Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/080464

Quantitative Robust Declassification

Hao Zhu[1,2,*], *Yi Zhuang*[2] *and Xiang Chen*[1]

[1] School of Computer Science and Technology, Nantong University, Nantong, 226019, P. R. China
[2] School of Computer Science and Technology, Nanjing University of Aeronautics Astronautics, Nanjing, 210016, P. R. China

Received: 2 Sep. 2013, Revised: 5 Dec. 2013, Accepted: 6 Dec. 2013
Published online: 1 Jul. 2014

Abstract: The previous declassification policies focus on qualitative analysis of security properties along different dimensions, lacking quantitative analysis of them. As a step in this direction, we relax restrictiveness of robustness of declassification from the quantitative aspect, and propose a definition of robustness rate of declassification, based on Shannon's measure method of information lattice. We show our definition is equivalent to robust declassification when the value of robustness rate is equal to 1. Moreover, we make a theoretical and experimental analysis of robust rate about the laundering attack on average salary, respectively. The experimental results are consistent with the theoretical results completely.

Keywords: Declassification, quantitative information flow, non-interference, robustness

1 Introduction

Language-based information-flow security holds the promise of automatically and efficiently analyzing the information flow within a program so that end-to-end security policies may be enforced [1]. The baseline policy of information-flow is non-interference, which prohibits secret inputs from leaking to public outputs [2]. However, non-interference is too restrictive to write useful programs in the real world. The programs often deliberately release certain secret information, e.g. password authentication. Declassification policies relax the restrictiveness of noninterference and allow deliberate release of secret information to public outputs.

Sabelfeld and Sands [3] surveyed declassification policies by classifying the basic goals according to what information is released, who releases information, where information is released and when information can be released. The different dimensions of policies tend to address different aspects of declassification, so declassification policies trend to combine different dimensions to offer enhanced security. However, declassification policies proposed so far are limited to qualitative analysis [4–6].

On the other hand, quantifying information flow research aims to measure the amount of leaked information that can be gained by observations on the running of programs. In the past years a number of works have provided a solid background for this area [7, 8]. The theory is mainly based on information theoretical terms, the main being Shannon's entropy, Renyi's entropy and so on. More recently, Malacaria and Heusser [9] presented the algebraic view of quantifying information flow, based on lattice theory, and this brings quantifying information flow research closer to existing qualitative works. These theoretical approaches have been shown to be powerful, providing precise reasoning techniques for program constructs including loops [15]. As for the enforcement mechanism, some works have investigated automation of such theory, and verification techniques like abstract interpretation, bounded model checkers and SAT solvers have been applied in this area, but there remains a number of challenges [16].

With this article, we close the gap between the qualitative policies and quantitative analysis by integrating quantifying information flow with the robust declassification policy [5, 11, 12]. Our idea is that the program is secure if the amount of information flow from secret to public, excluding information flow of legal declassification, is not too much from a quantitative point of view. Based on this idea, we propose a definition of robustness rate of declassification and present the enforcement mechanism of model checking.

The rest of the paper is organized as follows. Section 2 presents the information lattice. Section 3 states the model of programs based on the transition system.

* Corresponding author e-mail: searain@nuaa.edu.cn

Section 4 describes quantitative measure of declassification. Section 5 depicts model checking experiments. Finally, Section 6 concludes this paper.

2 Information Lattice

Landauer and Redmond [10] showed that information can be represented as a lattice. Let Σ be a state space of a system. Information lattice can be defined in two equivalent manners as follows.

