hashing", Theoretical Computer Science, Vol. 182, 1–143 (1997).
[8] Y. Desmedt, R. Safavi-Naini and H. Wang, "Redistribution of mechanical secret shares"', Financial Cryptography '02, Lecture Notes in Computer Science, Vol. 2357, 238–252 (2002)
[9] Y. Desmedt, R. Safavi-Naini, H. Wang, L. M. Batten, C. Charnes and J. Pieprzyk, "Broadcast anti-jamming systems", Computer Networks, Vol. 35 (2-3), 223–236 (2001).
[10] W. Du, J. Deng, Y.S. Han, and P.K. Varsheney, "A pairwise key predistribution schemes for wireless sensor network", Proc. of the 9th ACM Conference on Computer and Communications security, 42–51(2003).
[11] A. G. Dyachkov and V. V. Rykov, "Bounds on the length of disjunctive codes", Problemy Paredachi Informatsii, Vol. 18, No. 3, 7–13 (1982).
[12] M. Dyer, T. Fenner, A. Frieze, and A. Thomason, "On key storage in secure networks", J. Cryptography, Vol. 8, 189–200 (1995).
[13] K. Engel, "Interval packing and covering in the boolean lattice"', Combin, Probab. Comput. Vol. 5, 373–384 (1996).
[14] P. Erdös, P. Frankl and Z. Füredi, "Families of finite sets in which no set is covered by the union of r others", Israel J. Math., Vol. 51, 79–89 (1985).
[15] L. Eschenauer and V. Gligor, "A key-management scheme for distributed sensor networks", Proceedings of the 9th ACM Conference on Computer and Communication Security, 41–47 (2002).
[16] A. Fiat and M. Naor, "Broadcast encryption", Advances in Cryptology – CRYPTO '93, Lecture Notes in Computer Science, Vol. 773, 480–491 (1994).
[17] A. Fiat and T. Tassa, "Dynamic traitor tracing", Advances in Cryptology – CRYPTO '99, Lecture Notes in Computer Science, Vol. 1666, 354–371 (1999).
[18] Z. Füredi, "On r-cover-free families", J. Combinatorial Theory Series A, Vol. 73, 172–173 (1996).
[19] L. Gong and D.H. Wheeler, "A matrix key storage scheme", Journal of Cryptology, Vol. 2, 51–59 (1990).

[20] H.-D. O. F. Gronau and R. S. Mullin, "On super-simple $2-(v, 4\lambda)$ designs", J. Combin. Math. Combin. Comput, Vol. 11, 113–121 (1992)

[21] W. H. Kautz and R. C. Singleton, "Nonrandom binary superimposed codes", IEEE Trans. Inform. Theory, Vol. 10, 363–377 (1964).

[22] H.-K. Kim and V. Lebedev, "On optimal superimposed codes", Journ. Combin. Designs, Vol. 12, 79–91 (2003).

[23] H.-K. Kim, V. Lebedev and D. Y. Oh, "Some new results on superimposed codes", Journ. Combin. Designs, (2004).

[24] R. Kumar, S. Rajagopalan and A. Sahai, "Coding constructions for blacklisting problems without computational assumptions", Advances in Cryptology – CRYPTO '99, Lecture Notes in Computer Science, Vol. 1666, 609–623 (1999).

[25] H. Kurnio, R. Safavi-Naini, W. Susilo and H. Wang, "Key management for secure multicast with dynamic constrollers", Information Security and Privacy, 5th Australasian Conference, ACISP00, Lecture Notes in Computer Science, Vol. 1841, 178–190 (2000)

[26] V. Lebedev, "New asymptotic upper bound on the rate of (w, r) cover free codes", Problems of Information Transmission, Vol. 39, 75–89 (2003)

[27] J. Lee and D.R. Stinson, "Deterministic key predistribution schemes for distributed sensor networks", Proc. of SAC, Lecture Notes in Computer Science, Vol. 3357, 294–307 (2004).

[28] D. Liu and P. Ning, "Establishing pairwise keys in distributed sensor networks", ACM Transactions on Information and System Security (TISSEC), Vol. 8, 41–77 (2005)

[29] X. Ma and R. Wei, "On a bound of cover-free families", Designs, Codes and Cryptography, Vol. 32, 303-321 (2004).

[30] K. Martin, J. Pieprzyk, R. Safavi-Naini, H. Wang and P. Wild, "Threshold MACs", 5th International Conference on Information Security and Cryptology (ICISC '02), Lecture Notes in Computer Science, Vol. 2587, 237–252 (2003).

[31] K. Martin, R. Safavi-Naini, H. Wang and P. Wild, "Distributing the encryption and decryption of a block cipher", Designs, Codes and Cryptography, Vol. 36, 263–287 (2005).

[32] K. Mehlhorn, *Data Structures and Algorithms*, Volume 1, Springer, Berlin, 1984.

[33] A. Menezes, P. van Oorschot, and S. Vanstone, *Applied Cryptography*. CRC, Boca Raton, 1996.

[34] C. J. Mitchell and F. C. Piper, "Key storage in secure networks", Discrete Applied Math., Vol. 21, 215–228 (1988).

[35] N. Niederreiter, H. Wang and C. Xing, "Function fields over finite fields and their applications to cryptography," in Topics in Geometry, Coding Theory and Cryptography, by Arnaldo Garcia and Henning Stichtenoth (editors), Springer, 2006, to appear.

[36] H. Niederreiter and C. P. Xing, *Rational points on curves over finite fields: theory and applications*, Cambridge University Press, Cambridge, 2001.

[37] J. Pieprzyk, H. Wang and C. P. Xing, "Multiple-time signature schemes secure against adaptive chosen message attacks", 10th Workshop on Selected Areas in Cryptography (SAC '03), Lecture Notes in Computer Science, Vol. 3006, 88–100 (2004).
[38] M. Ruszinkó, "On the upper bound of the size of the r-cover-free families", J. Combinatorial Theory Series A, Vol. 66, 302–310 (1994).
[39] R. Safavi-Naini and H. Wang, "New results on multireceiver authentication codes", Advances in Cryptology – EUROCRYPT '98, Lecture Notes in Computer Science, Vol. 1403, 527–541 (1998).
[40] R. Safavi-Naini and H. Wang, "Efficient authentication for group communication", Theoretical Computer Science, Vol. 269, 1–21 (2001).
[41] M Satyanarayanan, "Mobile Information Access", IEEE Personal Communication, Vol 3, No 1, 26 -33 (1996)
[42] A. Shamir, How to share a secret, Communications of the ACM, Vol. 22, 612–613 (1976).
[43] J. N. Staddon, D. R. Stinson and R. Wei, "Combinatorial properties of frameproof and traceability codes", IEEE Trans. Inform. Theory, Vol. 47, 1042–1049 (2001).
[44] D. R. Stinson, "On some methods for unconditionally secure key distribution and broadcast encryption", Designs, Codes and Cryptography, Vol. 12, 215–243 (1997).
[45] D. R. Stinson, T. van Trung and R. Wei, "Secure frameproof codes, key distribution patterns, group testing algorithms and related structures", J. Statist. Plan. Infer., Vol. 86, 595–617 (2000).
[46] D. R. Stinson and R. Wei, "Generalised cover-free families", Discrete Mathematics, Vol. 279, 463-477 (2004).
[47] D. R. Stinson, R. Wei and L. Zhu. "Some new bounds for cover-free families", J. Combinatorial Theory Series A, Vol. 90, 224–234 (2000).
[48] A. J. Viterbi, *CDMA principles of spread spectrum communications*, Addison-Wesley, Reading, Massachusetts, 1995.
[49] R. Wei, "On cover-free families", manuscript, 2006.

Chapter 5

Group Rekeying in Multi-Privileged Group Communications for Distributed Networking Services

Guojun Wang[1,3], Jie Ouyang[1], Hsiao-Hwa Chen[2,*] and Minyi Guo[3]

[1]*School of Information Science and Engineering, Central South University, Changsha, Hunan Province, P. R. China, 410083;*
[2]*Institute of Communications Engineering, National Sun Yat-Sen University, Kaohsiung City, 804 Taiwan;*
[3]*School of Computer Science and Engineering, University of Aizu, Aizu-Wakamatsu City, Fukushima 965-8580, Japan.*

Many applications need to support multi-privileged group communications, which contain multiple data streams. Group users can subscribe to different data streams according to their interest and have multiple access privileges. In this paper, we first introduce some existing rekeying schemes for multi-privileged group communications and analyze their advantages and disadvantages. Then we propose an ID-based Hierarchical Key Graph Scheme (IDHKGS) to manage multi-privileged group communications. The proposed scheme employs a key graph, on which each node is assigned a unique ID according to access relations between nodes. When a user joins/leaves the group or changes access privileges, other users in the group can deduce the new keys using one-way function by themselves according to the ID of joining/leaving/changing node on the graph, and thus this scheme can greatly reduce the rekeying overhead.

[*]Prof. Hsiao-Hwa Chen is the corresponding author. He is with the Institute of Communications Engineering, National Sun Yat-Sen University, Kaohsiung City, 804 Taiwan, Tel: +886-7-5254488, Fax: +886-7-5254475, Email: hshwchen@ieee.org.

1. Introduction

With the rapid development of the Internet and the increase of network bandwidth, more and more applications integrate with the Internet. These network applications are based upon unicast or multicast communications. Unicast employs a client-server model, where the server handles the requests of all users and delivers suitable packets to users via a dedicated point-to-point channel. However, unicast is inefficient if all users request the same data stream. On the contrary, multicast is an efficient method for delivery of data from a source to multiple recipients, such as teleconference, information service and live sports. Compared with unicast, multicast can reduce sender transmission overhead and network bandwidth requirements.

In order to guarantee the security of communications, group communications must ensure that a user that isn't a member in a group can't access any communications among the group. In order to achieve this requirement, an encryption key, also known as the Session Key (SK) is shared by all legitimate group members.[1] In addition, in order to ensure forward secrecy and backward secrecy,[2] the SK should be changed after every join and leave so that a former group member has no access to current communications and a new member has no access to previous communications.

In traditional group communication schemes,[3-8] all members in a group have same level of access privilege. In these schemes, if members hold the decryption key, they can access all the content; otherwise, they can't read anything. In order to improve the scalability of these schemes, reducing the communication overhead and the rekeying overhead are the major design concerns.

However, many group applications contain multiple related data streams and the members have different access privileges. For example:

- Multimedia applications distributing contents in multi-layer coding format. In video broadcasting, users with normal TV receivers can receive the contents with the normal format only, while users with HDTV receivers can receive the contents with both the normal format and the extra information needed to achieve HDTV resolution.[9]

- In e-newspaper broadcasting, there are multiple data streams to broadcast the contents of top news, weather forecasts, financial news, stock quotes, and sports news. The service provider also classifies users into several membership groups, such as gold, silver sports, silver finance and basic. In such an application, different membership groups can access different contents.[10]

Key management in multi-privileged group communications is crucial and complicated due to two factors. First, the service generally provides multiple data streams and encrypts different data streams using separate SKs.[11] Users can subscribe to one or multiple data streams and should have the corresponding SKs for the purpose of security. The challenge is how to manage these keys while ensuring that no users can access the key and data beyond their privileges. Second, not only the users can join or leave the group at will, but also the users can change their access privileges according to their interest at any time. Hence, a key management scheme should be flexible to accommodate users' join/leave/change requirements. These challenging issues raise the critical problem of how we can efficiently manage the keys when users join/leave/change their access privileges. In this paper, we will present a new multi-privileged group rekeying scheme. The proposed scheme employs a key graph to manage SKs and exploits a one-way function to update the keys[12] in order to reduce the rekeying messages in the join/leave and change operations.

The rest of the paper is structured as follows. In Section 2, we describe the service model and logical key hierarchy. In Section 3, we introduce some existing rekeying schemes for multi-privileged group communications. In Section 4, we propose a novel rekeying scheme. Finally, we conclude this paper in Section 5.

2. Preliminaries

In this section, we first introduce the basic concepts that describe the group communication systems containing multiple data streams and users with different access privileges, and then describe the basic idea of key tree in group communications containing single data streams.

2.1. *System descriptions*

2.1.1. *One-dimensional data stream*

Let $\{r_1, r_2, ...\}$ denote the set of *resources* in a group communication system. In such a system, each resource corresponds to a data stream.

A Data Group (DG) consists of a set of users that can access to a particular resource. Obviously, the DGs can have overlapped membership because users may subscribe to multiple resources. The DGs are denoted by $D_1, D_2, ..., D_M$, where M is the total number of the DGs. A Service Group (SG) consists of a set of users who can access the exactly same set of resources. The users in each SG have same access privilege. The SGs have non-overlapped membership. The SGs are denoted by $S_1, S_2, ..., S_I$, where I is the total number of SGs. In order to make clear mathematical the typical access relationships in group communications, t_m^i is defined as:

$$t_m^i = \begin{cases} 1, \text{ the users in SG } S_i \text{ subscribe to resource } r_m \\ 0, \text{ otherwise} \end{cases}$$

for $i = 1, ..., I$, and $m = 1, ..., M$. In addition, S_0 is defined as a virtual SG, which represents users who do not participate in any group communications.

