International Journal of Innovative Computing, Information and Control
Volume **8**, Number **11**, November **2012**

 ISSN 1349-4198
pp. 7969–7981

TOWARDS FORMAL SECURITY ANALYSIS OF DECENTRALIZED INFORMATION FLOW CONTROL POLICIES

Zhi Yang[1,2], Lihua Yin[1,3], Shuyuan Jin[1,*] and MiYi Duan[1,3]

[1]Institute of Computing Technology
Chinese Academy of Sciences
No. 6, Kexueyuan South Rd., Zhongguancun, Haidian Dist., Beijing 100190, P. R. China
[*]Corresponding author: jinshuyuan@ict.ac.cn
zynoah@gmail.com; yinlh@hit.edu.cn; duanmiyi@software.ict.ac.cn

[2]Institute of Electronic Technology
Information Engineering University
No. 7, Kexue Rd., Jinshui District, Zhengzhou 450004, P. R. China

[3]National Engineering Laboratory for Information Security Technologies
Beijing 100190, P. R. China

Received July 2011; revised December 2011

Abstract. *Decentralized information flow control (DIFC) is a key innovation of traditional information flow control (IFC). Compared with IFC, DIFC provides new features including decentralized declassification, taint-tracking, and privilege-transferring. These characteristics also make DIFC able to achieve more fine-grained security goals. However, the flexibility of DIFC also presents challenges to its policy verification which existing approaches have not been able to effectively solve. This paper formalizes the DIFC's policy verification problem, and uses Computational Tree Logic formulae to express fine-grained security goals. This paper also proves that the DIFC's policy verification problem is NP-complete, and discusses the main factors resulting in its high computational complexity. Further, a model checking approach is proposed to realize DIFC's policy verification. Experimental results show that our proposed approach is effective.*
Keywords: Access control, Formal verification, NP-complete, Model checking

1. **Introduction.** Access control is one of the most fundamental and pervasive security mechanisms used in information systems [1,2]. As a main access control technology, information flow control (IFC) enforces centralized mandatory access control over operating systems. IFC can effectively prevent bugs or malicious software from destroying the secrecy and integrity of systems. However, traditional IFC is found too restrictive and simple to build any practical systems [3-5].

Decentralized Information Flow Control (DIFC) [3-8] is a key innovation of IFC. Compared with IFC, the DIFC mechanism provides new features of decentralized declassification, taint-tracking, and privilege-transferring. These characteristics make DIFC more applicable to the control of information flows in systems. Specially, the DIFC decentralized declassification mechanism permits processes to declassify their own data. In contrast, IFC only permits a universally trusted process for the declassification of data. This mechanism guarantees that a DIFC process has the ability to only weaken its own policies rather than endanger the data that it does not own. The DIFC taint-tracking mechanism guarantees information flow tracking by recording process label changes either implicitly [4] or explicitly [5,6]. This mechanism reflects the principle of least privilege. The DIFC privilege-transferring mechanism permits processes to transfer a portion of

their label adjustment capabilities to other processes. Thus, the information propagation control becomes more flexible.

This paper aims to verify DIFC's policy, which currently represents a large problem. Policy verification determines whether DIFC systems are able to achieve their security goals. We recognize that a mistake in policy implementation often results in a process having more privileges than it should have. Undesirable right leakage resulting from these privileges can threaten system security seriously. However, the flexibility of DIFC challenges its policy verification for two reasons: 1) the flexibility of DIFC may cause the informational flow graph of a DIFC system to be highly complex [9-11], and resulting attempts to analyze such systems can be difficult; and 2) the flexibility of DIFC allows it to express fine-grained security goals. As such, approaches to DIFC's policy verification should involve the verification of fine-grained security goals.

In the field of DIFC system policy verification, the verified security goals can be categorized into two types: coarse-grained and fine-grained goals. The coarse-grained goal is expressed by the existence of an information flow path while the fine-grained goal is expressed by the constraints and correlations between informational flow paths. Most existing approaches have focused on verifying coarse-grained security goals, with few approaches able to verify fine-grained security goals. For example, Chaudhuri et al. [10] modeled an IFC system in an expressive and decidable extension of Datalog, and then translated questions surrounding the existence of an information flow path between two entities into a Datalog-style query. They applied their approach in an analysis of a DIFC system Asbestos policy. Harris et al. [11] used a relation abstraction method to model DIFC programs, using two kinds of first-order logic formulae to express their security properties. These two kinds of formulae only asserted the existence or nonexistence of an information flow path between two program points. Harris et al. [12] also studied how to construct DIFC applications automatically according to information flow reachability policies. Krohn and Tromer [7] used process algebra CSP to verify the nonexistence of information flow paths from a high-level to low-level process, according to the theory of noninterference. Some approaches [13-15] also have been proposed to verify the existence of information flow paths in SELinux system. These approaches are inefficient in verifying whether decentralized declassification, taint-tracking, and privilege-transferring are properly implemented (with implementation expressed by the constraints and correlations between information flow paths). In this paper, we propose an approach that can verify constraints and correlations between information flow paths, thus enabling these fine-grained security goals to be verified.

Moreover, few existing works have analyzed the computational complexity of DIFC's policy verification issues. Complexity analyses can help us evaluate and improve the algorithms to address this problem. Similar analysis on traditional access control models has been done well. For example, Harrison et al. [16] showed that, in general, the problem of security analysis in the access control matrix model is undecidable. S. Jha et al. [17] proved that the problem of security analysis in the administrative role-based access control model (ARBAC97) is PSPACE-complete.

In this paper, we study the policy verification problems of three main DIFC systems: Asbestos [4], HiStar [5], and Flume [6,7]. The key contributions of this paper are as follows:

- We model the policy verification problem in DIFC (referred to as Q-DIFC), and use Computational Tree Logic formulae to express the fine-grained security goals with forms of constraints and correlations between information flow paths. Further, a model checking approach is proposed to solve Q-DIFC. The experimental results in Flume's policy verification demonstrate the high efficiency of our approach.