Firstly, information lattice can be defined in terms of equivalence relations. The equivalence classes represent sets of states whose information is indistinguishable. Let $I(\Sigma)$ be the set of all equivalence relation on the set Σ. The ordering $\sqsubseteq$ on $I(\Sigma)$ is defined as

$$\begin{array}{c}\forall \approx_1,\approx_2 \in I(\Sigma)\ (\approx_1 \sqsubseteq \approx_2)\ \Leftrightarrow \\ \forall \sigma_1,\sigma_2 \in \Sigma\ (\sigma_1 \approx_2 \sigma_2 \Rightarrow \sigma_1 \approx_1 \sigma_2)\end{array} \quad (1)$$

It easily follows from (1) that $\langle I(\Sigma),\sqsubseteq\rangle$ is a complete lattice, which induces an algebraic system $\langle I(\Sigma),\sqcup,\sqcap\rangle$, where $\sqcup$ and $\sqcap$ denote join and meet operations standing for the intersection of elements and the transitive closure union of elements in $I(\Sigma)$ respectively. Higher elements in the information lattice can distinguish more states.

Secondly, information lattice can be defined in terms of functions from the set Σ. The functions express the information extracted from a state. For any function f whose domain is Σ, let $\|f\|$ to be the element of $I(\Sigma)$ for which

$$\forall \sigma_1,\sigma_2 \in \Sigma\ (\sigma_1 \|f\| \sigma_2 \Leftrightarrow f(\sigma_1) = f(\sigma_2)) \quad (2)$$

The ordering relationship (1) is translated into

$$\begin{array}{c}\forall \|g\|,\|f\| \in I(\Sigma)\ (\|g\| \sqsubseteq \|f\|)\ \Leftrightarrow \\ \exists \Phi,\forall \sigma \in \Sigma\ (g(\sigma) = \Phi(f(\sigma)))\end{array} \quad (3)$$

A random variable X can be seen as a mapping $\Sigma \to \mathbb{R}$, where Σ is a state space with a probability distribution and $\mathbb{R}$ is the set of real numbers. According to (2), let $\|X\|$ be equivalence relation

$$\forall \sigma_1,\sigma_2 \in \Sigma, \sigma_1 \|X\| \sigma_2 \Leftrightarrow X(\sigma_1) = X(\sigma_2) \quad (4)$$

The notion of semivaluation can be used to quantify the amount of information provided by an element in the lattice. Formally a semivaluation ν on $I(\Sigma)$ is a mapping $I(\Sigma) \to \mathbb{R}$ satisfying order-preserving property and weakened inclusion-exclusion principle. Malacaria and Heusser [9] showed the semivaluation of the join of two lattice elements is the joint entropy, formally

$$\nu(\|X\| \sqcup \|Y\|) = H(X,Y) \quad (5)$$

Thus, the following basic properties hold:

$$\nu(\|X\|) = H(X) \quad (6)$$

$$X \sqsubseteq Y \Rightarrow \nu(\|X\|) \leq \nu(\|Y\|) \quad (7)$$

3 Model of Programs

We consider sequential imperative programs in this paper. Following [11], a program P is modeled by a transition system $\langle M,\mapsto\rangle$, where M is a set of memories (A memory m is a finite mapping from variables to values.) and $\mapsto$ is a transition relation between memories. A trace s of program P is any finite transition sequence $m_0 \mapsto m_1 \mapsto \cdots \mapsto m_{n-1}$, where $n \geq 1$ and m_i $(0 \leq i \leq n-1)$ is any memory. We write $s(i)$ for the i^{th} memory in the trace s. We use notation $Trc_m(P)$ for the set of traces of P starting at the memory m. The set of all traces of P is denoted by $Trc(P) = \bigcup_{m\in M} Trc_m(P)$. We write $s \equiv s'$ if the traces s and s' satisfy the stutter-equivalence relation, which ignores empty transition such as $m \mapsto m$, for example, $(m_0 \mapsto m_1 \mapsto m_1 \mapsto m_2) \equiv (m_0 \mapsto m_1 \mapsto m_2 \mapsto m_2)$. Let T_1 and T_2 be sets of traces. $T_1 \equiv T_2$ if and only if they contain the same traces with respect to $\equiv$, formally:

$$(\forall s \in T_1, \exists s' \in T_2, s \equiv s') \wedge (\forall s' \in T_2, \exists s \in T_1, s \equiv s') \quad (8)$$