The following Example 1 and Example 2 are two typical applications of multimedia group communications.[9]

Example 1: Multimedia applications that distribute contents in multi-layer format. The access relations are illustrated in Table 1.

Table 1 Multi-layer service groups and their access relations

Access relation	D_1 (r_1: base layer)	D_2 (r_2: enhancement layer 1)	D_3 (r_3: enhancement layer 2)
$S_{\{001\}}$	√		
$S_{\{011\}}$	√	√	
$S_{\{111\}}$	√	√	√

Example 2: Multicast programs containing several related services, as shown in Table 2.

Table 2 Cellular phone service groups and their access relations

Access relation	D_1 (r_1: news)	D_2 (r_2: stock quote)	D_3 (r_3: traffic/weather)
$S_{\{001\}}$	√		
$S_{\{010\}}$		√	
$S_{\{100\}}$			√
$S_{\{011\}}$	√	√	
$S_{\{101\}}$	√		√
$S_{\{110\}}$		√	√
$S_{\{111\}}$	√	√	√

2.1.2. *Multi-dimensional data stream*

An MPEG-4 FGS video frame, supporting T PSNR service levels and M bitrate service levels, is divided into $T \times M$ different two-dimensional units.[13] A single tile JPEG 2000 frame can support 4-dimensional scalability innately: resolution, quality, component and precinct. That is, data streams are scalable in multiple dimensions.

Suppose there is a scalable video with 2 resolution levels and 3 quality layers. The group has 6 data streams, as shown in Table 3.

Table 3 Data streams in video

	Lowest quality	Middle quality	Full quality
Resolution level 0	r_{00}	r_{01}	r_{02}
Resolution level 1	r_{10}	r_{11}	r_{12}

Users that subscribe to r_{ij} can access a scalable unit of resolution i and quality layer j. Similarly, a Service Group (SG) defines a set of users who receive the same set of data streams. When an SG has a privilege to access the video stream at resolution 0 with full quality, the users in this SG can receive r_{00}, r_{01} and r_{02}.

Each data stream needs one unique SK to encrypt the data. In order to achieve access control, the users in each DG share an SK. If a user subscribes to multiple data streams, it needs an SK for each data stream. When a user joins or leaves the group, the SKs the user holds must be changed. However, when a user switchs between SGs, it is unnecessary to change SKs for data streams to which the user is still subscribing.

2.2. *Logical key hierarchy*

Logical Key Hierarchy (LKH)[14] scheme provides an efficient and secure mechanism to manage the keys and to coordinate the key update. The LKH employs a hierarchical tree whose root node is associated with a group key and whose leaf nodes are individual keys of all users in the group. The intermediate nodes correspond to Key Encryption Key (KEK). Each user in the group holds a set of keys on the path from its leaf to the root. Consider a multicast group with six users. The KDC constructs a hierarchy of keys as shown in Fig. 1. The root node $k_{1\text{-}6}$ is group key, and the user u_2 owns k_2, $k_{1\text{-}2}$, $k_{1\text{-}4}$ and $k_{1\text{-}6}$.

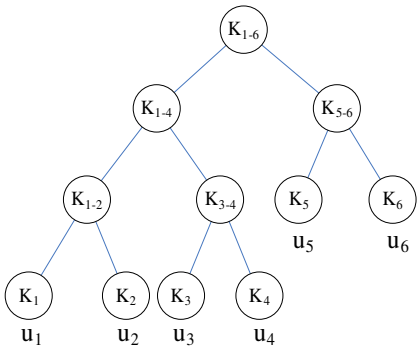

Fig. 1 Logical key hierarchy

Waldvogel et al.[15] present an efficient scheme to update the key tree when users dynamically join or leave. Each key contains a unique key ID, a version field and a revision field, as shown in Fig. 2. When a user wants to join the group, the KDC assigns a leaf node to represent the new user, and increases the revision numbers of all keys on the path from the leaf node to the root by passing the keys through a one-way function. When a

user notices the revision change in ordinate data packet, the user updates the keys with new revision number from old key using the one-way function. In this case, the KDC needs only to send one rekeying message to the new user. In addition, when a user wants to leave the group, the KDC updates the keys that are held by the leaving user. The number of rekeying messages for a user leaving increases linearly with the logarithm of group size.

Fig. 2 Structure of a key

2.3. *Requirements of the rekeying schemes for multi-privileged group communications*

Obviously, it is impossible to directly apply the logical key hierarchy to multi-privileged group communications. In order to achieve hierarchical access control, a simple method is to construct a separate key tree for each DG. The leaves are associated with the users in each DG. The method is very simple and it is easy to manage the keys and the communications. But the method can bring redundancy because DGs have overlapped membership and doesn't scale well when the number of data streams increases.

Therefore, the rekeying schemes for multi-privileged group communications should provide security, flexibility and scalability.

- Security: Each user may subscribe to one or multiple resources. The rekeying schemes must prevent the user from accessing any data before he joins or after he leaves the group. In addition, the rekeying schemes must ensure that the user can't access the data that he doesn't subscribe to.
- Flexibility: Besides joining or leaving the group, the users may change their access privileges, which can be considered that the users switch between different SGs. Because of the dynamics of users, the rekeying schemes need to support users' join/leave/ switch at any time.

- Scalability: When a new SG joins the group communications or an SG in a group decomposes, it should not lead to the reconstruction of the structure for key management. In other words, it should support the dynamic service group formation and decomposition.

3. The Existing Key Management Schemes

3.1. *Multi-group key management scheme (MGKMS)*

In order to eliminate the redundancy because of overlapped membership among DGs, Sun and Liu[9] propose a Multi-Group Key Management Scheme (MGKMS).

3.1.1. *Key graph construction*

The MGKMS scheme employs an integrated key graph to manage the keys when a user joins/leaves/switches. The key graph is constructed as follows:

(i) The KDC constructs an SG-subtree for each SG. The root of the SG_i-subtree is the SG key K_i^S. The leaves are the users in the SG S_i.
(ii) The KDC constructs a DG-subtree for each DG. The root of the DG_m-subtree is the DG key K_m^D. The leaves are the SG keys in which the users can access the resource r_m.
(iii) The KDC generates the key graph by connecting the leaves of the DG-subtrees and the roots of the SG-subtrees.

The procedure of constructing the integrated key graph is illustrated in Fig. 3. Suppose each SG has 4 users. Each user in S_i holds a set of keys on the paths from the leaf to the root of the DG-subtree of $\{D_m, \forall\, m:\ t_m^i = 1\}$.

3.1.2. *Rekeying algorithm*

Here, a switching user changes the SG from S_i to S_j. ϕ_i denotes a set of keys that the user holds in S_i, and ϕ_j denotes a set of keys that the user holds in S_j. The rekeying algorithm consists of 2 steps as follows:

(i) The KDC updates the keys in $\overline{\phi}_i \cap \phi_j$ through a one-way function and increases the revision numbers of these keys. When users notice that the revision numbers of keys they hold increase, they compute the new keys using the same one-way function.

(ii) The KDC updates the keys in $\phi_i \cap \overline{\phi}_j$, increases their version numbers and sends the new keys encrypted with their children keys to the users.

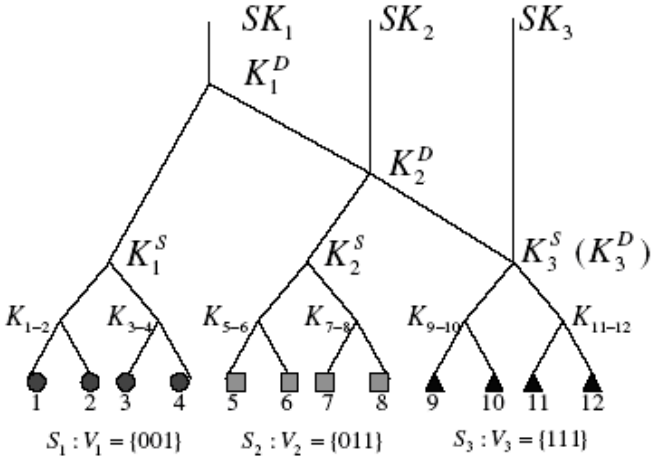

Fig. 3 Multi-group key management graph construction

3.1.4. *Summary*

The MGKMS scheme can achieve the forward and backward secrecy when users join/leave the group or switch between different SGs. Compared with the tree-based key management scheme in single multicast communications, it can greatly reduce the storage, computation and communication overheads. However, if there are complicated relations between SGs and DGs, the merging key graph step will also be complicated. In addition, the scheme can't flexibly deal with formation and decomposition of SGs.

3.2. *Hierarchical access control key management scheme (HACKMS)*

In many applications, users and data streams both form a partially ordered hierarchy, but the above MGKMS scheme only considers the former. Therefore, a Hierarchical Access Control Key Management Scheme (HACKMS)[10] is proposed, which considers both partially ordered users and partially ordered data streams.

Take Fig. 4 as an example, by using the MGKMS scheme, the key graph has five DGs and five SKs. However, in the HACKMS scheme, it needs only three SKs. Because in the HACKMS scheme, the data streams which can be accessed by same SGs are merged into one resource group, and each resource group needs only one SK even though it contains multiple data streams.

Access Relation	Sports	Financial	Stock	Top News	Weather
Gold	√	√	√	√	√
Silver Sport	√			√	√
Silver Finance		√	√	√	√
Basic				√	√

Fig. 4 E-newspaper service groups and their access relations

3.2.1. *Key graph construction*

Based on the access relations shown in Fig. 4 and according to the Directed Acyclic Graph (DAG) of SGs (Fig. 5) and the DAG of resource groups (Fig. 6),[16] the unified DAG (Fig. 7) is formed in order to unify the relations of SGs and resource groups.

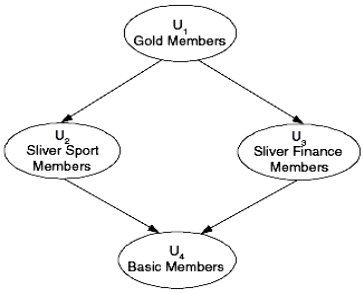

Fig. 5 SG DAG

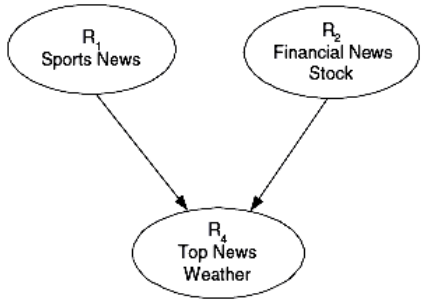

Fig. 6 Resource group DAG

The HACKMS scheme presents an algorithm to construct a key graph based unified DAG shown in Fig. 8. The algorithm traverses the unified DAG in breath-first search and constructs the key graph from bottom to top. The main steps of the algorithm are illustrated as follows:

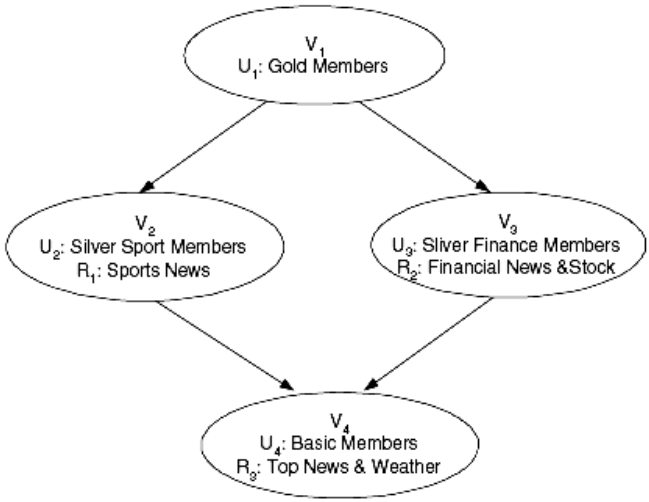

Fig. 7 Unified DAG

(i) The KDC constructs a subtree for each SG.
(ii) Each vertex in unified DAG is colored white.
(iii) If vertex V_i is visited and all vertices that are adjacent to V_i are colored black, then the V_i is colored black.
(iv) The KDC constructs a tree for all SGs in V_i and in all vertices reachable to V_i, whose leaf nodes are associated with the roots of SG-subtrees and whose root node is notated rk_i.
(v) If the vertex V_i contains resource group R_i, rk_i is replaced by the resource group key dk_i.
(vi) Jump to (iii), until all vertices in unified DAG are colored black.