- We prove that, in general, Q-DIFC is NP-complete. We further study the main factors resulting in high computational complexity of Q-DIFC. To our best knowledge, ours is the first effort to prove that Q-DIFC is NP-complete.

The rest of the paper is organized as follows. Section 2 roughly introduces three main DIFC systems: Asbestos, HiStar, and Flume. Section 3 formalizes Q-DIFC, and proves that Q-DIFC is NP-complete. This section also studies the main factors resulting in a high computational complexity of Q-DIFC. Section 4 proposes a model checking approach to solve Q-DIFC. Section 5 concludes the paper.

2. **Preliminaries.**

We first give a brief overview of DIFC systems. For a more complete description, please see [3-8].

2.1. **Overview of the FLUME system.**

Flume uses *tags* and *labels* to track its information flows within it. In Flume, each tag is generally associated with some category of secrecy or integrity. Let T denote the set of all *tags*. *Labels* are subsets of T. For each process p, Flume maintains a secrecy label S_P, an integrity label I_P, and a capability set C_P. Flume represents privilege using two capabilities per tag. For tag t, the capabilities are denoted as t^+ and t^-. A process p with $t^+ \in C_P$ can add t to its label; similarly, a process$p$ with $t^- \in C_P$ can remove t from its label.

In order to track information flows within a system, Flume regulates label changes and process communications as follows: For a process p, let label L denote either S_P or I_P, and let L' denote p's new label. The change from L to L' is *safe* if and only if:

$L' - L \subseteq C_P^+$ and $L - L' \subseteq C_P^-$, where $C_P^+ = \{t | t^+ \in C_P\}$, $C_P^- = \{t | t^- \in C_P\}$.

The information flow from a process p to a process q is safe if and only if

$S_p - D_p \subseteq S_q \cup D_q$ and $I_p \cup D_p \supseteq I_q - D_q$, where $D_P = \{t | t^+ \in C_P \wedge t^- \in C_P\}$.

In the Flume system, a process p can grant capabilities in C_P to a process q so long as p is allowed to send message to q. p can also subtract some capabilities from C_P when needed. The label of a new process p is the same as the label of the process that spawns p, unless p's S_P, I_P and C_P are specified by p's creator. p's creator can change p's labels into specified labels S_P, I_P and C_P.

2.2. **Overview of the Asbestos system.**

In Asbestos, different information categories are referred to as *handles*. Handle's privileges are represented by *levels*. Levels are defined as members in an ordered set $[\star, 0, 1, 2, 3]$, where $\star$ represents the minimum level and 3 represents the maximum level. Asbestos uses *labels* to describe the mapping from handles to levels as follows: $\forall L_1$, $\forall L_2$, $L_1 \subseteq L_2 \Leftrightarrow \forall h$, $L_1(h) \leq L_2(h)$, where L_1 and L_2 are labels, and h is any handle. The operators $\cup$ and $\cap$ are defined as follows: $(L_1 \cup L_2)(h) = \max(L_1(h), L_2(h))$, $(L_1 \cap L_2)(h) = \min(L_1(h), L_2(h))$. Asbestos decentralizes declassification by using its special $\star$ operation defined as $L^*(h) = \begin{cases} * & \text{if } L(h) = * \\ 3 & \text{otherwise} \end{cases}$.

In the Asbestos system, each process P has two labels, a *send label* P_S and a *receive label* P_R. The send label represents a process's current contamination level, while the receive label represents the maximum level of contamination that a process can accept from other processes. Each process uses communication ports to communicate with each other. For each communication port p, its corresponding *port label* p_R represents the upper limit of the security messages that can be carried by p. A sender process can selectively taint a handle by providing an optional *contamination label* C_S when sending a message. A process with declassification privileges for a handle h can decontaminate other processes' labels with respect to h. This decontamination is facilitated by lowering their send labels (through a *decontaminate-send label* D_S) while raising their receive labels (through a *decontaminate-receive label* D_R).

A process P is allowed to send a message to a process Q on port p, when the following four requirements are satisfied: (1) $P_S \cup C_S \subseteq (Q_R \cup D_R) \cap p_R$; (2) If $D_S(h) < 3$, then $P_S(h) = *$; (3) If $D_R(h) > *$, then $P_S(h) = *$; (4) $D_R \subseteq p_R$. After receiving the message, Q's labels will be tainted by P as $Q_S \leftarrow (Q_S \cap D_S) \cup ((P_S \cup C_S) \cap Q_S^*)$, $Q_R \leftarrow Q_R \cup D_R$.

2.3. **Overview of the HiStar system.** HiStar employs the same labeling policy as Asbestos to track information flow in a system with one exception: it prohibits implicit label adjustments. Since implicit label adjustments are the source of a covert channel problem [4], HiStar allows each process to explicitly rather than implicitly contaminate itself via a system call *self_set_clearance*.

2.4. **Equivalence of Flume schema and Asbestos/HiStar schema.** In this section, we demonstrate that the basic control functions in-place among different DIFC systems are equivalent. The following example shows how to covert a Flume schema A into an Asbestos/Hitar schema B.

For each secrecy tag t in A, we construct a handle h in B. For each process p and pair of t/h, we construct the receive label $LR_h(p)$ and the send label $LS_h(p)$ for h in B, corresponding to the secrecy label $S(p)$ and capabilities $C(p)$ in A. Table 1 describes the mapping from labels in A to labels in B. For simplicity, Table 1 shows the mapping of secrecy protection schema of Flume. Flume's integrity protection sub-model is the dual of Flume's secrecy protection sub-model, and the two can be transformed into the other.