A view of the program P is an equivalence relation on M. Let $I(M)$ be the set of all views of the system. By (1), $\langle I(M),\sqsubseteq\rangle$ forms a complete lattice. Given a trace $s \in Trc(P)$, a view $r \in I(M)$, written s/r, is the sequence of equivalence classes of memories in s, formally:

$$\forall i \in \{0 \cdots len(s)\}, (s/r)(i) = [s(i)]_r \quad (9)$$

All possible sequences of equivalence classes under the view r whenever the program P starts in the memory m is the following set:

$$Trc_m(P,r) = \{s/r \mid s \in Trc_m(P)\} \quad (10)$$

We use the view r to denote an ability to distinguish memories which can be directly accessible to an observer for his/her clearance level, and the view $P[r]$ to express an ability to distinguish memories by watching the trace of program P through the view r. Two memories are equivalent under the view $P[r]$ only if the possible traces leading from these memories are indistinguishable with respect to the view r, formally:

$$\begin{array}{c}\forall m_1,m_2 \in M, \langle m_1,m_2\rangle \in P[r] \Leftrightarrow \\ Trc_{m_1}(P,r) \equiv Trc_{m_2}(P,r)\end{array} \quad (11)$$

Given deterministic program P, the initial memory as an input of program P determines a trace; that is, P can be seen as a function from set of initial memories to set of traces. By (2), the corresponding equivalence relation is $\|P\|$.

$$\begin{array}{c}\forall m_1,m_2 \in M, \langle m_1,m_2\rangle \in \|P\| \Leftrightarrow \\ Trc_{m_1}(P,r) \equiv Trc_{m_2}(P,r)\end{array} \quad (12)$$

According to the definition of information lattice, $P[r]$ and $\|P\|$ are equivalent in essence. By (6), the amount of

information provided by $\|P\|$ is $H(\|P\|)$ (i.e. $H(P[r])$). To illustrate, consider the program P_1 containing a secret variable h and a public variable l:

$$\textbf{if } (h == 0) \textbf{ then } l = 0 \textbf{ else } l = 1$$

If we assume that h is a uniformly-distributed 2-bit unsigned integer, with range $0 \leq h < 3$, then the program P_1 partitions h into two equivalence classes:

$$\{\overbrace{\{0\}}^{l=0}, \overbrace{\{1,2,3\}}^{l=1}\}$$

Hence we get

$$H(\|P_1\|) = 1/4 \times \log_2 4 + 3/4 \times \log_2(4/3) = 0.81$$

But if the two equivalence classes is uniformly distributed, then we have $H(\|P_1\|) = 1/2 \times \log_2 2 \times 2 = 1$, which is the channel capacity of $\|P_1\|$, and this case is the most favorable for the attacker. The following result establishes basic relationships between channel capacity and number of equivalence classes [14]:

Theorem 1 If P is deterministic and N is the number of equivalence classes of $\|P\|$, then the maximum possible information leaked where we consider all possible probability distributions on the inputs is $\log_2(N)$ bits, which is the channel capacity of $\|P\|$ (i.e. $C(\|P\|)$).

4 Quantitative Robustness

An active attacker can both change and observe program execution. Let A be the attacking code fragments injected into the program P, and P_A be the attacked program. Hence view $\|P_A\|$ express an ability to distinguish memories by watching the trace of P_A. The attack considered in this paper is an information laundering attack which satisfies $\|A\| \sqsubseteq r$, i.e., the execution of attacking code fragments themselves can not distinguish more memories than the view r [11].

As mentioned before, our starting point for this paper is the robust declassification policy [11, 12], which says that observing attacked program reveals no more information than watching the original program. Here, we restate the definition of this policy and present how our definition improves it.

Definition 1 The program P is robust with respect to A if and only if $(\|P\| \sqcup \|P_A\|) \sqsubseteq \|P\|$.