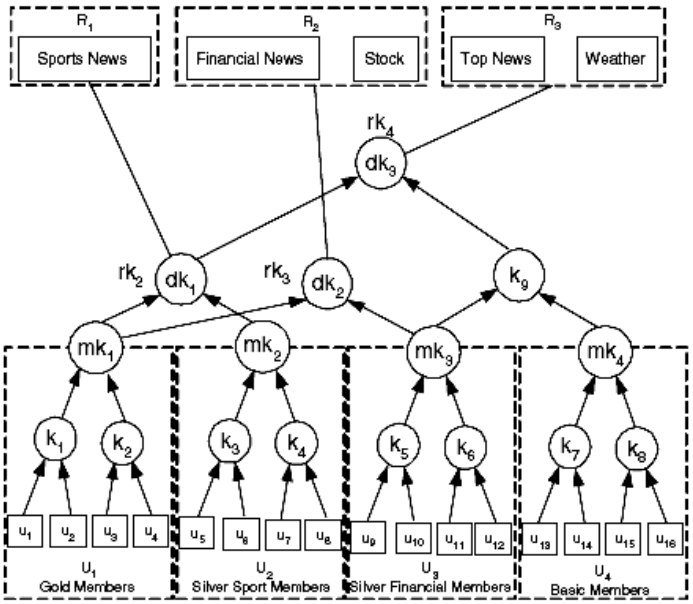

Fig. 8 The key graph by the HACKMS scheme

3.2.2. *Summary*

The HACKMS scheme considers the partially ordered relationship among data streams, and the number of auxiliary keys of DGs and SKs is less than that of the MGKMS scheme. So, compared with the MGKMS scheme, it reduces the storage and rekeying overheads at key server and users. But the construction of key graph is a little bit complicated, because it must first form a unified DAG for SGs and resource groups. In addition, if the partially ordered relationship of data streams is changed, the key graph must be reconstructed.

3.3. *Dynamic access control scheme (DACS)*

The above schemes only support one-dimensional data streams. However, in some applications, there are multi-dimensional data streams.[17] Therefore, Dynamic Access Control Scheme (DACS)[18] is proposed, which supports not only one-dimensional data streams, but

also multi-dimensional data streams. Similarly, this scheme employs a key graph to manage the keys.

3.3.1. *Key graph construction*

(i) The KDC constructs a subtree for each SG. The root node of SG-subtree S_i is SG key srk_i.

(ii) The root of SG-subtree S_i is associated with an Access Key (AK) set Ω_i. The AK set consists of the SKs of the scalable streams that the users in S_i can access. The CEK set consists of all the unit encryption keys of a scalable stream.

The key management graph is shown in Fig. 9. Each user in S_i holds the keys on the path from the leaf node to the root srk_i and an AK set Ω_i.

3.3.2. *Rekeying algorithm*

Suppose a user u switches from S_i to S_j, the rekeying steps are illustrated as follows:

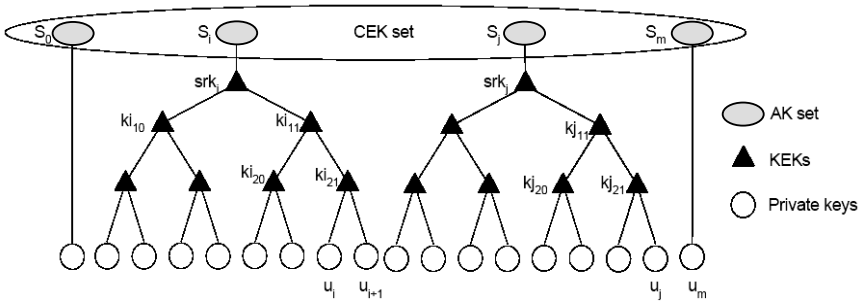

Fig. 9 Key management graph supporting multiple service groups

(i) Update of the keys on the path from the individual key of u to the srk_i. The KDC generates these new keys, encrypts them using their children keys and distributes them to users in S_i.

(ii) Update of the keys in $\Omega_i \cap \overline{\Omega}_j$. The KDC generates a secret ck_s to update the new keys in $\Omega_i \cap \overline{\Omega}_j$ from old keys using a one-way function such that $k' = H_{cks}(k)$ (k' denotes the new version of the old k), and increases the version numbers of those keys. The KDC

encrypts the ck_s with the SG key srk_l (where $\Omega_l \cap (\Omega_i \cap \overline{\Omega}_j) \neq \phi$) and sends out the rekeying messages. The affected users notice the version change in data packet, and compute the new keys using the same one-way function.

(iii) Update of the keys on the path from the new joining user u to the srk_j. The KDC updates all keys on the path from the new leaf to srk_i using a one-way function and increases the revision numbers of those keys. The users notice the revision change and update the keys using the same one-way function.

(iv) Update of the keys in $\overline{\Omega}_i \cap \Omega_j$. The KDC updates the keys in $\overline{\Omega}_i \cap \Omega_j$ using a one-way function and increases the revision numbers of those keys. The affected users update the keys using the same one-way function.

Notice that, if a new service group S_m is formed, then only the step 4 in rekeying algorithm is needed to update the keys.

3.3.3. *Summary*

This scheme isn't only suitable for the multi-dimensional data streams but also flexible for the dynamic service group formation and decomposition. It scales well when the new SG is formed. In addition, the storage overhead and the rekeying overhead are less than those in the MGKMS scheme because of no auxiliary keys in DG-subtrees.

3.4. *Distributed key management scheme (DKMS)*

In Distributed Key Management Scheme (DKMS),[19] every SG maintains an SG server to be used to manage all the users in the SG. The DKMS proposed a structure that includes two parts: DG part and SG part. The DG part consists of all SG servers and is used to manage those servers. The SG part includes an SG server and all users in this SG.

3.4.1. *Key graph construction*

(i) All SG servers form an SG Server Group (SGSG) and one group key is assigned to the SGSG. In addition, each SG sever i holds the

SG key k_i^S and the related SKs of data streams that the users in S_i can access.

(ii) Each SG server constructs a subtree. The root node of the SG-subtree S_i is associated with the SG key, k_i^S.

(iii) The SG servers connect the SG keys to the root of the SG-subtrees.

The structure of the DKMS is shown in Fig. 10.

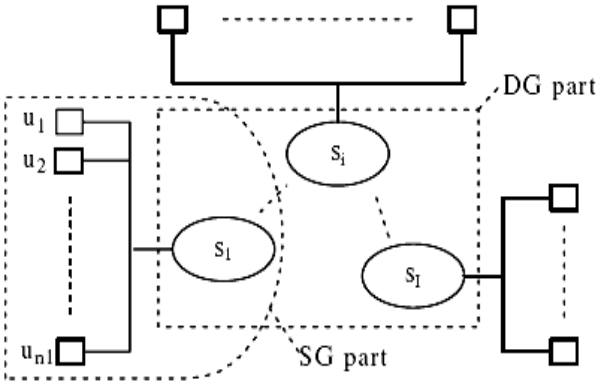

Fig. 10 Structure for DKMS

Each user in SG holds the keys on the path from the leaf node to the root of the SG-subtree and the related SKs of data streams that the users in the SG can access. Each SG server holds the SG-subtree, the related SKs and the group key of SGSG.

3.4.2. *Rekeying algorithm*

Suppose a user u wants to join the SG S_i:

(i) The SG server inserts u at the end of one of the shortest paths of the SG-subtree S_i, updates the keys on the path from the leaf to the root using a one-way function, and increases the revision numbers of those keys. The users notice the revision change and compute the new keys using the same one-way function.

(ii) The related SKs that the SG server i holds should be updated. The SG server i negotiates the new SKs with other SG servers and multicasts the new SKs encrypted with the group key of SGSG.

(iii) The affected SG servers update the related SKs and multicast the new SKs encrypted with the SG keys.

Suppose a user u wants to leave the SG S_i:

(i) The keys on the path from the leaving user to the root of the SG-subtree S_i should be updated. The SG server i generates the new keys and multicasts the new keys encrypted with their children keys.
(ii) The related SKs in S_i should be updated. The SG server i negotiates the new SKs with other SG servers and multicasts the new SKs encrypted with the group key of SGSG.
(iii) The affected SG servers update the related SKs and multicast the new SKs encrypted with the SG keys.

3.4.3. *Summary*

Compared with the MGKMS scheme, both the storage overhead and the rekeying overhead can be reduced. And this scheme supports the service group formation and decomposition. However, compared with the DACS scheme, forming a new SG in DKMS is more complicated because the group key of SGSG shared by all SG servers needs to be updated.

4. Our Proposed Scheme

We propose an ID-based Hierarchical Key Graph Scheme (IDHKGS) to manage multi-privileged group communications. The proposed scheme employs a key graph[9] and each node is assigned an ID to uniquely identify a key.

The key graph contains two types of nodes: u-nodes which contain individual keys and k-nodes which contain SG keys, DG keys and auxiliary keys. The proposed scheme differs from the MGKMS scheme in two aspects, i.e., the identification of a key and the rekeying operation. In the proposed scheme, as long as a user knows the IDs of another user's u-node in the group, it can deduce the IDs of k-nodes on the paths from the u-node to the SK nodes which contain SKs. In addition, we update the keys using a one-way function for the old users to compute the new keys by themselves when a user joins/leaves the group or switches between different SGs.

4.1. *Identification of a key*

In our proposed scheme, the key graph contains two parts, the SG part and the DG part. The SG part is composed of all SG-subtrees, and the DG part is composed of all SK nodes and the *k*-nodes between the SG *k*-nodes and the SK nodes on the key graph. The SGs are denoted by S_2, S_3, ... , S_i, ... , where i is a prime number. The server assigns two integers as the ID of each node on the key graph. In each SG-subtree, a node is identified by the SG i (i= 2, 3, 5, ...) to which the node belongs and by the position m ($m \geq 0$) which is numbered from the root of its SG-subtree in a top-down and left-right order. The node <i, 0> is the root of the S_i-subtree. We observe that the IDs of a node and its parent node have the following simple relationship: $k_{<i, \lfloor (m-1)/2 \rfloor>}$ is the parent node of $k_{<i, m>}$.

In the DG part, if a node has two children nodes <j_1, n_1> and < j_2, n_2>, the node is identified by j that is the least common multiple of j_1 and j_2 and by n (n = max (j_1, j_2)). If a node only has one child node <j_1, n_1>, such as the SK node, the node is identified by j_1 and by -1. Fig. 11 illustrates the IDs of nodes in Fig. 3 of Section 3.

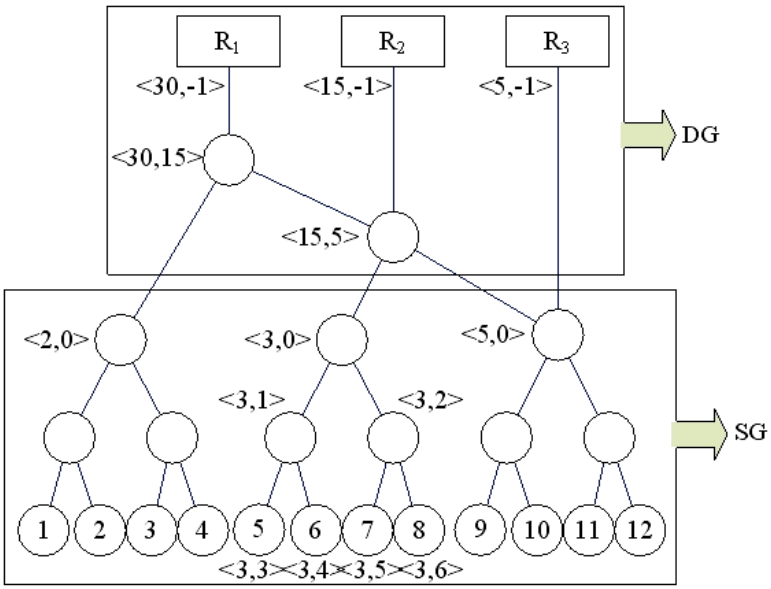

Fig. 11 Illustration of key identification

A user in SG S_i holds a set of keys on the paths from the leaf to the root of the DG-subtree of $\{D_m, \forall\, m : t_m^i = 1\}$. When a user in a group knows the ID of u_5's u-node, <3, 3>, then this user can deduce that user u_5 holds $k_{<3,\,1>}$, $k_{<3,\,0>}$, $k_{<15,\,5>}$, $k_{<15,\,-1>}$, $k_{<30,\,15>}$, $k_{<30,\,-1>}$.

In order to maintain forward secrecy and backward secrecy, a rekeying operation is executed when a user joins/leaves a group or switches between SGs.

4.2. *Rekeying algorithm*

In order to maintain the forward secrecy and backward secrecy, a rekeying operation is executed when a user joins/leaves a group or switches between SGs.

4.2.1. *Single user join*

When a user joins or switches between SGs, the IDs of some u-nodes in the SG part will be changed and users should know the up-to-date IDs of their u-nodes. People prove that a user can deduce its current ID by knowing its old ID and the maximum ID of the current k-nodes.[20] Then, a user in SG can compute its current ID using the same method in our proposed scheme.

Suppose that a user u requests to join S_i. The server inserts it at the end of one of the shortest paths of the S_i-subtree, assigns $<i, m>$ to the new u-node, broadcasts the ID of the new u-node, $<i, m>_J$ (where J is named after the join operation of the joining u-node), and the maximum ID of the current k-nodes in S_i, $<i, n_k>$. When a user receives the broadcast messages, the users in the group can deduce the new keys using a one-way function so that $k' = f(k)$ (where k' denotes the updated version of key k). If $k_{<i,\,n>}$ is a newly created one, the new key is $k'_{<i,\,n>} = f(k_{<i,\,l>} \oplus k_{<i,\,0>})$, where $k_{<i,\,l>}$ is the key of the spitted u-node.