TABLE 1. Mapping from Flume labels to *Asbestos/HiStar* labels

Flume		**Asbestos (Histar)**	
Secrecy label	*Capabilities*	*receive label*	*send label*
$t \notin S(p)$	$\{t^{\pm}\} \notin C(p)$	$LR_h(p) = 0$	$LS_h(p) = 0$
$t \notin S(p)$	$t^+ \in C(p)$, $t^- \notin C(p)$	$LR_h(p) = 1$	$LS_h(p) = 0$
$t \notin S(p)$	$t^{\pm} \in C(p)$	$LR_h(p) = *$	$LS_h(p) = 0$
$t \in S(p)$	$\{t^{\pm}\} \notin C(p)$	$LR_h(p) = 1$	$LS_h(p) = 1$
$t \in S(p)$	$t^{\pm} \in C(p)$	$LR_h(p) = *$	$LS_h(p) = 1$

The mapping of a process p granting t^+ (or $t^{\pm}$) to a process q through inter-process communication in the Flume schema is p adjusting q's receive labels. This adjustment takes place through inter-process communication with the parameter of the decontaminate-receive label D_R satisfying $D_R(h) = 1$ or $D_R(h) = *$ in the Asbestos/HiStar schema. Thus, given a Flume schema, we have the ability to construct an Asbestos/HiStar schema. In the rest of this paper, we narrow our discussion of the Flume system.

3. **Formal Definition and Complexity Analysis of Q-DIFC.** In this section, we first use the theory of infinite state machine and Computational Tree Logic to model Q-DIFC. Then, we move on to prove that Q-DIFC is NP-complete while showing that the high complexity of Q-DIFC is due to the mechanisms of decentralized declassification and taint-tracking.

3.1. **Formal definition of Q-DIFC.** In order to verify whether the security goals of DIFC systems can be achieved, there is a fundamental question that policy verification should answer: "given the current authorization state and the policy specification, will information ever flow from a subject (object) to another subject (object)?" This question surrounds the existence of an information flow path.

However, to answer the question with a simple "yes" or "no" would be inadequate. We need to further verify more fine-grained security properties of the DIFC systems. These

fine-grained properties include the constraints and correlations between information flow paths. For example, the existence of an information flow path depends on the existence of another path; or from a different perspective, the existence of an information flow path depends on the nonexistence of another. A more in-depth information flow analysis is required to answer these questions.

We regard a DIFC system as a state transition system $< \Gamma, \gamma_0, U, \delta >$, where Γ denotes the set of all possible states, γ_0 denotes the initial state, U denotes the set of all processes, and δ denotes DIFC's state transition relations. Permitted information flows trigger off state transitions. Each state consists of labels of the processes in U and a function f: $U \times U \rightarrow \{true, false\}$. In a state, $f(u, v) = true$ if a flow from u to v exists; otherwise $f(u, v) = false$. In state γ_0, for arbitrary $u, v \in U$, $f(u, v)$ is false.

Definition 3.1. (Reachability of information flow in DIFC). *Given a DIFC system* $sys =< \Gamma, \gamma_0, U, \delta >$*, for arbitrary processes* $p, q \in U$*, we say* p *can reach* q *if and only if there exists a state transition path* $< \gamma_0, \gamma_1, \ldots, \gamma_n >$ *and an information flow path* $< v_0, v_1, \ldots, v_n >$*, where* $\gamma_i \in \Gamma$ *and* $v_i \in U$ $(0 \leq i \leq n)$*,* $v_0 = p$*,* $v_n = q$*, satisfying the following conditions:* $f(v_i, v_{i+1}) = true$ *in state* γ_i *and* $f(v_i, v_{i+1}) = false$ *in state* γ_j $(0 \leq j < i < n)$*. Further, we say that state* γ_n *satisfies a primitive proposition* $reached(p, q)$*, which is notated as* $\gamma_n| = reached(p, q)$*.*

Based on the set of primitive propositions $\{reached(x, y)|x \in U, y \in U\}$, we can describe fine-grained security goals with CTL formulae [18].

Definition 3.2. (Security Goals of information flow control). *Let* Q *denote the set of primitive propositions* $\{reached(x, y)|x \in U, y \in U\}$*. We define a security goal statement with a CTL formula by the following grammar:* $\varphi ::= a|\neg\varphi|(\varphi \wedge \varphi)|(\varphi \vee \varphi)|(\varphi \rightarrow \varphi)|AX\varphi|EX\varphi|AF\varphi|EF\varphi|AG\varphi|EG\varphi|A[\varphi\, U\, \varphi]|E[\varphi\, U\, \varphi]$*, where* $a \in Q$*.*

The semantics of temporal operators in CTL formulas are as follows. $AG\varphi$: along all state paths, φ holds globally. $EG\varphi$: There exists a state path where φ holds globally. $AF\varphi$: along all state paths, φ holds at some state in the future. $EF\varphi$: there exists a state path where φ holds at some state in the future. $AX\varphi$: along all state paths, φ holds in the next state. $EX\varphi$: there exists a state path where φ holds in the next state. A $[p\ U\ q]$: along all state paths, p holds until q holds. E $[p\ U\ q]$: there exists a state path where p holds until q holds.

Now, we can formally describe Q-DIFC as follows:

Definition 3.3. (Policy Verification Problem in DIFC). *Given a DIFC system* $sys =< \Gamma, \gamma_0, U, \delta >$*, a set of primitive propositions* $Q = \{reached(x, y)|x \in U, y \in U\}$*, and a security goal* ψ *(*ψ *is a CTL formula over* Q*), a DIFC policy verification problem instance* $< sys, \psi >$ *determines whether* sys *satisfies* ψ*.*

We provide some examples of fine-grained security goals in Q-DIFC as follows:

(i) To verify reachability policies – the following formula asserts that P can reach Q.

$$EF\ [reached(P, Q)]$$

(ii) To verify the path length – the following formula asserts that the length of path from P to Q is not less than 3. (We assume no capability grant operations.)

$$\neg(reached(P, Q) \vee EX[reached(P, Q)] \vee EX[EX[reached(P, Q)]])$$

(iii) To verify taint-tracking policies – the following formula asserts that an information flow from Q to R is not allowed any longer after Q receives tainted data from P.

$$AG\ [reached(P, Q) \rightarrow \neg reached(Q, R)]$$

(iv) To verify privilege transferring policies – the following formula asserts that P cannot send information to R before Q sends declassification capabilities to P.

$$AG\ [\neg reached(P, R)\ U\ reached(Q, P)]$$

3.2. **Complexity analysis of Q-DIFC.** In the following section, we show that in general Q-DIFC is a NP-complete problem. The main reason for the high complexity is that the explored state space is potentially large. We would like to understand how different features of DIFC affect this search space; as such, we consider special cases of Q-DIFC that result from restricting the DIFC schema in various ways.