According to the basic principles of lattice theory, we have $\|P\| \sqsubseteq (\|P\| \sqcup \|P_A\|)$ always holds. Therefore, the condition in Definition 1 can also be $(\|P\| \sqcup \|P_A\|) = \|P\|$. Indeed, if this condition is not satisfied, but the difference in information leaked between views $\|P\| \sqcup \|P_A\|$ and $\|P\|$ is small enough, the program P is robust with respect to A to some extent. This extent of robustness with respect to the attack A can be expressed with the notion of robust rate:

Definition 2 Let Num expresses the number of bits of secret information in program P , the robust rate of program P with respect to A, denoted Rbt_Rate, is given by

$$Rbt\_Rate = \frac{Num - C(\|P\| \sqcup \|P_A\|)}{Num - C(\|P\|)} \tag{13}$$

where $Num - C(\|P\|) \neq 0$.

Note that if $Num - C(\|P\|) = 0$, then secret information is totally leaked by the view $\|P\|$. Hence there is no need to inject the attacking code A into the program P, and the discussion on robust rate of program P to A is meaningless.

Obviously $\|P\| \sqsubseteq (\|P\| \sqcup \|P_A\|)$ always holds, and by (7) we get $C(\|P\|) \leq C(\|P\| \sqcup \|P_A\|)$. Additionally, information leaked is less than secret information in the memory, i.e. $Num - C(\|P\| \sqcup \|P_A\|) \geq 0$. Hence, we have $Rbt\_Rate \in [0,1]$. If $Num - C(\|P\| \sqcup \|P_A\|) = 0$, then $Rbt\_Rate = 0$, which implies that secret information is totally leaked by the view $\|P\| \sqcup \|P_A\|$. If $C(\|P\| \sqcup \|P_A\|) = C(\|P\|)$, which indicates the condition of Definition 1 holds, then the program P is completely robust with respect to A, and by (13) we have $Rbt\_Rate = 1$.

For example, consider the program P_2, which is the example of averaging salaries [4]: suppose secret variables $h_1, \cdots, h_n$ store the salaries of n employees. The average salary computation intentionally declassifies the average but no other information about $h_1, \cdots, h_n$ to a public variable avg:

$$avg = \textbf{declassify}((h_1 + \cdots + h_n)/n)$$

Consider an information laundering attack A_1:

$$h_2 = h_1; h_3 = h_1; \cdots; h_n = h_1$$

We inject A_1 into P_2 to form the attacked program P_{2A_1}:

$$h_2 = h_1; h_3 = h_1; \cdots; h_n = h_1;$$
$$avg = \textbf{declassify}((h_1 + \cdots + h_n)/n)$$

This program leaks all the information of h_1 to avg. Consider another information laundering attack A_2: $h_2 = h_1$. The attacked program P_{2A_2} is:

$$h_2 = h_1; avg = \textbf{declassify}((h_1 + \cdots + h_n)/n)$$

Intuitively, this program leaks partial information of h_1 to avg (assuming $n > 2$) excluding the information allowed to be declassified. Formally, we can make qualitative analysis of this program by the transition system $\langle M, \mapsto \rangle$ mentioned above. The set M consists of a 3-tuple $\langle t, \langle h_1, \cdots, h_n \rangle, avg \rangle$. The component $t \in \{0,1\}$ is a public variable representing the time roughly—the value 0 indicates the program has not run yet, and the value 1 denotes the program has completed. The view r mentioned in Section 3 is given by:

$$\langle t, \langle h_1, \cdots, h_n \rangle, avg \rangle r \langle t', \langle h_1', \cdots, h_n' \rangle, avg' \rangle$$
$$\Leftrightarrow$$
$$(t = t') \wedge (avg = avg')$$

The view $\|P_2\|$ ($i.e. P_2[r]$) can be described as:

$$\langle t, \langle h_1, \cdots, h_n\rangle, avg\rangle \|P_2\| \langle t', \langle h'_1, \cdots, h'_n\rangle, avg'\rangle$$
$$\Leftrightarrow$$
$$(t = t') \wedge (avg = avg') \wedge$$
$$(t = 0 \Rightarrow (h_1 + \cdots + h_n = h'_1 + \cdots + h'_n))$$