We explain the join operation of user u_{13}, as shown in Fig. 12. When a user u_{13} joins the group, a new u-node is created to hold u_{13}'s individual key. The server broadcasts $<3, 8>_J$ (the ID of the new user u_{13}'s u-node) and <3, 3> (the maximum ID of the k-node in S_3 after u_{13} joins the group).

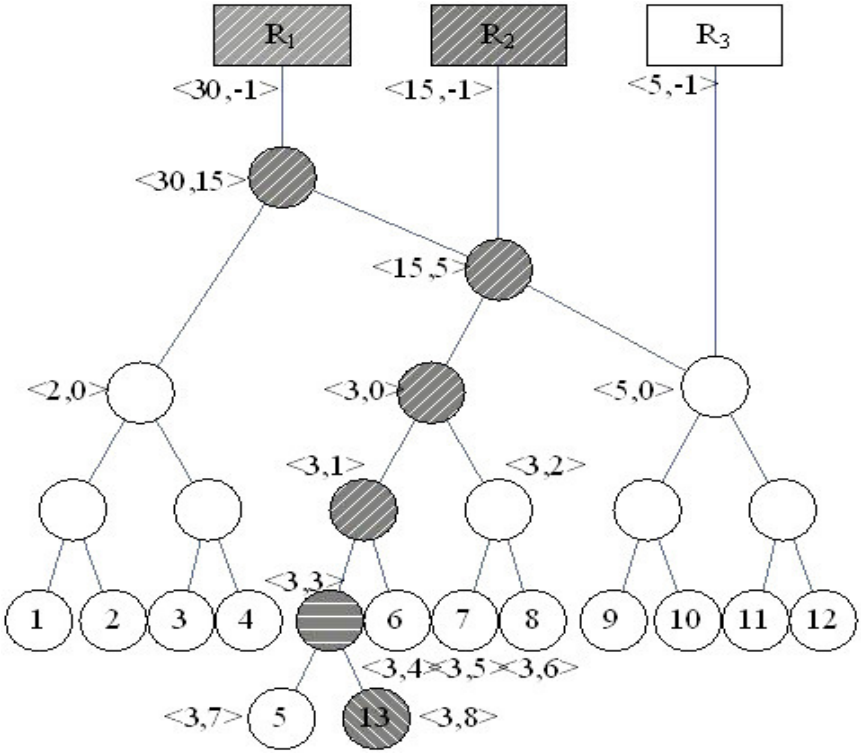

Fig. 12 Key graph after u_{13} joins

According to the IDs, the users in the group can deduce that $k_{<3,\,1>}$, $k_{<3,\,0>}$, $k_{<15,\,5>}$, $k_{<15,\,-1>}$, $k_{<30,\,15>}$ and $k_{<30,\,-1>}$ need to be updated and $k_{<3,\,3>}$ is the newly created one. Then, the users compute the new key values using a one-way function by themselves. The new keys are:

$$k'_{<3,\,1>} = f(k_{<3,\,1>}), \quad k'_{<3,\,0>} = f(k_{<3,\,0>}),$$
$$k'_{<15,\,5>} = f(k_{<15,\,5>}), \quad k'_{<15,\,-1>} = f(k_{<15,\,-1>}),$$
$$k'_{<30,\,15>} = f(k_{<30,\,15>}), \quad k'_{<30,\,-1>} = f(k_{<30,\,-1>}).$$
$$k'_{<3,\,3>} = f(k_{<3,\,7>} \oplus k_{<3,\,0>}).$$

Finally, the server only needs to encrypt all new keys for the newly joining user.

$$s \rightarrow u_{13}: \{ k'_{<3,\,3>}, k'_{<3,\,1>}, k'_{<3,\,0>}, k'_{<15,\,5>}, k'_{<15,\,-1>}, k'_{<30,\,15>}, k'_{<30,\,-1>} \}_{k_{<3,8>}}$$

4.2.2. *Single user leave*

Suppose that a user u requests to leave S_i. Then all the keys the user u holds must be updated. The server broadcasts the ID of the leaving u-node that is $<i, n>_L$ (where L is named after the leave operation of the leaving u-node). The users in the group deduce the new keys. The user in SG S_i computes the IDs from the leaving node to the root of S_i-subtree and updates these keys through a one-way function such that $k' = f(k \oplus k_1)$ where k_1 is one of the auxiliary keys that is not on the leave paths.

For the DG part keys, if a user holds $k_{<j, n>}$ where i is a common factor of j, then $k_{<j, n>}$ needs to be updated. $k_{<j, n>}$ has two children nodes, $k_{<n, l_1>}$ and $k_{<j/n, l_2>}$. If i is not a common factor of n, the new key is $k'_{<j, n>} = f(k_{<j, n>} \oplus k_{<n, l_1>})$ (l_1=0 when n is a prime number, otherwise l_1!=-1,0). Users who hold $k_{<n, 0>}$ or $k_{<n, l_1>}$ can compute the new key by themselves. If i is a common factor of j/n, the new key is $k'_{<j, n>} = f(k_{<j, n>} \oplus k_{<j/n, l_2>})$ (l_2=0 when j/n is a prime number, otherwise l_2!=-1,0). Users who hold $k_{<j/n, 0>}$ or $k_{<j/n, l_2>}$ can compute the new key. If $k_{<j, n>}$ only has one child, such as the SKs of data streams to which the leaving user can subscribe, $k'_{<j, n>} = f(k_{<j, n>} \oplus k'_{<j, l>})$ where $k'_{<j, l>}$ is updated child k-node of $k_{<j, n>}$.

The server only needs to encrypt and send these new keys to the users who can not deduce them.

We explain the leave operation of user u_8, as shown in Fig. 13. The server removes the u-node of u_8. The server broadcasts $<3, 6>_L$ (the ID of the leaving user u_8's u-node). The users in S_3 can deduce that $k_{<3, 2>}$ and $k_{<3, 0>}$ need to be updated and $k_{<3, 5>}$ and $k_{<3, 1>}$ are chosen to compute $k_{<3, 2>}$ and $k_{<3, 0>}$, respectively.

$$k'_{<3, 2>} = f(k_{<3, 2>} \oplus k_{<3, 5>}), \quad k'_{<3, 0>} = f(k_{<3, 0>} \oplus k_{<3, 1>}).$$

In the DG part, the users deduce that $k_{<15, 5>}$, $k_{<30, 15>}$ $k_{<15, -1>}$ and $k_{<30, -1>}$ need to be updated and the new keys except the SKs are:

$$k'_{<15, 5>} = f(k_{<15, 5>} \oplus k_{<5, 0>}), k'_{<30, 15>} = f(k_{<30, 15>} \oplus k_{<2, 0>}).$$

The server needs to encrypt and send the new keys to the users that can not deduce them. $s \rightarrow u_7$: $\{k'_{<3, 0>}\}_{k'_{<3, 2>}}$.

$s \rightarrow u_5 - u_7$: $\{k'_{<15, 5>}\}_{k'_{<3, 0>}}$, $s \rightarrow u_5 - u_7$, $u_9 - u_{12}$: $\{k'_{<30, 15>}\}_{k'_{<15, 5>}}$.

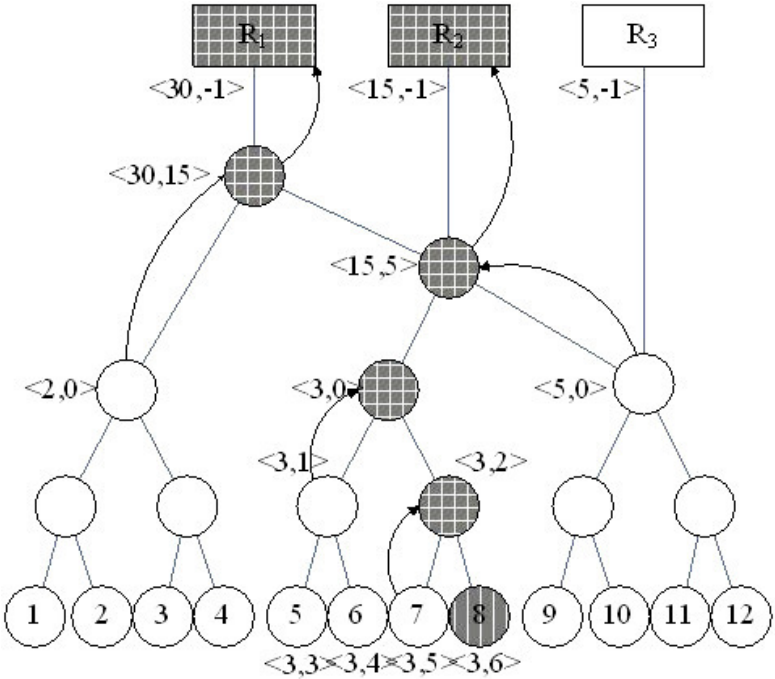

Fig. 13 Key graph after u_8 leaves

In addition, the server computes the new SKs,

$$k'_{<30,\,-1>} = f(k_{<30,\,-1>} \oplus k'_{<30,\,15>}),\ k'_{<15,\,-1>} = f(k_{<15,\,-1>} \oplus k'_{<15,\,5>}),$$

and it encrypts and sends them to the users.

$$s \rightarrow u_1 - u_7,\ u_9 - u_{12} : \{k'_{<30,\,-1>}\}_{k'_{<30,\,15>}},$$
$$s \rightarrow u_5 - u_7,\ u_9 - u_{12} : \{k'_{<15,\,-1>}\}_{k'_{<15,\,5>}}.$$

4.2.3. *Single user switch*

In multi-privileged group communications, the users can flexibly change their access privileges according to their interest at any time. That is, the users are able to switch between different SGs.

Suppose that a user wants to switch from S_i to S_j, which can be considered as that the user first leaves S_i and then joins S_j. The server broadcasts the IDs of leaving/joining node and the maximum ID of the current k-nodes in S_j, $<i, n>_{SL}$, $<j, m>_{SJ}$ and $<j, n_k>$, where SL is named

after the leave operation of the switching user *u*-node in S_i and *SJ* is named after the join operation of the switching user *u*-node in S_j. The users deduce the new keys using one-way function that are similar to the join operation and the leave operation.

We explain the single user switch operation in Fig. 14. A user u_8 wants to switch from S_2 to S_1. A new *u*-node in S_1 is created to hold u_8's individual key. The server broadcasts the ID of the switching user's old *u*-node and new *u*-node and the maximum ID of the *k*-node in S_1 after u_8 switches to S_1, $<3, 6>_{SL}$, $<2, 6>_{SJ}$ and <2, 3>. The users deduce that $k_{<2, 1>}$, $k_{<2, 0>}$, $k_{<3, 2>}$, $k_{<3, 0>}$ need to be updated and $k_{<2, 1>}$ is a newly created one.

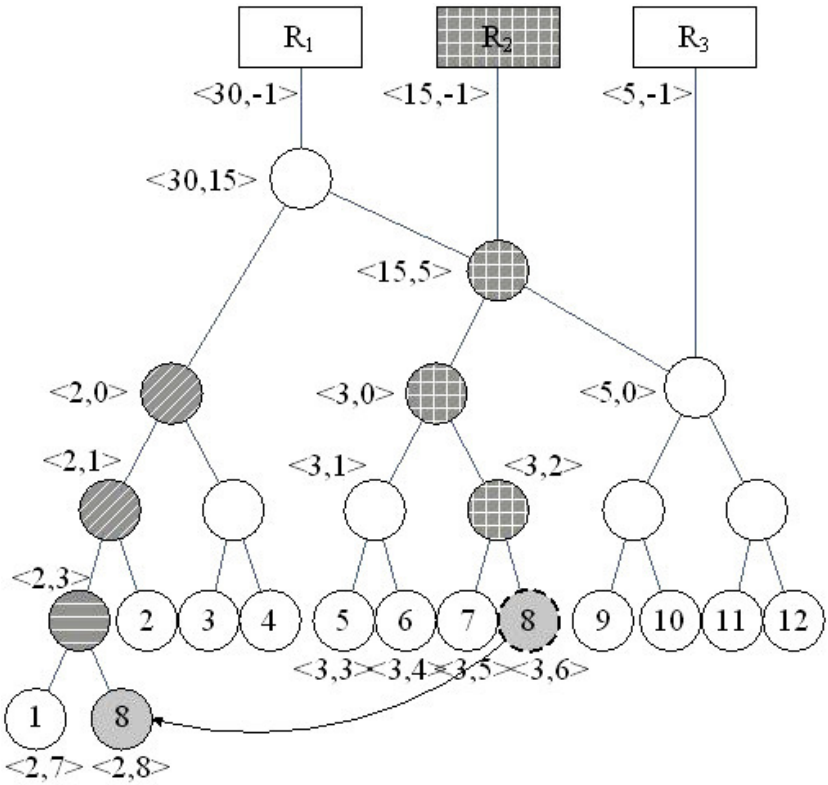

Fig. 14 Key graph after u_8 switches from S_2 to S_1

The new keys are computed as follows:

$$k'_{<2,\,1>}=f(k_{<2,\,1>}),\quad k'_{<2,\,0>}=f(k_{<2,\,0>}),$$
$$k'_{<2,\,3>}=f(k_{<2,\,7>}\oplus k_{<2,\,0>}),\quad k'_{<3,\,2>}=f(k_{<3,\,2>}\oplus k_{<3,\,5>}),$$
$$k'_{<3,\,0>}=f(k_{<3,\,0>}\oplus k_{<3,\,1>}).$$