We first analyze the complexity of Q-DIFC in the Flume system. We call the problem of policy verification in the Flume system Q-Flume, with the problem of reachability verification in Q-Flume called Q-Flume$^-$. We present the following results.

Lemma 3.1. *Q-Flume$^-$ without privilege-transferring is NP-hard.*

Proof: Q-Flume$^-$ is a decision problem. We use a reduction approach to prove this lemma. We introduce an NP-complete problem – MONOTONE 3SAT, and reduce the MONOTONE 3SAT problem to Q-Flume$^-$ by transforming any instance of MONOTONE 3SAT to an instance of *Q-Flume$^-$*. MONOTONE 3SAT problem [19] is defined as the following:

MONOTONE 3SAT: Given a set X of boolean variables and a collection L of clauses over X, such that each $l \in L$ has $|l| = 3$ and l contains all positive or negative variables. The question is whether there exists a satisfying truth assignment on L over X. This problem is denoted as M3SAT (L, X).

We reduce M3SAT (L, X) to Q-Flume$^-$ in polynomial time as follows: Given a MONOTONE 3SAT instance M3SAT(L, X), assume $X = \{x_1, x_2, \ldots, x_K\}$ and $L = a_1 \wedge a_2 \wedge \ldots \wedge a_M \wedge b_1 \wedge b_2 \wedge \ldots \wedge b_N$, where each clause $a_i = x_{i_1} \vee x_{i_2} \vee x_{i_3}$ and each clause $b_j = \overline{x_{j_1}} \vee \overline{x_{j_2}} \vee \overline{x_{j_3}}$. In the corresponding Q-Flume$^-$ instance, we let S denote the set of secrecy tags while allowing U to be the set of processes. For each process p in U, the secrecy labels and the capabilities of p are denoted as $S(p)$ and $C(p)$, respectively. Assume that both the sets of integrity tags and the set of global capabilities are empty. We produce a Q-Flume$^-$ instance as summarized in the following process.

Tag Construction. For each clause a_i, $1 \leq i \leq M$, we introduce a secrecy protection tag d_i into S. For each clause b_j, $1 \leq j \leq N$, we introduce six protection tag ${d'_j}^1$, ${d'_j}^2$, ${d'_j}^3$, t_j^1, t_j^2, t_j^3 into S.

Processes Construction. For each Boolean variable $x_i \in X$, $1 \leq i \leq K$, we introduce a process u_i into U. For each clause $b_j = \overline{x_{j_1}} \vee \overline{x_{j_2}} \vee \overline{x_{j_3}}$, $1 \leq j \leq N$, we introduce three processes ${u'_j}^1$, ${u'_j}^2$, ${u'_j}^3$ into U. In addition, we introduce two processes p and q. The initializations of secrecy labels and capabilities of the constructed process in the Q-Flume$^-$ instance are described in Table 2.

The security goal of the Q-Flume$^-$ instance is to determine if an information flow path exists from p to q. Clearly, the construction can be finished in polynomial time.

Next, we prove that the M3SAT (L, X) instance is true if and only if the Q-Flume$^-$ instance is true.

We will first prove if the M3SAT (L, X) instance is true, then the Q-Flume$^-$ instance is also true. In the M3SAT (L, X) instance, when the true/false assignments to the variables satisfy the formula L, we let set $X' = \{x_{s_1}, x_{s_2}, \ldots, x_{s_w}\}$ represent the variables assigned to true. We can prove that in the *Q-Flume$^-$* instance there exists a path $<$ p, u_{s_1}, $u_{s_2}, \ldots, u_{s_w}$, $select({u'_1}^1, {u'_1}^2, {u'_1}^3)$, $select({u'_2}^1, {u'_2}^2, {u'_2}^3), \ldots,$ $select({u'_N}^1, {u'_N}^2, {u'_N}^3)$, $q >$, where $select({u'_j}^1, {u'_j}^2, {u'_j}^3)$ $(1 \leq j \leq N)$ denotes the selection of one process from

TABLE 2. Initializations of labels and capabilities of processes in the Q-Flume$^-$ instance

Process	Secrecy label	Capability
p	$\{d_1, d_2, \ldots, d_M\}$	$\{(t_1^1)^{\pm}, (t_1^2)^{\pm}, (t_1^3)^{\pm}\}$
u_x	$\{\}$	$\{d_1^+, d_2^+, \ldots, d_M^+\} \cup \{d_i^- \mid variable\ x\ occurs\ in\ a_i\} \cup$ $\{(d_i'^w)^+ \mid 1 \le i \le N, 1 \le w \le 3\} \cup$ $\{d_i'^w \mid variable\ x\ occurs\ in\ the\ position\ w\ of\ b_i\}$
$u_j'^w$	$\{\}$	$\{(d_i'^y)^+ \mid 1 \le i \le N, 1 \le y \le 3, i \ne j\} \cup$ $\{(d_j'^z)^{\pm} \mid 1 \le z \le 3, z \ne w\}$
q	$\{\}$	$\{\}$