The transition $\mapsto_{A_2}$ induced by the attack A_2 is represented as:

$$\langle 0, \langle h_1, h_2, h_3, \cdots, h_n\rangle, avg\rangle \mapsto_{A_2}$$
$$\langle 0, \langle h_1, h_1, h_3, \cdots, h_n\rangle, avg\rangle$$

The view $\|P_{2A_2}\|$ is depicted as:

$$\langle t, \langle h_1, \cdots, h_n\rangle, avg\rangle \|P_{2A_2}\| \langle t', \langle h'_1, \cdots, h'_n\rangle, avg'\rangle$$
$$\Leftrightarrow$$
$$(t = t') \wedge (avg = avg') \wedge$$
$$(t = 0 \Rightarrow (2h_1 + h_3 + \cdots + h_n = 2h'_1 + h'_3 + \cdots + h'_n))$$

The view $\|P_2\| \sqcup \|P_{2A_2}\|$ is denoted as:

$$\langle t, \langle h_1, h_2, h_3, \cdots, h_n\rangle, avg\rangle (\|P_2\| \sqcup \|P_{A_2}\|)$$
$$\langle t', \langle h'_1, h'_2, h'_3, \cdots, h'_n\rangle, avg'\rangle$$
$$\Leftrightarrow$$
$$(t = t') \wedge (avg = avg') \wedge (t = 0 \Rightarrow$$
$$((h_1 + h_2 + h_3 + \cdots + h_n = h'_1 + h'_2 + h'_3 + \cdots + h'_n)$$
$$\vee (2h_1 + h_3 + \cdots + h_n = 2h'_1 + h'_3 + \cdots + h'_n)$$
$$\vee (h_1 + h_2 + h_3 + \cdots + h_n = 2h'_1 + h'_3 + \cdots + h'_n)$$
$$\vee (2h_1 + h_3 + \cdots + h_n = h'_1 + h'_2 + h'_3 + \cdots + h'_n)))$$

From the above analysis we can see that $(\|P_2\| \sqcup \|P_{2A_2}\|) \sqsubseteq \|P_2\|$ does not hold, so the program P_2 is not robust with respect to A_2 by Definition 1. On the other hand, we will make quantitative analysis of these programs by Definition 2 as follows.

To simplify the calculation of robust rate, we restrict the value of the secret variable h_i $(1 \le i \le n)$ to integers 0 and 1, while the public variable can take real numbers. Here we illustrate this calculation with the above programs when $n = 3$. In this case, the secret information can be represented by a triple $\langle h_1, h_2, h_3\rangle$ which contains 3 bits of information (i.e. $Num = 3$), and the observable trace of programs have only one memory status needing consideration, which is variable *avg*. The equivalence classes of view $\|P_2\|$ is listed in Table 1, where the first column gives the possible values of *avg*, and the second column contains the equivalence classes corresponding to the different values of *avg*. Similarly, Table 2 shows the equivalence classes of view $\|P_{2A_2}\|$. The equivalence classes of view $\|P_2\| \sqcup \|P_{2A_2}\|$ is given in Table 3, but the first column of this table has the form of tuple $\langle avg_1, avg_2\rangle$, where the elements avg_1 and avg_2 denote the variable *avg* in the program P_2 and P_{2A_2} respectively; the values of second column in Table 3 is the intersection of the second columns in Table 1 and 2 based on the value of first column in the corresponding line. According to Table 1 and 3, we can have $C(\|P_2\|) = \log_2 4 = 2$ and $C(\|P_2\| \sqcup \|P_{2A_2}\|) = \log_2 8 = 3$. Thus, by (13) in Definition 2 we have $Rbt\_Rate = \frac{Num - C(\|P_2\| \sqcup \|P_{2A_2}\|)}{Num - C(\|P_2\|)} = 0$.

Similarly, the equivalence classes of these views when n is equal to 4 is illustrated in Table 4, 5 and 6. According to them, we have $Rbt\_Rate = 0.322137$.