In the DG part, the users deduce that $k_{<15,\,5>}$ and $k_{<15,\,-1>}$ need to be updated,

$$k'_{<15,\,5>}=f(k_{<15,\,5>}\oplus k_{<5,\,0>}).$$

The server encrypts $k'_{<3,\,0>}$ and $k'_{<15,\,5>}$, and sends them to the users who can not deduce them.

$$s\rightarrow u_7\colon \{k'_{<3,\,0>}\}_{k'_{<3,2>}},\qquad s\rightarrow u_5-u_7\colon \{k'_{<15,\,5>}\}_{k'_{<3,0>}}.$$

Finally, the server computes the new SKs,

$$k'_{<15,\,-1>}=f(k_{<15,\,-1>}\oplus k'_{<15,\,5>}),$$

and sends them to the corresponding users.

$$s\rightarrow u_5-u_7,\ u_9-u_{12}:\{k'_{<15,\,-1>}\}_{k'_{<15,5>}}.$$

4.2.4. *Batch update operation*

If users join, leave or switch frequently, the individual rekeying operations, that is, rekeying after each join, leave or switch request, has very large rekeying overhead. In periodic batch rekeying,[21] the server collects all join, leave and switch requests. At the end of each rekeying period of time, the server processes all requests, generates new keys and sends them to the corresponding users. In our proposed scheme, when the SG-subtrees that the users want to join or switch to become full binary trees, the server splits nodes after the rightmost *k*-node at the highest level to accommodate the extra joins. Firstly, the server labels the *k*-nodes, and the process consists of 2 steps as follows:

(i) The server removes the *u*-nodes of leaving users and switching users in the SGs from which the users switch, labels all *k*-nodes on the leave paths as LEAVE.
(ii) The server labels the newly created *k*-nodes as NEW, labels all *k*-nodes on the join paths as JOIN.

To the switching users, the server labels the *k*-nodes which the users hold originally but do not hold after the switch operation as LEAVE, labels the *k*-nodes which the users do not hold originally but hold after the switch operation as JOIN.

After the key graph is labeled, the server needs to broadcast the IDs of all joining users, leaving users and switching users, $<a_i, b_i>_L$, $<c_j, d_j>_J$, $<e_p, f_p>_{SL_p}$, $<g_p, h_p>_{SJ_p}$, and the IDs of the *k*-nodes in some SGs that the users want to join and switch to. According to the broadcast IDs, the users label the *k*-nodes which they hold as JOIN, LEAVE or NEW. Then, the users can deduce the new key for all labeled *k*-nodes according to the following three cases.

Case 1: As shown in Fig. 15, if the *k*-node is labeled as LEAVE, whether or not it is labeled as JOIN, the users compute the new key value as follows.

(i) i is a prime number. The operation is similar to the leave operation. The new key is $k'_{<i,j>} = f(k_{<i,j>} \oplus k_{<i,l>})$ when $k_{<i,l>}$ is not labeled. If both two children nodes are labeled, the server computes the new key, $k'_{<i,j>} = f(k_{<i,j>} \oplus k'_{<i,j*2+1>})$, encrypts and sends it to the users.

(ii) i is not a prime number. $k_{<i,j>}$ has two children nodes, $k_{<j,l_1>}$ and $k_{<i/j,l_2>}$. If all a_i and e_p are not common factors of j, the new key is $k'_{<i,j>} = f(k_{<i,j>} \oplus k_{<j,l_1>})$ (l_1=0 when j is a prime number, otherwise l_1!=-1,0). If all a_i and e_p are not common factors of i/j, the new key is $k'_{<i,j>} = f(k_{<i,j>} \oplus k_{<i/j,l_2>})$ (l_2=0 when i/j is a prime number, otherwise l_2!=-1,0). When the two children nodes are labeled, the server chooses a new child node key to compute $k'_{<i,j>}$.

Case 2: As shown in Fig. 16, the new key is $k'_{<i,j>} = f(k_{<i,j>})$.

Case 3: As shown in Fig. 17, the *k*-node is newly created, the new key is $k'_{<i,j>} = f(k_{<i,j*2+1>} \oplus k_{<i,0>})$.

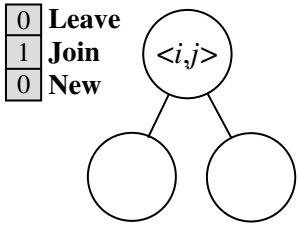

Fig. 15 $k_{<i,j>}$ is labeled as LEAVE

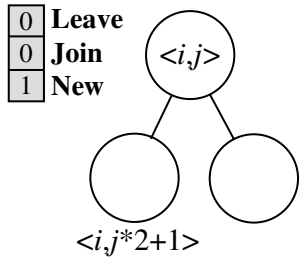

Fig. 16 $k_{<i,j>}$ is labeled as JOIN

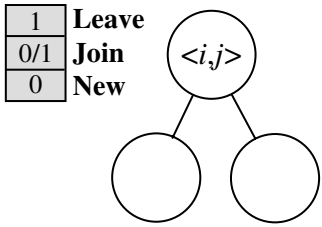

Fig. 17 $k_{<i,j>}$ is labeled as NEW

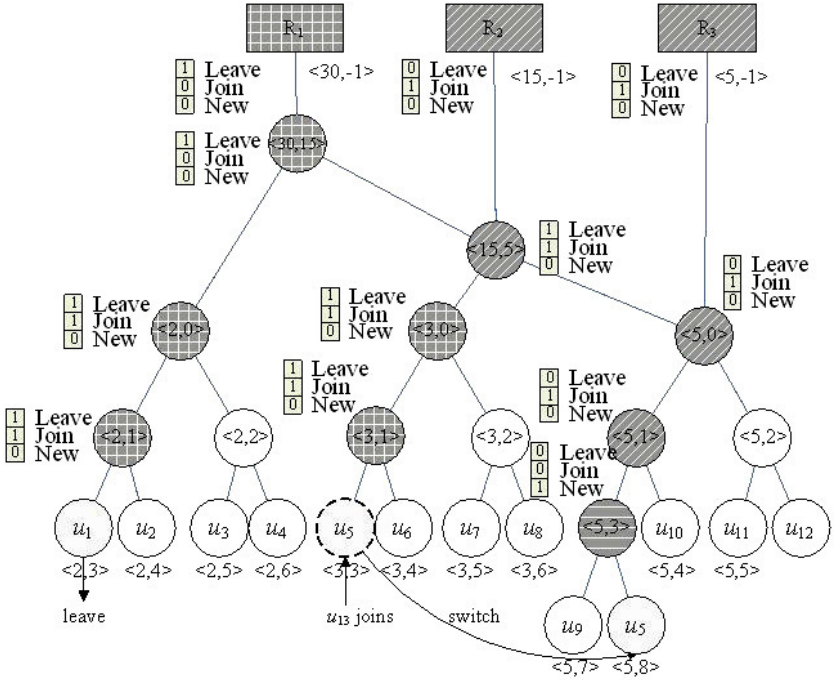

Fig. 18 A batch update example

Fig. 18 shows an example of the batch update operation. During a rekeying period of time, u_1 leaves the group, u_{13} joins S_2, and u_5 switches from S_2 to S_3. The server removes the *u*-node $u_{<2,3>}$, broadcasts their IDs and the maximum IDs of *k*-nodes in some SGs which some users want to

join and switch to, $<2, 3>_L$, $<3, 3>_{SL_1}$, $<5, 8>_{SJ_1}$, $<3, 3>_J$, and <3, 2>, <5, 3>. The server and the users label the *k*-nodes. The users can compute the new keys according to the above method.

$$k'_{<5,\,1>}=f(k_{<5,\,1>}),\quad k'_{<5,\,0>}=f(k_{<5,\,0>}),\quad k'_{<5,\,-1>}=f(k_{<5,\,-1>}),$$
$$k'_{<15,\,5>}=f(k_{<15,\,5>}),\quad k'_{<15,\,-1>}=f(k_{<15,\,-1>}),$$
$$k'_{<5,\,3>}=f(k_{<5,\,7>}\oplus k_{<5,\,0>}),$$
$$k'_{<3,\,1>}=f(k_{<3,\,1>}\oplus k_{<3,\,4>}),\quad k'_{<3,\,0>}=f(k_{<3,\,0>}\oplus k_{<3,\,2>}),$$
$$k'_{<2,\,1>}=f(k_{<2,\,1>}\oplus k_{<2,\,4>}),\quad k'_{<2,\,0>}=f(k_{<2,\,0>}\oplus k_{<2,\,2>}),$$
$$k'_{<30,\,15>}=f(k_{<30,\,15>}\oplus k'_{<2,\,0>}).$$

To the SK, $k_{<30,\,-1>}$, the server computes the new key,

$$k'_{<30,\,-1>}=f(k_{<30,\,-1>}\oplus k'_{<30,\,15>}).$$

Finally, the server encrypts the new keys that some users can not compute by themselves and sends them to these users.

$$s\rightarrow u_6\colon \{k'_{<3,\,0>}\}_{k'_{<3,\,1>}},\quad s\rightarrow u_2\colon \{k'_{<2,\,0>}, k'_{<30,\,15>}\}_{k'_{<2,\,1>}},$$
$$s\rightarrow u_5-u_{13}\colon \{k'_{<30,\,15>}\}_{k'_{<15,\,5>}},\ s\rightarrow u_2-u_{13}\colon \{k'_{<30,\,-1>}\}_{k'_{<30,\,15>}}.$$

5. Conclusion

In this paper, we investigated the issues of key management in support of multi-privileged group communications and proposed an ID-based hierarchical key graph scheme to manage multi-privileged group communications. According to the relationship between children nodes and parent node as well as the relationship between data streams and SGs, each node on key graph is assigned a unique ID, in order for the node to deduce the IDs of his parent node and ancestor nodes. The server only needs to broadcast the IDs of joining user and leaving user, the old users in the group know which nodes they hold should be updated. The new key value is computed by a one-way function, the server doesn't need to send rekeying messages when a user joins. When a user leaves, part of users also can compute the new keys by themselves. No matter joining/leaving or switching, the users in the group can deduce some updated keys, thus the goal of reducing the rekeying overhead at the server can be achieved.

Currently the proposed scheme uses only binary tree and can't flexibly deal with formation and decomposition of service groups. We

plan to extend its usage of *k*-ary trees and to propose solutions to the dynamic formation and decomposition of service groups.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China under Grants No. 60503007 and No. 60533040, in part by the research grant NSC 95-2221-E-110-062 from National Science Council, Taiwan, and in part by the Program for New Century Excellent Talents in University (NCET) of the Chinese Ministry of Education.

References

1. W. Trappe, J. Song, R. Poovendran and K. J. R. Liu, *Proceedings of 2001 IEEE International Conference on Acoustics, Speech, and Signal,* (ICASSP, Salt Lake City, 2001), p 1449.
2. S. Rafaeli and D. Hutchison, *ACM Computing Surveys*, 309(2003).
3. A. Perrig, D. Song and D. Tygar, *Proceedings of IEEE Symposium on Security and Privacy*, (IEEE, Oakland, 2001), p. 247.
4. D. McGrew and A. Sherman, *Technical Report 0755*, (1998).
5. D. M. Wallner, E. J. Harder and R. C. Agee, Internet Draft Report, Filename: draft-wallner-key-arch -01.txt, (1998).
6. G. H. Chiou and W. T. Chen, *IEEE Transactions on Software Engineering*, (IEEE TSE, 1989), p. 929.
7. S. Banerjee and B. Bhattacharjee, IEEE Journal on Selected Areas in Communications, Special Issue on Network Support for Group Communication, 1511(2002).
8. S. Mittra, *Computer Communication Review*, (ACM Press, New York, 1997), p. 277.
9. Y. Sun and K. J. R. Liu, *Proceedings of IEEE INFOCOM 2004*, (INFOCOM, Hong Kong, 2004), p. 1296.
10. Q. Zhang and Y. Wang, *Proceedings of Global Telecommunications Conference*, (GLOBECOM, Dallas, 2004), p. 2067.
11. A. M. Eskicioglu, S. Dexter and E. J. Delp, *Proceedings of SPIE Security and Watermarking of Multimedia Contents*, (SPIE, San Diego, 2003), p. 505.
12. J. C. Lin, P. F. Lai and H. C. Lee, Proceedings of IEEE conference on Local Computer Networks 30th Anniversary, (LCN, Dublin, 2005), p. 336.