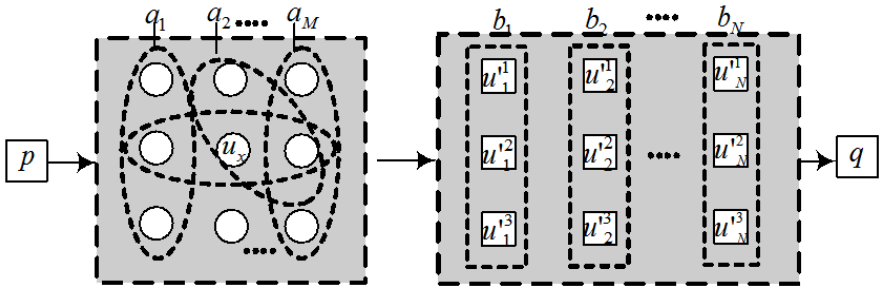

FIGURE 1. An information flow path from p to q in the *Q-Flume*$^-$ instance

the three processes $u_j'^1$, $u_j'^2$, $u_j'^3$ to act as the node having the ability to legally forward information to the next node in the path. Here, we can reasonably assume that each process in the path will delete all of its tags from its contaminated label upon receiving information from the previous node. The reasons for the existence of this path are as follows: 1) since each clause $a_i = x_{i_1} \vee x_{i_2} \vee x_{i_3}$ is true, X' includes at least one variant in the set $\{x_{i_1}, x_{i_2}, x_{i_3}\}$. At this point, corresponding to each tag d_i in p's label, there exists at least one process u_d in the set $\{u_{s_1}, u_{s_2}, \ldots, u_{s_w}\}$ which can delete d_i. This deletion takes place when the information flow passes u_d; and 2) since each $b_j = \overline{x_{j_1}} \vee \overline{x_{j_2}} \vee \overline{x_{j_3}}$ is true, X' includes at most two variables in the set $\{x_{j_1}, x_{j_2}, x_{j_3}\}$. Correspondingly, for each group of tags $\{d_j'^1, d_j'^2, d_j'^3\}$, the current secrecy label of information includes at most two tags in the set $\{d_j'^1, d_j'^2, d_j'^3\}$ when the information flow passes the path $< p, u_{s_1}, u_{s_2}, \ldots, u_{s_w} >$. This guarantees that at least one process in the set $\{u_j'^1, u_j'^2, u_j'^3\}$ can remove these tags from the label of the information before forwarding. Therefore, if L is satisfied, then p can reach q. Figure 1 shows the corresponding information flow path in *Q-Flume*$^-$ instance.

We will now prove that when Q-Flume$^-$ is true, M3SAT (L, X) is also true. Assume that a legal information flow path exists from p to q, referred to as $flow_{p \to \ldots \to q}$. Obviously, p cannot reach q if tags $d_1, d_2, \ldots, d_M$ are not deleted from the label of $flow_{p \to \ldots \to q}$. For each d_i, $flow_{p \to \ldots \to q}$ must have contained at least one of three processes having the capability d_i^-. Meanwhile, the processes that delete tags $\{d_1, d_2, \ldots, d_M\}$ from the label of $flow_{p \to \ldots \to q}$ also bring some tags from $\{d_z'^w \mid 1 \le z \le N, 1 \le i \le w\}$ into the $flow_{p \to \ldots \to q}$. Since no process can obtain the capabilities $(d_j'^1)^-$, $(d_j'^2)^-$, $(d_j'^3)^-$ simultaneously, $flow_{p \to \ldots \to q}$ could not be contaminated by $d_j'^1$, $d_j'^2$, $d_j'^3$ simultaneously. Let U' denote the set of processes which delete tags $\{d_i \mid 1 \le i \le M\}$ from the label of $flow_{p \to \ldots \to q}$. Let X' denote the set of variables in X that corresponds U'. Each variable in X' is assigned to be true, and each variable in $X - X'$ is assigned to be false. At this point, the formula L is satisfied.

Now, we prove that the M3SAT (L, X) instance is true if and only if the Q-Flume$^-$ instance is true. This proves that Q-Flume$^-$ without privilege-transferring is NP-hard.

Lemma 3.2. *Q-Flume$^-$ without decentralized declassification is a P problem.*

Proof: In a Flume system without decentralized declassification, process label changes are monotonic. That is, the changes of processes' secrecy labels are non-decreasing while the changes of their integrity labels are non-increasing. In this case, if a process p is not permitted to send information to a process q directly, p cannot reach q via other processes. Therefore, the reachability from p to q depends on whether or not p's label is less than or equal to q's new maximum label after all other processes have granted all their label-heightening abilities to q. (Such grant operations should be permitted by Flume rules.)

Assume there are n processes. We only consider at most $(n-1)(n-2)/2+1$ label operations and comparisons. Thus, the problem of Q-Flume$^-$ without decentralized declassification can be solved in polynomial time.

Lemma 3.3. *Q-Flume$^-$ without taint-tracking is a P problem.*

Proof: The proof is similar to the proof of Lemma 3.2, without considering the opposite direction of label changes. In a Flume system without taint-tracking, the label changes are monotonic. The changes of processes' secrecy label are non-increasing while the changes of their integrity labels are non-decreasing. In this case, if a process p is not allowed to send information to a process q directly, p cannot reach q via other processes. Therefore, the reachability from p to q depends on whether or not p's new minimum label is less than or equal to q's label after all other processes have granted their declassification abilities to p. (Such grant operations should be permitted by Flume rules.)

Assume there are n processes. We only consider at most $(n-1)(n-2)/2+1$ label operations and comparisons. Thus, the problem of Q-Flume$^-$ without taint-tracking can be solved in polynomial time.

Lemma 3.4. *Q-Flume is in NP.*

Proof: We need to demonstrate that a candidate solution (evidence) for any Q-Flume instance can be verified in polynomial time. Since Q-Flume is to verify whether there exists a counterexample that dissatisfies a specified goal, we now take negative evidences into consideration. Given a negative evidence e (a state path) of a Q-Flume instance, we can prove that the length of this path is not longer than $n(n-1)$, where n represents the number of processes. The reasoning is supported by the role of each primitive proposition in the Q-Flume instance, which asserts an information flow path from one process to another. This denotes that if a primitive proposition q holds in a state γ, then q holds in all successor state of γ. Thus, the set of primitive propositions holding in a state γ is larger than the set of primitive propositions holding in γ's predecessor state. Since the total number of the primitive propositions is $n(n-1)$, the length of an effective negative evidence will not be larger than $n(n-1)$.

On the other hand, there are at most $n(n-1)$ primitive propositions that need to be verified in each state. Therefore, the verification can be finished in O $(n^2(n-1)^2)$ in a worst-case scenario. With this line of logic, we prove our result.