Table 1: $\|P_2\|$ when $n = 3$

avg	$\langle h_1, h_2, h_3\rangle$
0	$\langle 0,0,0\rangle$
1/3	$\langle 0,1,0\rangle, \langle 1,0,0\rangle, \langle 0,0,1\rangle$
2/3	$\langle 1,1,0\rangle, \langle 0,1,1\rangle, \langle 1,0,1\rangle$

Table 2: $\|P_{2A_2}\|$ when $n = 3$

avg	$\langle h_1, h_2, h_3\rangle$
0	$\langle 0,0,0\rangle, \langle 0,1,0\rangle$
1/3	$\langle 0,0,1\rangle, \langle 0,1,1\rangle$
2/3	$\langle 1,1,0\rangle, \langle 1,0,0\rangle$
1	$\langle 1,1,1\rangle, \langle 1,0,1\rangle$

Table 3: $\|P_2\| \sqcup \|P_{2A_2}\|$ when $n = 3$

$\langle avg_1, avg_2\rangle$	$\langle h_1, h_2, h_3\rangle$
$\langle 0,0\rangle$	$\langle 0,0,0\rangle$
$\langle 1/3,0\rangle$	$\langle 0,1,0\rangle$
$\langle 1/3,1/3\rangle$	$\langle 0,0,1\rangle$
$\langle 1/3,2/3\rangle$	$\langle 1,0,0\rangle$
$\langle 2/3,1/3\rangle$	$\langle 0,1,1\rangle$
$\langle 2/3,2/3\rangle$	$\langle 1,1,0\rangle$
$\langle 2/3,1\rangle$	$\langle 1,0,1\rangle$
$\langle 1,1\rangle$	$\langle 1,1,1\rangle$

5 Model Checking Experiments

It is a heavy work that manual calculating the equivalence classes of each view when the number of employees is large. Therefore, we introduce the bounded model checking tool CBMC [13] to find the number of equivalence classes of a view, borrowed ideas of quantitative information flow from [14].

A policy for model checking defines the assumed number of equivalence classes of a view. The tool CBMC is used to verify or falsify this policy. A view violates a

Table 4: $\|P_2\|$ when $n = 4$

avg	$\langle h_1, h_2, h_3, h_4 \rangle$
0	$\langle 0,0,0,0 \rangle$
1/4	$\langle 0,0,0,1 \rangle, \langle 0,0,1,0 \rangle, \langle 0,1,0,0 \rangle, \langle 1,0,0,0 \rangle$
2/4	$\langle 0,0,1,1 \rangle, \langle 0,1,0,1 \rangle, \langle 0,1,1,0 \rangle$ $\langle 1,0,0,1 \rangle, \langle 1,0,1,0 \rangle, \langle 1,1,0,0 \rangle$
3/4	$\langle 1,1,1,0 \rangle, \langle 1,0,1,1 \rangle$ $\langle 1,1,0,1 \rangle, \langle 0,1,1,1 \rangle$
1	$\langle 1,1,1,1 \rangle$

Table 5: $\|P_{2A_2}\|$ when $n = 4$

avg	$\langle h_1, h_2, h_3, h_4 \rangle$
0	$\langle 0,0,0,0 \rangle, \langle 0,1,0,0 \rangle$
1/4	$\langle 0,0,0,1 \rangle, \langle 0,0,1,0 \rangle, \langle 0,1,0,1 \rangle, \langle 0,1,1,0 \rangle$
2/4	$\langle 0,0,1,1 \rangle, \langle 1,0,0,0 \rangle, \langle 1,1,0,0 \rangle, \langle 0,1,1,1 \rangle$
3/4	$\langle 1,1,1,0 \rangle, \langle 1,0,1,0 \rangle, \langle 1,1,0,1 \rangle, \langle 1,0,0,1 \rangle$
1	$\langle 1,1,1,1 \rangle, \langle 1,0,1,1 \rangle$