13. C. Yuan, B. Zhu, M. Su, X. Wang, S. Li and Y. Zhong, *Proceedings of IEEE International Conference on Image Processing 2003*, (ICIP, Barcelona, 2003), p. I-517-20.
14. C. K. Wong, M. Gouda and S. S. Lam, *Proceedings of ACM SIGCOMM98*, (SIGCOMM, Vancouver, 1998), p. 68.
15. M. Waldvogel, G. Caronni, D. Sun, N. Weiler and B. Plattner, *IEEE Journal on Selected Areas in Communications*, 1614(1999).
16. J. C. Birget, X. Zou, G. Noubir and B. Ramamurthy, *Proceedings of International Conference on Communications*, (ICC, Helsinki, 2001), 229(2001).
17. R. Deng, Y. Wu and D. Ma, *Computer Security in the 21st Century*, 229(2005).
18. D. Ma, Y. Wu, R. Deng and T. Li, Proceedings of 6th International Conference on Information and Communications Security, (ICICS, Malaga, 2004), p.508.
19. R. Li, J. Li and H. Kameda, *Proceedings of ICCNMC*, (ICCNMC, Zhangjiajie, 2005), p.539.
20. X. B. Zhang, S. S. Lam, D. Y. Lee, and Y. R. Yang, *IEEE/ACM Transactions on Networking*, 908(2003).
21. Y. R. Yang, X. S. Li, X. B. Zhang and S. S. Lam, Proceedings of the ACM 2001 conference on applications, technologies, architectures, and protocols for computer communications, (SIGCOMM, San Diego, 2001), 27 (2001).

Chapter 6

Access Control Policy Negotiation for Remote Hot-Deployed Grid Services

Jinpeng Huai and Wei Xue

School of Computer Science and Engineering
Beihang University, Beijing, 100083, China
E-mail: huaijp@buaa.edu.cn

Yunhao Liu and Lionel M. Ni

Department of Computer Science and Engineering
Hong Kong University of Science and Technology, Hong Kong, China
E-mail: liu@cs.ust.hk

Service grid is a widely distributed environment, where service deployers and containers might locate in different autonomous domains. Different from traditional scenarios like J2EE applications, in service grids, the access control policy should not be determined by a deployer or a container only. Existing grid deployment solutions do not address this unique requirement. We introduce a general approach, CROWN.ST, an access control policy negotiation solution on remote hot-deployment for grid services. Based on an access control policy language derived from non-recursive stratified Datalog with constraints, we design the negotiation procedure and three types of meta-policies. We implement a CROWN.ST prototype and evaluate our design through comprehensive experiments.

1. Introduction

Grid computing has been an attractive distributed computing paradigm over wide-area network, enabling resource sharing and collaborating across multiple domains [8, 9]. The research described in this chapter is a

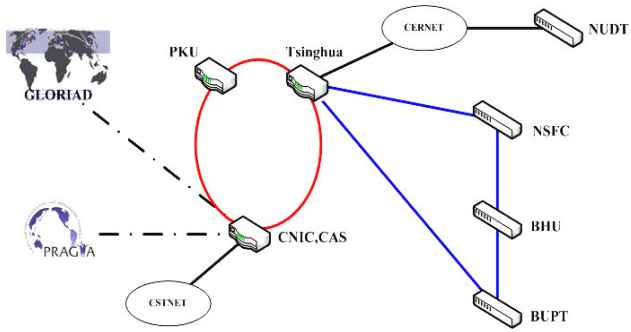

Figure 1 Network topology of CROWN grid

part of a larger project named CROWN (China R&D Environment Over Wide-area Network) [12, 27] , which aims to promote the utilization of valuable resources and cooperation of researchers nationwide and worldwide.

The CROWN project is started in late 2003. Several universities and institutes, such as Tsinghua University, Peking University, Computer Network Information Center of CAS (China Academy of Science) and Beihang University, have joined CROWN as the initiating partners. Till March 2005, CROWN has gathered more than 0.7 Tflops computational resources, 10TB storage resources and many applications ranging from gene comparison to climate pattern prediction. Figure 1 illustrates CROWN Grid topology.

CROWN, as a service grid, with heterogeneous resources wrapped as grid services, can be accessed using standardized protocol, for example, the Simple Object Access Protocol (SOAP). All resources being wrapped are hidden from grid users.

For the convenience of developers and administrators of grid applications, we develop *service container* to support the maintenance and management of grid services. Each service must be deployed into some target service container before it is accessible to users. We call the one which deploys a grid service as the *deployer* of the service.

As a grid is often a widely distributed environment, service deployers might be located far away from service containers. Hence, we proposed a mechanism for remote and hot grid service deployment [25]. Due to the dynamic nature of grids, remote and hot service deployment is quite

often in CROWN. In traditional scenarios such as J2EE application deployments, a deployer is absolutely trusted by an application server (after authentication and authorization) and can determine the security policy of the application by itself. In grid environments, the access control policy of a grid service cannot be determined by the deployer or the container only. Existing grid deployment solutions do not address such a unique requirement [2, 11].

In this chapter, we introduce a general approach, CROWN.ST, an access control policy negotiation solution for remote and hot-deployment of grid services. As non-recursive stratified Datalog with constraints is suitable to provide logical semantics for the core parts of the eXtensible Access Control Markup Language (XACML)[10], we propose an access control policy language based on it. We then design a negotiation procedure and meta-policies for the creation of proposals, conflict resolution, and policy validation during negotiations. Thus, deployers and containers are able to specify detailed strategies, automating the policy negotiation procedure and guaranteeing their own concerns are respected. We implement a CROWN.ST prototype and evaluate our design by comprehensive experiments. The preliminary results show that our approach is feasible and effective.

The rest of the chapter is organized as follows. Section 2 describes the background and related works. Section 3 briefly introduces our access control policy language and related notions. Section 4 describes the policy negotiation procedure and meta-policies. Section 5 introduces the CROWN.ST prototype implementation. We analyze the complexity of the negotiation procedure in Section 6 and show the experimental results in Section 7. Section 8 concludes our work and presents future directions.

2. Background and Related Works

Remote deployment of applications has been investigated by researchers for a long time. Several popular fundamental software platforms, as well as those in the grid community, have remote deployment mechanisms built in [2, 11]. However, up to now, none of them takes access control policy negotiation into account.

2.1. *Access Control*

The remotely deployed grid service wraps raw resources supplied by the container and exposes higher level service interfaces to the end users. Normally, neither the deployer nor the container owns both the grid service and the raw resources. As a nature result, the access control policy for the grid service should be jointly determined by both parties, which is a unique requirement in grid systems, and is not supported by existing access control solutions for grid. For example, PRIMA[19] allows authoritative users to delegate fine-grained privileges to other subjects. The Community Authorization Service (CAS)[22] allows sites to delegate management of a subset of their policy space to the VO. Akenti[26] allows multiple stakeholders to create policy assertions. However, above solutions expect the resources have clear ownership and there exists unique source of authority that has the ultimate authority. They do not support policy negotiation.

Beyond grid scenarios, some automatic approaches for security policy reconciliation or negotiation have been proposed. Patrick McDaniel et al identify an efficient algorithm for two-policy reconciliation and suggest efficient heuristics for the detection and resolution of intractable reconciliation[21]. But their target application scenarios are mainly secure group communications, and the policy language they proposed, i.e. Ismene, cannot depict detailed authorization policies. Furthermore, their reconciliation algorithm takes the conservative approach, which is essentially denials take precedence, to synthesize all access control policies and cannot choose different approaches dynamically. Oppositely, our language can be used to depict detailed policies, and meta-policies are used to select different combining algorithms and validation queries according to both parties' requirements.

H. Khurana and V. D. Gligor propose a formal state-transition model for access control policy negotiation[15]. They cast the negotiation problem as one of satisfying diverse coalition-member objectives and a specified set of negotiation constraints. Such a model is based solely on Role-Based Access Control (RBAC) model and does not provide automatic mechanisms for the negotiation. Vijay G. Bharadwaj et al propose a mathematical framework based on semiring-based CSPs

(SCSPs) for automatic access control policy negotiation among autonomous domains[3]. But we think that the guidance provided by constraints is not enough to bring out practical solutions for automatic negotiation. We believe that agents for all parties should have prepared rules for negotiation in order to get concrete policies. Instead, we use rule-based meta-policies to determine the policy proposals, combining algorithms and validation queries in which different kinds of constraints can be expressed.

2.2. *Policy Language*

In this subsection, we briefly compare our access control policy language with those policy languages mentioned in literatures.

For every system with security concerns, it is critical to assure that the access control policies of a system are coherent and meet the requirements of stakeholders. So, many access control systems use formal languages or languages with formal semantics to specify their policies. Our access control policy language is based on non-recursive stratified Datalog with constraints and can be used to define the formal semantics of XACML, which is one of the design principles for the language.

Compared with other access control policy languages with logical foundation, the advantage of our language is twofold.

First, our language is non-monotonic. In another words, conclusions drawn before may become wrong when new facts are considered. Many popular trust management languages, such as RT (Role-based Trust-management) framework[18, 16], SD3 (Secure Dynamically Distribute Datalog)[14], and Binder[6], are monotonic, or have monotonic subset, such as Delegation Logic[17]. The hypothesis of monotonicity simplifies the distributed management of policies (through delegation, for example), while it fail to support explicit negation. However, explicit negation is necessary for resolving potential conflicts between proposals of the deployers and containers. Our design addressed this issue. Secondly, we propose to use constructive negation[24] as the operational model for negation when analyzing and validating policies as constraint logic programs. In this way, queries can get constructive answers even when

meeting non-ground negative goals during the evaluation. If the negotiation fails, these answers can be returned to the negotiation partner as hints for the next round of negotiation.

3. Access Control Policy Language

In this section, we introduce the basic constructs of our access control policy language, which is designed based on the constraint logic programming paradigm. We refer readers to the surveys [5, 13] for details about basic logical terms such as facts, rules, monotonic, stratification, non-recursive and constraints.

3.1. *Notations*

Our policy language is a multi-sorted logic language created from the following alphabet.

Constant Symbols: We regard the sets of subjects, resources, actions and environments as data types and separate them from basic data types such as integer and float. Accordingly, we use constant symbols begin with lowercase letter, such as `sub_1`, `res_1`, `act_1` and `env_1` to denote elements of these types respectively.

Variable Symbols: We use symbols in forms of `Sub`, `Res`, `Act` and `Env` as variable symbols ranging over the sets of subject, resources, actions and environments respectively. For simplicity, variable symbols ranging over basic data types are not classified accordingly in this work. In the following, we refer to constant symbols and variable symbols of type X as "X terms". For example, `sub_1` is a subject term.

Predicate Symbols: Three types of predicate symbols are considered.

(1) Primitive Constraint Predicate Symbols. Constraints are special relations upon terms of the corresponding constraint domain. A primitive constraint takes the form $r(t_1,...t_n)$ where r is an n-ary primitive constraint predicate symbol, and t_i s are terms. A constraint is the conjunction of several primitive constraints.

(2) Built-in Predicate Symbols, including

Ternary predicate symbols, `sub_att`*,* `res_att`*,* `act_att` *and* `env_att`. They represent the attributes of subject, resource, action and

environment respectively. To illustrate with `sub_att`, the first argument is a subject term, and the second is a string term identifying an attribute, while the third is a term of some constraint domain. *Ternary predicate symbols, `sub_att`, `res_att`, `act_att` and `env_att`.* They represent the attributes of subject, resource, action and environment respectively. To illustrate with `sub_att`, the first argument is a subject term, and the second is a string term identifying an attribute, while the third is a term of some constraint domain.

4-ary predicate symbols, in the form of `permit_i`, where `i` is a unique ordinal number used to stratify the resulting logic program. The first argument of `permit_i` is a subject term, the second is a resource term, and the third is an action term, while the fourth is an environment term. The predicate `permit_i` represents a positive authorization explicitly granted to or implicitly derived for the subject.

4-ary predicate symbols, in the form of `deny_i`, where `i` and arguments are the same as `permit_i`. The predicate `deny_i` represents a negative authorization explicitly granted to or implicitly derived for the subject.

A 4-ary predicate symbol, `permit`, with the same arguments as `permit_i`. The predicate `permit` represents the positive authorization explicitly granted to or implicitly derived for the subject finally.

A 4-ary predicate symbol, `deny`, with the same arguments as `deny_i`. The predicate `deny` represents the negative authorization explicitly granted to or implicitly derived for the subject finally.

3.2. *Definition of Authorization Policies*

According to the above access control policy language, an authorization policy is defined as follows.

Definition 3.1 An **access control policy** is a mapping of 4-tuples (s, r, a, e) consisting of a subject, a resource, an action, and an environment, respectively to the set $\{permit, deny\}$. The policy is specified as a program in non-recursive stratified Datalog with constraint which defines the predicates `permit` and `deny`. In following

discussions, we will use usual terms such as atom, literal when define the logic rules that can be expressed in our access control policy language.

Definition 3.2 A **subject attribute fact** is a rule of the form: $\texttt{sub\_att}(s,id,val) \leftarrow$.

Where s is a subject term, id is a string term identifying an attribute, and val is the value of the attribute.

We define resource attribute facts, action attribute facts and environment attribute facts similarly. All of these facts are called attribute facts.

Attribute facts represent the authorization information related to subjects, resources, actions and environments. They maybe specified in the policy base beforehand, or gathered and specified by the access control system upon user accesses. To make it clearer, an example, E.A.1, is given in appendix.