Theorem 3.1. *Q-DIFC is NP-complete.*

Proof: According to Lemmas 3.1-3.4, we know that Q-Flume is in NP and that a sub-problem of Q-Flume is NP-hard. Therefore, *Q-Flume is NP-complete.* Further, as shown in Section 2.4, when given a Flume system, we have the ability to construct an

equivalent Asbestos/Histar system in linear time. Thus, the results of Lemmas 3.1-3.4 still hold in Asbestos/Histar systems. By this logic, Q-DIFC in Asbestos/Histar systems is also NP-complete. We conclude that in general Q-DIFC is NP-complete.

Theorem 3.1 shows that: 1) in the worst case, the policy verification problem in DIFC (we call this problem Q-DIFC) cannot be solved in polynomial time since Q-DIFC is NP-complete; 2) the high complexity of Q-DIFC is attributed to the mechanisms of decentralized declassification and taint-tracking. One can easily solve a Q-DIFC without decentralized declassification and taint-tracking with the use of a fast algorithm. However, this algorithm may not be practical since DIFC lacking decentralized declassification and taint-tracking is unable to achieve fine-grained security goals. Therefore, to solve a general Q-DIFC, an acceptable solution should solve fewer Q-DIFC instances over long time periods, while solving most Q-DIFC instances in short time periods.

4. Model Checking Approach to DIFC Policy Verification.

In this section, we propose a formal method of model checking to solve Q-DIFC. Formal methods consist of mathematically-based verification techniques, which are widely used in the hardware and software industries [20]. Our experimental results show that the proposed model checking approach can solve most Q-DIFC instances in reasonable amounts of time.

4.1. Model checking approach.

Generally speaking, model checking can be used to verify whether or not a system satisfies a desired property. If a model M satisfies the property, a model checker will report true. If M does not satisfy the property, a model checker will provide a counterexample showing a violation of the property.

In detail, model checking uses the Kripke structure [21] to describe a verification model. This structure is a 4-tuple $M = (S, S_0, R, L)$, where S represents a set of finite states, S_0 is a set of initial states, $R \subseteq S \times S$ is a transition relation, $L : S \to 2^{AP}$ is a labeling function, where AP is a set of atomic propositions.

In order to construct the formal policy verification model M of an DIFC system $< \Gamma, \gamma_0, U, \delta >$, we can let $S = \Gamma$, $S_0 = \gamma_0$, $R = \delta$, $AP = \{reached(x, y) | x \in U, y \in U\}$, and L represent the set of existing information flow paths in a given state. Moreover, we use CTL formulae over atomic propositions AP to describe its security goals (properties).

This section demonstrates how to use the proposed approach to verify polices in the Flume system, although this approach can also be applied in any arbitrary DIFC systems. We cast the model of Flume's policy verification into NuSMV language as shown in Figure 2.

As shown in Figure 2, each process i corresponds to a $p[i]$ – an instance of a reusable module *proc*. We introduce the following vraiables for each $p[i]$: variable s denotes the secrecy label of process i, variable *add* denotes i's capabilities of adding tags to its labels, variable *sub* denotes i's capabilities of deleting tags from its labels, variable *id* denotes the process identifier, and array-type variable *taint* denotes the taint state of a process (where *taint* $[j]$ records whether process i has been contaminated by the information flow originated from process j). The variables are initialized in the *main* module.

The *main* module randomly selects a pair of sender and receiver processes in the current system state. After receiving the sender's information via the interface parameters of *proc* module, the receiver updates its state information according to Flume's control rules and the sender's taint information.

By default, each sender will grant its capabilities to the receiver, thus enabling the receiver to have the greatest chance to forward the received information. Each sender/receiver pair communicates once in virtue of the variable *once* in *proc* module (here omitting the integrity label to simplify description). Based on this verification model, we are able to

```
MODULE main
VAR
   p :array 1..NUM_PROC of proc(next(receiver),next(sender),
      next (in_taint),next(in_label),next(in_add),next(in_sub));
   sender , receiver :1..NUM_PROCESS;
   in_s, in_add, in_sub :set;
   in_taint : array 1..NUM_PROCESS of boolean;
ASSIGN
   for (i=1; i≤NUM_PROC;i++)
     for (j=1; j≤NUM_PROC; j++)
     init (p[i].taint[j]):= i=j?TRUE:FALSE;
     init (p[i].id):=i;
     initialize the labels and capabilities of P [i];
   next (sender)    := 1..NUM_PROCESS;
   next (receiver)  := 1..NUM_PROCESS;
   next (in_taint)  := p[sender].taint;
   next (in_s)      := p[sender].s;
   next (in_add)    := p[sender].add;
   next (in_sub)    := p[sender].sub;
SPEC   CTL formula
```

```
MODULE proc(receiver,sender ,tainted,in_s,in_add,in_sub)
VAR
   flow :boolean;
   taint :array 1..NUM_PROCESS of boolean
   s ,add,sub : set;
   once   :array 1..NUM_PROCESS of boolean;
   id     :1..NUM_PROCESS
ASSIGN
   for (i=1; i≤NUM_PROCESS;i++) init(once[i]):= FALSE;
   next(flow) :=!once[sender] & id=receiver & id!=sender &
              (in_s−in_sub) ≤(s ∪add)?TRUE:FALSE;
   for (i=1; i≤NUM_PROCESS;i++)
      next(taint[i]):=next(flow) & tainted[i]?TRUE:taint[i];
   For (i=1; i≤NUM_PROCESS;i++)
     next(once[i]):=selected=i & next(flow)?TRUE:once[i];
   next(s)      :=next(flow)?(label∪(in_s−in_sub):s;
   next(add)    :=next(flow)?in_add∪add:add;
   next(sub)    :=next(flow)?in_sub∪sub:sub;
```

FIGURE 2. Policy verification model in NuSMV language

verify various security goals. For example, a CTL formula with the form "$EF!p[i].taint[j]$" verifies whether process j can reach process i.

4.2. **Preprocessing.** Note that some processes may be irrelevant to the verification process. The preprocessing phase is used to delete irrelevant processes; thus enabling the size of search space can be reduced.