Table 6: $\|P_2\| \sqcup \|P_{2A_2}\|$ when $n = 4$

$\langle avg_1, avg_2 \rangle$	$\langle h_1, h_2, h_3 \rangle$
$\langle 0,0 \rangle$	$\langle 0,0,0,0 \rangle$
$\langle 1/4,0 \rangle$	$\langle 0,1,0,0 \rangle$
$\langle 1/4,1/4 \rangle$	$\langle 0,0,0,1 \rangle, \langle 0,0,1,0 \rangle$
$\langle 1/4,2/4 \rangle$	$\langle 1,0,0,0 \rangle$
$\langle 2/4,1/4 \rangle$	$\langle 0,1,0,1 \rangle, \langle 0,1,1,0 \rangle$
$\langle 2/4,2/4 \rangle$	$\langle 0,0,1,1 \rangle, \langle 1,1,0,0 \rangle$
$\langle 2/4,3/4 \rangle$	$\langle 1,0,1,0 \rangle, \langle 1,0,0,1 \rangle$
$\langle 3/4,2/4 \rangle$	$\langle 0,1,1,1 \rangle$
$\langle 3/4,3/4 \rangle$	$\langle 1,1,1,0 \rangle, \langle 1,1,0,1 \rangle$
$\langle 3/4,1 \rangle$	$\langle 1,0,1,1 \rangle$
$\langle 1,1 \rangle$	$\langle 1,1,1,1 \rangle$

policy if it makes more equivalence classes than what is assumed in the policy. We aim to find a successfully verified policy, whose assumed number of equivalence classes of a view is minimal in all successfully verified policies, and this minimal number is the true number of equivalence classes of the view. To illustrate, we consider the policy for the view $\|P_2\| \sqcup \|P_{2A_2}\|$ when assumed number of equivalence classes is 3 and the number of employees is also 3. In the policy, the input of program P_2 or P_{2A_2} can be modeled as follows:

```
int input() {
  int tmp=nondet_int(); //(A)
  __CPROVER_assume(tmp==0 || //(B)
                   tmp==1);
  return tmp;
}
```

In line (A), function *nondet_int*() provided by CBMC is used to model nondeterministic integer values, which is restricted to the range from 0 to 1 by the function *__CPROVER_assume*() of CBMC in line (B). The main function of the policy is illustrated as follows:

```
struct OUT_PA { //(C)
  float o_P;
  float o_A;
  };
 int main() {
  int s[4][3]; // (D)
  struct OUT_PA o_PA[4];
  for(int j=0;j<4;j++) //initializing
    for(int k=0;k<3;k++)
      s[j][k]=input();
  for(int i=0;i<4;i++) {
    o_PA[i].o_P=func_P(s,i); //(E)
    o_PA[i].o_A=func_A(s,i); //(F)
    }
  __CPROVER_assume(
              (o_PA[0]!=o_PA[1])&& //(G)
              (o_PA[0]!=o_PA[2])&&
              (o_PA[1]!=o_PA[2]));
  assert((o_PA[3]==o_PA[0])|| //(H)
         (o_PA[3]==o_PA[1])||
         (o_PA[3]==o_PA[2]));
}
```

In line (C), the structure *OUT_PA* models the tuple $\langle avg_1, avg_2 \rangle$. The two-dimension array in line (D) is used to store 4 groups of inputs, and each group contains salaries of 3 employees. The lines (E) and (F) show that the programs P_2 and P_{2A_2} is called 4 times with 4 group of inputs respectively, where the functions *func_P* and *func_A* model the the programs P_2 and P_{2A_2} respectively. We omit the details of the two functions for reasons of space. The lines (G) and (H) are used to capture the policy's intent, which implies the assumed number of equivalence classes is 3.

We have implemented the function of generating policy files automatically according to the number of equivalence classes and of employees. The experimental results with different parameter values are shown in Table 7, where the first column denotes the number of employees, the following two columns indicate the number of equivalence classes of the views $\|P_2\|$ and $\|P_2\| \sqcup \|P_{2A_2}\|$ respectively, and the last column gives the value of *Rbt_Rate*, which is trending up with the growth in the number of employees.