Definition 3.3 A **basic authorization rule** is a rule of the form:

$\texttt{permit\_i}(s,r,a,e) \leftarrow L_1 \,\&\, ... \,\&\, L_n$. or

$\texttt{deny\_i}(s,r,a,e) \leftarrow L_1 \,\&\, ... \,\&\, L_n$.

where s, r, a, e are subject term, resource term, action term and environment term respectively, and for each $0 < i \leq n$, L_i is either an attribute literal or a primitive constraint literal.

Basic authorization rules are specified by administrators explicitly, or, in our scenario, specified in the policy proposals proposed by the deployers and containers. Each of them represents a special kind of cases where the user access should be explicitly permitted or denied. An example, E.A.2, is given in appendix.

By means of different constraint domains and related complete theories, we can deal with subjects, resources, actions and environments with complex structures using the basic authorization rules. However, there may be conflicts among basic authorization rules. In order to express coherent policies with practical usage, we need the following composition rules.

Definition 3.4 A **composition rule** is of the form:

$\texttt{permit\_j}(s,r,a,e) \leftarrow L_1 \,\&\, ... \,\&\, L_m$. or

$\texttt{deny\_j}(s,r,a,e) \leftarrow L_1 \,\&\, ... \,\&\, L_m.$ or
$\texttt{permit}(s,r,a,e) \leftarrow L_1 \,\&\, ... \,\&\, L_m.$ or
$\texttt{deny}(s,r,a,e) \leftarrow L_1 \,\&\, ... \,\&\, L_m.$

where s, r, a, e are subject term, resource term, action term and environment term respectively, and for each $0 < i \le m$, L_i is either an attribute literal, a primitive constraint literal, a `permit_k` (`deny_k`) or a negative `permit_k` (`deny_k`) literal with lower ordinal number ($k < j$). `permit` (`deny`) should be regarded as `permit_k` (`deny_k`) with highest ordinal number.

Composition rules are used to derive authorizations from basic authorization rules and resolve possible conflicts among lower level rules. By means of composition rules, we can establish a tree of sets of authorization rules. The leaves of this tree are the sets consist of basic authorization rules and attribute facts. The root is a set of composition rules with `permit` and `deny` atoms as heads. Every non-leaf node of the tree is a set of composition rules with `permit_k` and `deny_k` atoms as heads. Thus, each sub-tree represents a consistent sub-policy of the whole authorization policy. This tree can be easily mapped to the hierarchy of policy set, policy, and rules defined in XACML. An example, E.A.3, is given in appendix.

By specifying an access control policy as a logic program in our language, you can depict the access control requirements of many real life applications. The evaluation of an access control policy can be implemented as the execution of corresponding logic program.

Note that the authorization policy specified above should be transformed before analyzing or validating it as a constraint logic program. This is because of the constraint propagation and solving mechanism used by most constraint logic programming systems, which needs the constraint variables appear in the head of logic rules. The transformation procedure is quite straightforward. First, we remove the subject, resource, action and environment variables from the head, and drop the attribute literals in the body of logic rules. Second, we insert appropriate constraint variables into the head literal and add appropriate constraint literals into the body. An example of the transformation, E.A.4, is given in appendix.

Clearly, the transformation is not necessary if we only want to evaluate the policy against concrete attribute facts and get yes/no decision.

4. Negotiation Procedure & Meta-Policies

In CROWN, service deployers and containers are often located in different security domains. As a result, before the access control policy negotiation for the remote hot-deployed a grid service will be considered, an appropriate trust relationship must be established, while the mechanism for trust establishment is out of our discussion scope.

After trust establishment, as shown in Fig. 2, the negotiation procedure of access control policy takes place. WS-Security [1] is used to secure the communications between the two parties. We use dashed line for steps 7 and 8 in Fig. 2 because these two steps may be skipped.

During negotiation, the actions of both parties are controlled by meta-policies. CROWN.ST has three types of meta-policies as follows.

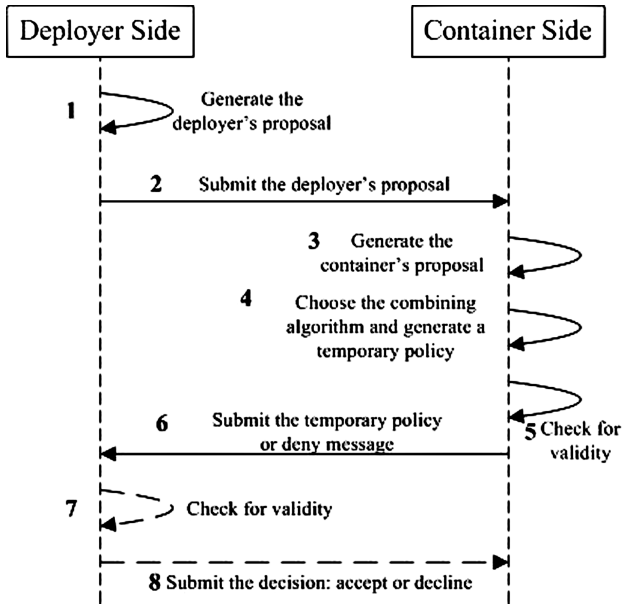

Figure 2 Negotiation of access control policy

(1) *Proposal making meta-policies* are used to dynamically generate policy proposals. They are application-specific and mainly used on the container side. In order to accommodate diverse application requirements, we used rule-based language to specify this kind of meta-policies. Typically, they are specified in accordance with the service level agreements (SLAs) between the deployer and the container, or other collaboration agreements between them.

(2) *Combining algorithm selection meta-policies*. We do not see any single combining algorithm suitable for resolving all possible conflicts in grid environments, so CROWN.ST employs meta-policies on the container side to dynamically select appropriate combining algorithms for synthesis of policy proposals. These meta-policies are also rule-based. The selection criteria in these rules are logical expressions defined for each candidate algorithms in terms of properties of the deployer, the service and the raw resources. Commonly used combining algorithms include deny-overrides, permit-overrides and explicit priority based algorithms.

(3) *Validity checking meta-policies* are used to check the validity of temporary policies. They are rule-based too and specified in company with proposal making meta-policies. Their evaluation results are logical queries consist of constraint literals and a `permit` or `deny` literal, and must be evaluated to true according to the temporary policy before the temporary policy is accepted. The queries with negative `permit` literal and negative `deny` literal correspond to the traditional safety and availability queries respectively. Besides authorization constraints such as separation of duties, the deployer could derive validity checking meta-policies from SLAs to take full advantage of the raw resources provided by the container.

To illustrate the negotiation procedure and meta-policies, we consider the following simple scenario. Alice has a grid service named service1 and want to provide it to her classmate Julius Hibbert. But Alice doesn't own enough resources to host the service herself. She finds a remote container which provides application hosting services and wants to deploy the service on it. The procedure she takes is the following:

Step 1. Alice generates its policy proposal, i.e. rule `permit_1` in E.A.2 which permits Julius Hibbert to access service1.

Step 2. Alice submits the proposal to the negotiation service representing the container.

Step 3. In this scenario, we suppose the container has established some SLA with Alice beforehand. The negotiation service authenticates Alice and generates its own policy proposal according to the SLA between them. The resulting proposal is the rule `deny_2` in E.A.2, which denies user access when CPU usage exceeds 50%. It's worthy to note that these two proposals concern different parts of the policy. While the deployer's proposal concerns who can access the grid service, the negotiation service's proposal concerns how much raw resources can be used by the grid service.

Step 4. We suppose the negotiation service has two candidate combining algorithms in this scenario, permit-overrides and deny-overrides, which are provided for container owner and other remote users respectively. So, deny-overrides algorithm is selected for Alice. The resulting temporary policy is illustrated in E.A.3.

Step 5. Because deny-overrides algorithm is selected, the validity checking on the container side could be omitted safely.

Step 6. The negotiation service returns the temporary policy to the deployer.

Step 7. The deployer checks the validity of the temporary policy and makes a decision, i.e., accept it and continue the deployment, or decline it and terminate the deployment. In this scenario, the validation queries are also generated according to the SLA between Alice and the container in order to take full advantage of the raw resources provided by the container. The resulting query is

$$Cpu\_usage <= 50,$$
$$permit(\text{"service1"},\text{"Julius Hibbert"},Cpu\_usage).$$

It is evaluated to true.

Step 8. The deployer submits its acceptance to the negotiation service and continues the real deployment.

5. CROWN.ST Prototype Implementation

To implement our access control policy language, we use an open source constraint logic programming system, YAP[4], as the underlying engine.

Our current prototype only supports linear arithmetic constraints over rational number. Many authorization information used in grid environments (e.g., user identifier, time, storage space, cpu frequency) can be treated as rational numbers, so we could express real life policies over this constraint domain. For grid users' convenience, we implement a translation tool for translating access control policy specified with simplified XACML into constraint logic program written in our access control policy language.

Because current constraint logic programming systems lack support for general constructive negation (most implementations of constructive negation are specially designed for the Herbrand domain), we developed a tool for translating policies in our language to logically equivalent constraint logic program without negation, which will be evaluated using YAP. In this way, validation queries could get constructive answers even when validation fails. These answers may be used as hints for the negotiation partners to accelerate the next round of negotiation.

To implement meta-policies, we used a general, efficient and open source rule engine, Drools[23], which is based on Rete algorithm [7].

Besides above mentioned tools and libraries, we integrate the functionalities used in negotiation with other components of CROWN. On the deployer-side, we developed a GUI tool for negotiation, which prompts users for necessary decisions such as parameter choosing. The negotiation progress and temporary policy are visually shown to users. The user could use the translation and analysis function provided by the tool to translate and validate temp policies.

On the container-side, the functionalities used in negotiation are implemented as a standalone grid service, which is called negotiation service. We deploy this service in each CROWN node. Before deploying a service, the deployer is redirected to the negotiation service first. The negotiation service will generate a container side policy proposal and combine it with deployer's proposal, then validate the temporary policy and return validation result accordingly.

In order to reduce the cost spent on maintaining states for long-lived negotiations, the negotiation service signs and timestamps the temporary policy and then halts the negotiation procedure if the validation on the

deployer-side is likely time consuming. The deployer can validate the temporary policy offline and then resume the remote deployment.

6. Complexity Analysis

Besides the cost of network transfer and message (de)serialization, the complexity of access control policy negotiation mainly comes from the evaluation and enforcement of meta-policies.

The three kinds of meta-policies are rule based, so their evaluations are tractable (particularly, the Drools engine can achieve linear complexity w.r.t. the meta-policy size after compilation). However, the validation queries derived from validation meta-policies must also be evaluated, which is intractable in general. Because that our policy language supports explicit negation, the independence of negated constraints property (INC)[20] does not hold on our constraint domains. Particularly, testing the satisfiability of a conjunction of constraints and negated constraints can not be reduced to a series of tests involving a single negated constraint. As a result, the worst-case complexity of validation query evaluation is at least co-NP-hard w.r.t. the size of the temporary policy in general. Besides this, the cost of testing the satisfiability of each constraint may be not neglectable. For example, the complexity of constraint solving over discrete finite domain is NP-hard in general. For our prototype, the complexity of solving linear arithmetic equations/inequations is polynomial w.r.t. the variable number and equaitons/inequations size, so its impact is relatively small.

Despite the high worst-case complexity mentioned above, the access control policy negotiations between grid service deployers and containers are not too complex according to our experience. Firstly, the basic authorization rules in these policies usually involve only few (for example, no more than 3) primitive constraints with few (no more than 3) variables, because the deployer and container owner usually concern with different authorization factors. For example, the deployers usually concern with the grid service user's identity and other properties. In contrast, the container owners usually concern with factors about the raw resources, such as CPU usage, storage size and network speed. Apparently, this will keep the explosion of sub-goals and the cost of

constraint solving grow relatively slow when the rule number increase, as will also shown in Section 7.

Second, many safety properties can be achieved through careful proposal making and combination instead of temporary policy validation. For example, the container owner's meta-policies can choose deny-overrides combining algorithm or specify high priorities for deny rules in its proposal in order to assure the final policy will not abuse the raw resources.

Third, the validity checking meta-policies used in real scenarios usually generate validation queries with quite a number of constraint literals, which could be used to reduce the search space of evaluation effectively.

7. Performance Evaluation

We successfully deploy CROWN.ST prototype in CROWN Grid environment. To evaluate its performance, we conduct a series of experiments. The negotiation service (with underlying container) is deployed on cluster nodes with Intel Xeon 2.8GHz CPU, 2G RAM, RedHat Linux EL3.0 and 100M bps Internet connection. On the deployer-side, we use a notebook with 1.6GHz CPU, 512M RAM, Debian Linux with kernel 2.6.8 and 100M bps Internet connection. To make sure the measurements are accurate, no other tasks are running on cluster nodes and the notebook, except the necessary CROWN middleware. If not explicitly specified otherwise, each experiment takes 10 run and we plot the average.