Given a Q-DIFC instance $< sys, \psi >$, the preprocessing phase includes three types of pruning, as follows:

(1) It deletes processes lacking any capability (since these processes do not have the ability to influence any information flow path);

(2) It deletes the processes whose capabilities cannot influence the information flow paths described in the formula ψ. To illustrate this, we use V to represent these processes that are capability-irrelevant to ψ. For each primitive proposition $x = reached(p, q)$ in ψ, let $L = (S_p - C_P^-) - (S_q \cup C_q^+)$, and let K_x represent the set of processes that are capability-irrelevant to x (with K_x's initial value equal to the set of all processes). We first select a process r from K_x, where r satisfies the condition that $C_r^+ \cap L \neq \varphi$ or $C_r^- \cap L \neq \varphi$. We then update L and K_x as $L = L \cup (S_r \cup C_r^+)$ and $K_x = K_x - \{r\}$. This search process is repeated until K is empty or no r can be found in K. Now, we obtain K_x. Further, $V = \underset{x \in \psi}{\cap} K_x$, where x is a primitive proposition;

(3) It deletes the processes that cannot reach any process in ψ, since they have no chance to exert their abilities to influence the information flow paths in ψ. For the purpose of this explanation, we use F to represent these processes which cannot reach any process in ψ. Let F's initial value be $M - B$, where M is the set of all processes and B is the set of the processes in ψ. We first select a process r from F, where r is permitted to send information to a process in B. Then F and B are updated as $F = L - \{r\}$ and $B = B \cup \{r\}$. The search process is repeated until F is empty or no r can be found in F. At last, we obtain F.

Suppose an instance has a total of m processes. The first two types of pruning described here take time m, while the last type of pruning takes time m^2. The computational complexity of the preprocessing is $O(m^2)$.

4.3. **Experiments.** In this section, we evaluate the performance of the proposed approach. We use NuSMV 2.5.2 running on a Windows XP with an Intel P4 2.8G CPU and 2GB of memory in the experiments.

We constructed 8 groups of Q-DIFC instances. They are G1, G2, ..., and G8 as shown in the first row of Table 3. Each group has a total of 10 instances. Each instance occurring in the same group has the same number of processes and tags, as shown in row 2 "Num of processes" and row 3 "Number of tags" of Table 3 respectively. For each process p and each tag d, p's label depends on the random choice based on $\{0, 1\}$. p's capabilities are determined by the random choice based on $\{0, 1, 2, 3\}$, where $0, 1, 2$ and 3 denote the capabilities of the following options: adding d, removing d, both adding and removing d, and no capabilities. Moreover, we constructed 10 different CTL formulae to express security goals, with the connectives in these formulae being evenly selected from the CTL operator set. We set the n-th version of these 10 formulae as the security goal of the n-th instance of each group.

TABLE 3. Algorithm performance evaluation on various-scales data

	G1	G2	G3	G4	G5	G6	G7	G8
Num of processes	10	20	30	40	80	120	160	200
Num of tags	4	6	8	8	15	20	25	30
N/A	0	0	0	0	0	1	3	6
Runtime-1	2	6	27	47	174	245	633	1565
Runtime-2	2	8	39	68	235	660	1465	–

According to our experimental results, the instances can be categorized into three types: *Type 1* instance can be solved in 30 minutes with its specified security goal not satisfied; *Type 2* instance can be solved in 30 minutes with its specified security goal satisfied; *Type 3* instance cannot be solved in 30 minutes. Row "*Runtime-1*" describes the average time in seconds taken to solve the *Types 1* instances, while Row "*Runtime-2*" describes this average time for the *Types 2* instances. Row "N/A" describes the number of the *Type 3* instances in each group. The experimental results are shown in Table 3. Please note, there is no *Type 2* instance in G8; thus the entry (Runtime-2, G8) in Table 3 is empty.

Table 3 shows that the proposed approach can solve most cases of nontrivial size. The runtime of the proposed approach does not increase exponentially with the increase of the processes and tags. A reason for this could be that the scale of the problem does not increase in proportion to the number of processes and tags occurring under constraints of partial relations of the processes' labels.

Further, we evaluate the effects of preprocessing and capability grant operations on the overall performance of the approach. We generate 6 instances with a different number of processes (denoted as n) and a different number of tags (denoted as m). The experimental results are shown in Figure 3. *Runtime-A* measures the runtime in the case where preprocessing is not conducted and capability grant operations are permitted. *Runtime-B* measures the runtime in the case where preprocessing is conducted while no capability grant operations are permitted. *Runtime-C* measures the runtime in the case where both preprocessing and capability grant operations are permitted.

By comparing the results between *Runtime-A* and *Runtime-C*, we can find preprocessing plays an important role in improving the efficiency of the approach. In particular, with the increase of m or n, preprocessing improves the efficiency of the approach accordingly. By comparing the results between *Runtime-B* and *Runtime-C*, we find that, although capability grant operations may not represent the key factor of the exponential complexity

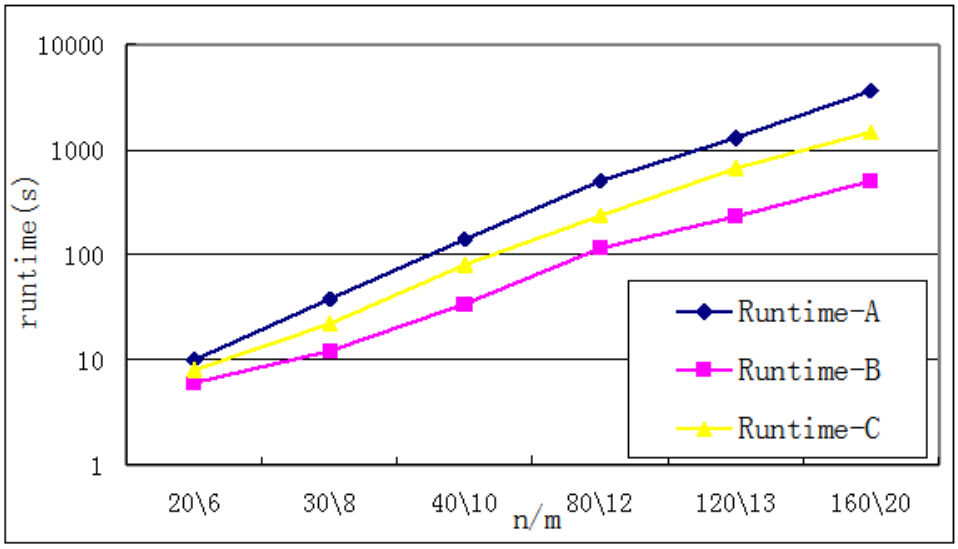

FIGURE 3. Effects of preprocessing and capability grant operations on the performance of the algorithm

of policy verification, in most case its presence will remarkably increase the computing time of policy verification. In particular, the increase of m or n will cause the effects of capability grant operations on performance to increase accordingly.