6 Conclusions

Declassification policies relax noninterference policy such that a deliberate release of some secret information becomes possible, and in this paper we further relax declassification policies from the quantitative aspect,

Table 7: Experimental results

n	$\|\|P\|\|$	$\|\|P\|\| \sqcup \|\|P_{2A_2}\|\|$	*Rbt_Rate*
2	3	4	0
3	4	8	0
4	5	11	0.322137
5	6	14	0.493841

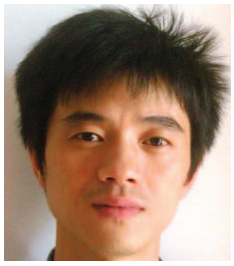
providing more flexible controls of robust but tolerant declassification. Concretely, we extend the definition of robust declassification and propose the notion of robust rate for declassification; we make an experimental analysis of robust rate about the laundering attack on average salary, based on the on-the-shelf symbolic model checkers CBMC. The experimental results verify the theoretical results successfully.

Acknowledgements

This work is partially supported by the National Natural Science Foundation of China under Grant No. 61202006. Thanks for the help.

References

[1] A. Sabelfeld and A. C. Myers, Language-based information flow security, Selected Areas in Communications, **21**, 5-19 (2003).
[2] J. A. Goguen and J. Meseguer, Security policies and security models, Proc. IEEE Symposium on Security and Privacy, 11-20 (1982).
[3] A. Sabelfeld and D. Sands, Declassification: dimensions and principles, Journal of Computer Security, **17**, 517-548 (2009).
[4] A. Askarov and A. Sabelfeld, Localized delimited release: combining the what and where dimensions of information release, Proc. Programming Languages and Analysis for Security, 53-60 (2007).
[5] A. Askarov and A. C. Myers, A semantic framework for declassification and endorsement, Programming Languages and Systems, LNCS, **6012**, 64-84 (2010).
[6] A. Almeida Matos and G. Boudol, On declassification and the non-disclosure policy, Journal of Computer Security, **17**, 549-597 (2009).
[7] G. Smith, On the foundations of quantitative information flow, Foundations of Software Science and Computational Structures, LNCS, **5504**, 288-302 (2009).
[8] H. Yasuoka and T. Terauchi, On bounding problems of quantitative information flow, Computer Security-ESORICS, LNCS, **6345**, 357-372 (2010).
[9] P. Malacaria and J. Heusser, Information theory and security: quantitative information flow, Formal Methods for Quantitative Aspects of Programming Languages, LNCS, **6154**, 87-134 (2010).
[10] J. Landauer and T. Redmond, A lattice of information, Proc. Computer Security Foundations Workshop, **VI**, 65-70 (1993).
[11] S. Zdancewic and A. C. Myers, Robust declassification. Proc. IEEE Computer Security Foundations Workshop, 15-23 (2001).
[12] A. C. Myers, A. Sabelfeld and S. Zdancewic, Enforcing robust declassification and qualified robustness, Journal of Computer Security, **14**, 157-196 (2006).
[13] E. Clarke, D. Kroening and F. Lerda, A tool for checking ANSI-C programs, Tools and Algorithms for the Construction and Analysis of Systems, LNCS, **2988**, 168-176 (2004).
[14] J. Heusser and P. Malacaria, Quantifying information leaks in software, Proc. the 26th Annual Computer Security Applications Conference, 261-269 (2010).
[15] P. Malacaria, Risk assessment of security threats for looping constructs, Journal Of Computer Security, **18**, 191-228 (2010).
[16] P. Malacaria, Quantitative information flow: from theory to practice?, Computer Aided Verification, LNCS, **6174**, 20-22 (2010).



Hao Zhu
received his M.Sc. degree in 2005 from Jiangsu University. He is a Ph.D. candidate in Nanjing University of Aeronautics and Astronautics. He is a associate professor of computer science and technology in Nantong university. His research interests include information security and intelligent computing.

Yi Zhuang
is a professor and Ph.D. supervisor of computer science and technology in Nanjing University of Aeronautics and Astronautics. Her research interests include information security, trusted computing, distributed computing, computer network and wireless senor network et al.

Xian Chen
received a Ph.D. degree in Nanjing University. He is a lecturer of computer science and technology in Nantong university. His research interests include software testing and program analysis.