Concurrent thread numbers and the sizes of temporary policies are taken as parameters. The size of a temporary policy is further characterized using 4 parameters. (1)The number of primitive constraints in each basic authorization rule, denoted by *PC* in the following figures; (2) The size of a primitive constraint, i.e. the number of variables appearing in the primitive constraint, which is denoted by *PCS*; (3) The number of variables appear in the temporary policy, denoted by *VAR*; (4) The number of basic authorization rules appear in the temporary policy, denoted by *R*. We take the time used by the whole negotiation procedure

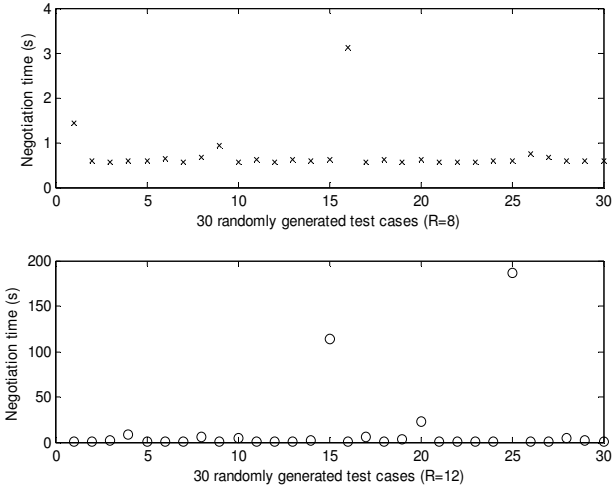

Figure 3 Negotiation time for 60 randomly generated test cases with $PC = 3, PCS = 3, VAR = 8$

as the evaluation metric, which excludes the time used for user interactions and digital encryption/decryption.

In our first experiment, we randomly generate four groups of test cases. Each group consists of 30 test cases generated with the same

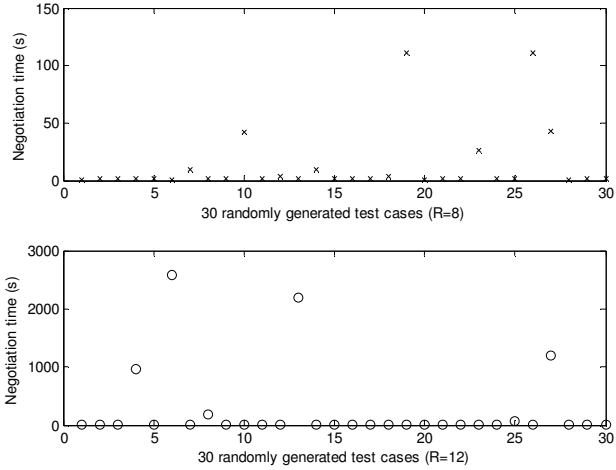

Figure 4 Negotiation time for 60 randomly generated test cases with $PC = 5, PCS = 5, VAR = 8$

policy size parameters. We further separate the four groups into two subgroups according to *PC* and *PCS* (*VAR* is fixed in this experiment because its impact is relatively small). Figures 3 and 4 show the negotiation time used by these test cases. Each point represents the negotiation time used by one test case.

From these two figures, we can see that the negotiation time used by test cases with the same size parameters may differ significantly. This is because that the application-specific policy and query structures have an important impact on the negotiation time. As mentioned in Section 6, the worst-case complexity is at least co-NP hard, but not all cases are the worst cases. Contrarily, most of the cases are simple according to our experiences.

The peak value of 30 randomly generated test cases can be regarded as representing the worst-case. From these two figures we can see that the worst-case cost increases relatively slow against *R* when *PC* and *PCS* are relatively small, which are the cases for most grid applications. As a result, the approach presented in this chapter can be employed in real grid scenarios.

Figure 5 plots the average negotiation time against the number of concurrent requests. The policy size parameters of the test case is *PC=5*, *PCS=3*, *VAR=8*, *R=8*. As we can see in this figure, the average negotiation time increases linearly with the increase of concurrent requests.

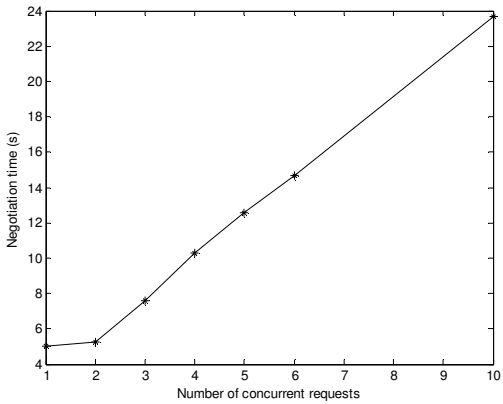

Figure 5 Negotiation time vs. the number of concurrent requests

8. Conclusions and Future Works

We propose a general approach for access control policy negotiation during remote hot-deployment of grid services, which is an important requirement for service grids like CROWN. We define an access control policy language based on non-recursive stratified Datalog with constraints for grid services. The language can be used to specify and analyze practical access control policies for real life applications. Based on this language, we design a negotiation procedure, which dynamically and automatically determine the final access control policy for the grid service being deployed.

We successfully implement a CROWN.ST prototype, which has been deployed in our CROWN Grid. We further evaluate CROWN.ST through comprehensive experiments. Due to the page limit, we only show the representative results.

CROWN is an actively ongoing project. Therefore, our solutions for secure, remote and hot deployment of grid services will be further extended and improved in future versions of CROWN. Future work will lead into several directions. First, we will improve the performance of the analyzing algorithm and extend CROWN.ST to support more constraint domains. Second, related topics such as the integration of trust negotiation, policy synthesis and service composition are going to be explored and implemented in CROWN. We believe these are the key technologies for better collaboration in service grids.

References

1. B. Atkinson and G. Della-Libera, Web Services Security Version 1.0, http://www-106.ibm.com/developerworks/webservices/library/ws-secure/
2. F. Baude, D. Caromel, F. Huet, L. Mestre, and J. Vayssiere, "Interactive and Descriptor-based Deployment of Object-Oriented Grid Applications," in Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, 2002.
3. V. G. Bharadwaj and J. S. Baras, "Towards Automated Negotiation of Access Control Policies," in Proceedings of IEEE 4th International Workshop on Policy for Distributed Systems and Networks, 2003.

4. F. Cornelli, E. Damiani, S. D. C. d. Vimercati, S. Paraboschi, and P. Samarati, "Choosing Reputable Servents in a P2P Network," in Proceedings of the 11th international conference on World Wide Web(WWW'02), 2002.
5. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov, "Complexity and Expressive Power of Logic Programming," ACM Computing Surveys, vol. 33, pp. 374-425, 2001.
6. J. DeTreville, "Binder, a logic-based security language," in Proceedings of 2002 IEEE Symposium on Security and Privacy, 2002.
7. C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," Artificial Intelligence, vol. 19, pp. 17-37, 1982.
8. I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," Intl. Journal of Supercomputing Applications, vol. 11, pp. 115-129, 1997.
9. I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organization," The International Journal of High Performance Computing Applications, 2001.
10. S. Godik and T. Moses, eXtensible Access Control Markup Language Version 2.0, working draft 12, http://www.docs.oasis-open.org/xacml/xacml-core-spec-2.0-wd-12.pdf
11. W. Goscinski and D. Abramson, "Distributed Ant: A System to Support Application Deployment in the Grid," in Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing, 2004.
12. J. Huai, Y. Zhang, X. Li, and Y. Liu, "Distributed Access Control in CROWN Groups," in Proceedings of International Conference on Parallel Processing (ICPP), 2005.
13. J. Jaffar and M. J. Maher, "Constraint Logic Programming: A Survey," Journal of Logic Programming, vol. 19/20, pp. 503-581, 1994.
14. T. Jim, "SD3: A trust management system with certified evaluation," in Proceedings of 2001 IEEE Symposium on Security and Privacy, 2001.
15. H. Khurana and V. D. Gligor, "A Model for Access Negotiations in Dynamic Coalitions," in Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (ICE'04), 2004.
16. N. Li, J. C. Mitchell, and W. H. Winsborough, "Design of a role-based trust management framework," in Proceedings of the 2002 IEEE Symposium on Security and Privacy, 2002.
17. N. Li, B. N. Grosof, and J. Feigenbaum, "Delegation Logic: A Logic-based Approach to Distributed Authorization," ACM Transactions on Information and System Security (TISSEC), vol. 6, pp. 128-171, 2003.
18. N. Li and J. C. Mitchell, "Datalog with constraints: A foundation for trust management languages," in Proceedings of the 15th International Symposium on Practical Aspects of Declarative Languages, 2003.
19. M. Lorch, D. Adams, D. Kafura, M. Koneni, A. Rathi, and S. Shah, "The PRIMA System for Privilege Management, Authorization and Enforcement in Grid

Environments," in Proceedings of The 4th International Workshop on Grid Computing (Grid 2003), 2003.
20. M. J. Maher, "Adding Constraints to Logic-based Formalisms," in The Logic Programming Paradigm: a 25 Years Perspective, Artificial Intelligence Series, V. M. K.R.Apt, M. Truszczynski and D.S. Warren, Ed.: Springer-Verlag, 1999, pp. 313-331.
21. P. McDaniel and A. Prakash, "Methods and limitations of security policy reconciliation," in Proceedings of IEEE Symposium on Security and Privacy, 2002.
22. L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke, "A Community Authorization Service for Group Collaboration," in Proceedings of IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2002.
23. N. A. Rupp, The Logic of the Bottom Line: An Introduction to The Drools Project, http://www.theserverside.com/articles/article.tss?l=Drools
24. P. J. Stuckey, "Negation and Constraint Logic Programming," Information and Computation, vol. 118, pp. 12-33, 1995.
25. H. Sun, Y. Zhu, C. Hu, J. Huai, Y. Liu, and J. Li, "Early Experience of Remote and Hot Service Deployment with Trustworthiness in CROWN Grid," in Proceedings of 6th ACM International Workshop on Advanced Parallel Processing Technologies, 2005.
26. M. R. Thompson and S. Mudumbai, "Certificate-based Authorization Policy in a PKI Environment," ACM Transactions on Information and System Security (TISSEC), vol. 6, pp. 566-588, 2003.
27. Y. Zhang, J. Huai, Y. Liu, L. Lin, and B. Yang, "A Framework to Provide Trust and Incentive in CROWN Grid for Dynamic Resource Management," in Proceedings of IEEE ICCCN, 2006.

Appendix

E.A.1 Consider the following attribute facts:

$$\texttt{sub\_att}(sub\_1, \text{"subject-id"}, \text{"Julius Hibbert"}) \leftarrow .$$
$$\texttt{env\_att}(env\_1, \text{"current-date"}, \text{"2004}-12-25\text{"}) \leftarrow .$$

The first fact states that the subject identifier of sub_1 is Julius Hibbert. This information maybe gathered by the access control system after verifying the signature of the user on the requesting message. The second fact states that the current date is 2004-12-25. This information maybe gathered by the system from local time server.

E.A.2 Consider the following basic authorization rules:

$$\texttt{permit\_1}(Sub, Res, Act, Env) \leftarrow$$
$$\texttt{res\_att}(Res, \text{"resource-id"}, \text{"service1"}),$$
$$\texttt{sub\_att}(Sub, \text{"subject-id"}, \text{"Julius Hibbert"}).$$
$$\texttt{deny\_2}(Sub, Res, Act, Env) \leftarrow$$
$$\texttt{env\_att}(Env, \text{"cpu-usage"}, X), X > 50.$$

The first rule states that the user Julius Hibbert can access service1 at any circumstance. The second rule states that nobody can access any service if the cpu-usage exceeds 50%.

E.A.3 There is a conflict between the two rules in example E.A.2, we can use the following composition rules to resolve it.

$$\texttt{permit}(Sub, Res, Act, Env) \leftarrow$$
$$\texttt{permit\_1}(Sub, Res, Act, Env),$$
$$\text{not } \texttt{deny\_2}(Sub, Res, Act, Env).$$
$$\texttt{deny}(Sub, Res, Act, Env) \leftarrow$$
$$\texttt{deny\_2}(Sub, Res, Act, Env).$$

These composition rules implement so called "denials take precedence" which is corresponding to the deny-overrides combining algorithm defined in XACML.

Many useful conflict resolution approaches based on explicit or implicit precedence can be implemented with our composition rules. Besides this, to assure the specification completeness of access control policy, we can further include some default authorization rules. For example, we can include the following default composition rule

$$\texttt{deny}(Sub, Res, Act, Env) \leftarrow$$
$$\text{not } \texttt{permit\_1}(Sub, Res, Act, Env).$$

to assure that "undefined" cases are regarded as "deny".

E.A.4 The rules in example E.A.2 and E.A.3 can be transformed to the following rules before validating.

$$\texttt{permit\_1}(Resource\_id, Subject\_id) \leftarrow$$
$$Resource\_id = \text{"service1"},$$
$$Subject\_id = \text{"Julius Hibbert"}.$$

$$\texttt{deny\_2}(Cpu\_usage) \leftarrow Cpu\_usage > 50.$$

$$\texttt{permit}(Resource\_id, Subject\_id, Cpu\_usage) \leftarrow$$
$$\texttt{permit\_1}(Resource\_id, Subject\_id),$$
$$\text{not } \texttt{deny\_2}(Cpu\_usage).$$

$$\texttt{deny}(Cpu\_usage) \leftarrow \texttt{deny\_2}(Cpu\_usage).$$

$$\texttt{deny}(Resource\_id, Subject\_id) \leftarrow$$
$$\text{not } \texttt{permit\_1}(Resource\_id, Subject\_id).$$

$Resource\_id$, $Subject\_id$, and Cpu_usage are constraint variables.