5. **Conclusions.** In this paper, we have modeled the policy verification problem in DIFC, while using Computational Tree Logic formulae to express the fine-grained security goals as they relate to the constraints and correlations between informational flow paths. In our research, we have proven that the policy verification problem in DIFC is NP-complete. We have further studied the main factors resulting in high computational complexity, and proposed a model-checking approach for policy verification in DIFC systems. The experimental results in Flume's policy verification demonstrate the high efficiency of the proposed approach.

The model checking approach in this paper provides a feasible method for solving the policy verification problem in DIFC. There are many other approaches in existing literature that can also be employed in this field. Future work will involve making further improvements in solving DIFC's policy verification issues.

Acknowledgment. This work is partially supported by the National Natural Science Foundation of China under Grant No. 61070186 and No. 61100181, and the National Basic Research Program of China under Grant No. 2011CB311801. We gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] C.-C. Lo, C.-C. Huang, F.-Y. Lee, K.-Y. Chen, P.-H. Ho and A. Graebner, A flexible access control mechanism for web services, *ICIC Express Letters*, vol.5, no.4(B), pp.1377-1383, 2011.

[2] Y. Liu, Research on the construction and assessment of the technical control models in network information communication, *ICIC Express Letters*, vol.5, no.1, pp.27-33, 2011.

[3] A. C. Myers and B. Liskov, Protecting privacy using the decentralized label model, *ACM Transactions on Software Engineering and Methodology*, vol.9, no.5, pp.410-442, 2000.

[4] S. VanDeBogart, P. Efstathopoulos, E. Kohler, M. Krohn, C. Frey, D. Ziegler, F. Kaashoek, R. Morris and D. Mazieres, Labels and event processes in the Asbestos operating system, *ACM Transactions on Computer Systems*, vol.25, no.4, 2007.

[5] N. Zeldovich, S. Boyd-Wickizer, E. Kohler and D. Mazieres, Making information flow explicit in HiStar, *Proc. of Usenix Association the 7th Usenix Symposium on Operating Systems Design and Implementation*, Seattle, Washington, pp.263-278, 2006.

[6] M. Krohn, A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler and R. Morris, Information flow control for standard OS abstractions, *Proc. of the 21st ACM Symposium on Operating Systems Principles*, Skamania Lodge, Southern Washington State, pp.321-334, 2007.

[7] M. Krohn and E. Tromer, Noninterference for a practical DIFC-based operating system, *Proc. of the IEEE Symposium on Security and Privacy*, Berkeley, California, pp.61-76, 2009.

[8] B. Lampson, Making untrusted code useful: Technical perspective, *Communications of the ACM*, vol.54, no.11, pp.92-92, 2011.

[9] P. Efstathopoulos and E. Kohler, Manageable fine-grained information flow, *Proc. of the 3rd ACM European Conference on Computer Systems*, Glasgow, Scotland, pp.301-313, 2008.

[10] A. Chaudhuri, P. Naldurg and S. K. Rajamani, EON: Modeling and analyzing dynamic access control systems with logic programs, *Proc. of the 15th ACM Conference on Computer and Communications Security*, Alexandria, Louisiana, pp.181-390, 2008.

[11] W. R. Harris, N. A. Kidd, S. Chaki, S. Jha and T. Reps, Verifying information flow control over unbounded processes, *Proc. of the 16th International Symposium on Formal Methods, LNCS*, Eindhoven, Netherlands, vol.5850, pp.773-789, 2009.

[12] W. R. Harris, S. Jha and T. Reps, DIFC programs by automatic instrumentation, *Proc. of the 17th ACM Conference on Computer and Communications Security*, Chicago, Illinois, pp.284-296, 2010.

[13] J. D. Guttman, A. L. Herzog, J. D. Ramsdell and C. W. Skorupka, Verifying information flow goals in security-enhanced Linux, *Journal of Computer Security*, vol.13, no.3, pp.115-134, 2005.

[14] B. Hicks, S. Rueda, L. S. Clair, T. Jaeger and P. McDaniel, A logical specification and analysis for SELinux MLS policy, *ACM Transactions on Information and System Security*, vol.13, no.3, pp.26:1-26:31, 2010.

[15] B. Sarna-Starosta and S. D. Stoller, Policy analysis for security enhanced Linux, *Proc. of the Workshop on Issues in the Theory of Security*, Barceloma, Spain, pp.1-12, 2004.

[16] M. Harrison, W. Ruzzo and J. Ullman, Protection in operating system, *ACM Communication*, vol.19, no.8, pp.461-471, 1976.

[17] S. Jha, N. H. Li, M. V. Tripunitara, Q. Wang and W. H. Winsboroug, Towards formal verification of role-based access control policies, *IEEE Transactions on Dependable and Secure Computing*, vol.5, no.4, pp.242-255, 2008.

[18] E. Clarke, O. Grumberg and D. Peled, *Model Checking*, MIT Press, Cambridge, MA, 1999.

[19] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., New York, 1979.

[20] L. Lu and J. Ma, Formal method for the analysis of security protocols, *ICIC Express Letters*, vol.5, no.10, pp.3785-3789, 2011.

[21] A. Biere, A. Cimtti, E. M. Clarke, M. Fujita and Y. Zhu, Symbolic model checking using SAT procedures instead of BDDs, *Proc. of the 36th Annual Conference on Design Automation*, New Orleans, Louisiana, pp.317-320, 1999.