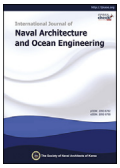
Available online at www.sciencedirect.com

ScienceDirect

Publishing Services by Elsevier

International Journal of Naval Architecture and Ocean Engineering 8 (2016) 423–433

http://www.journals.elsevier.com/international-journal-of-naval-architecture-and-ocean-engineering/



Enforcement of opacity security properties for ship information system☆

Bowen Xing [a,*], Jin Dai [b], Sheng Liu [c]

[a] College of Engineering Science and Technology, Shanghai Ocean University, Shanghai, 201306, China

[b] Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556, USA

[c] College of Automation, Harbin Engineering University, Harbin, Heilongjiang, 150001, China

Received 23 November 2015; revised 10 May 2016; accepted 11 May 2016

Available online 21 July 2016

Abstract

In this paper, we consider the cybersecurity issue of ship information system (SIS) from a new perspective which is called opacity. For a SIS, its confidential information (named as "secret") may be leaked through the working behaviors of each Distributed Control Unit (DCU) from an outside observer called an "intruder" which is able to determine ship's mission state by detecting the source of each data flow from the corresponding DCUs in SIS. Therefore we proposed a dual layer mechanism to enforce opacity by activating non-essential DCU during secret mission. This mechanism is calculated by two types of insertion functions: Safety-assured insertion function (f_{IS}) and Admissibility-assured insertion function (f_{IA}). Due to different objectives, f_{IS} is designed to confuse intruder by constructing a non-secret behaviors from a unsafe one, and the division of f_{IA} is to polish the modified output behaviors back to normal. We define the property of "I_2−*Enforceability*" that dual layer insertion functions has the ability to enforce opacity. By a given mission map of SIS and the marked secret missions, we propose an algorithm to select f_{IS} and compute its matchable f_{IA} and then the DCUs which should be activated to release non-essential data flow in each step is calculable.

Keywords: Ship information system; Enforcing opacity; Insertion function

1. Introduction

With the development of ship technology, the ship system tends to be integrated and distributed which leads to a new research area named ship information system (SIS). Although different ships have different functions, all definitions found in literature for a SIS have one key feature in common. As firstly defined by us in Liu et al. (2014), a typical SIS is composed by several independent subnets (sensor networks, display networks, etc.) and a total ship communication network which can exchange information (reference input, plant output, control input, etc.) among subnets and systems. Similar to a networked system, SIS research is categorized into the following two parts which are Information processing and Information transmission. The research on Information processing is mainly focused on service ability such as controllability, observability (Xing et al., Zhi) and reachability (Xing et al., 2015), etc. Meanwhile, the core attention on Information transmission is service quality, such as anonymity (Wang and Wang, 2014; Kumari and Khan, 2014) and secrecy (Rabbachin et al., 2015; Liang et al, 2009), etc. The application of these research results can ward off network intrusion and attacks, prevent the content of data flow from leaking. However, for a

☆ Fully documented templates are available in the elsarticle package on CTAN.

* Corresponding author.

E-mail address: xbwheu@hotmail.com (B. Xing).

Peer review under responsibility of Society of Naval Architects of Korea.

http://dx.doi.org/10.1016/j.ijnaoe.2016.05.012

SIS, if the functions of each Distributed Control Units are confirmable, the on-going mission may also be revealed by detecting the publishing actions of each DCU. In order to find a solution of protecting some important mission states in SIS, a new property of Information Flow named as opacity should be considered in cybersecurity of SIS. Opacity was first introduced in computer science literature (Mazaré, 2004) and later investigated in Discrete Event System (DES) framework (Lin, 2011; Takai and Oka, 2008; Cassez et al., 2012; Wu and Lafortune, 2013; Falcone and Marchand, 2015; Bryans et al., 2008).

Without losing generality, the model of SIS mission state can be also formulated as a DES and SIS is said to be opaque if its " secret" missions can be hidden from an undesirable external observer which is referred as intruder. The intruder is modeled to have full knowledge of the structure of SIS (include the function and characteristic of the data flow from each DCU), but can not obtain the content of each data flow. And the SIS is said to be opaque if for any secret mission, there must exist at least one other non-secret mission that observationally equivalent from the intruder. Due to the specificity of SIS, here we only consider about Current-Mission State Opacity (CMSO) which is similar to Current-State Opacity-SCO in Bryans et al. (2008). For those systems which are not SCO, the mainly research point is ensuring opacity, which is mainly approached as Supervisory Control Theory (SCT) (Lin, 2011; Dubreil et al., 2010; Ben-Kalefa and Lin, 2011) and enforcement (Falcone and Marchand, 2015). As what has been contrasted by Jacob et al. (2015), the difference between SCT and enforcement is the way they preserve secret, SCT constrains the happening of secret by the controller while enforcement does not restrict system's behaviors but modifies its output to hide the reveal of secret. As there may exist multi-information in a single data flow, not only for secret mission but also for normal mission, it is inadvisable to protect secret mission by restricting the publish of data flow. Thus we research on the enforcement of CMSO. There exist three basic implement methods to enforce opacity by modifying the output information to the observer, 1) deleting events, 2) adding events and 3) delaying the output. Among them, addition of events is a total non-intrusive approach and can be used in such system which a secret is not time-independent. This approach named as "insertion functions" is first proposed by Wu and Lafortune in Wu and Lafortune (2012).

As shown in Fig. 1, regarding as a monitoring interface at system's output, the insertion function has the ability to modify the actual output behavior with fictitious factors by inserting additional observable events. The basic idea to design a suitable insertion function selection algorithm which can cheat the intruder to omit the actual secrets meanwhile the intruder could not learn the setting rule by analyzing the modified output observation with system's structure.

In this paper, we follow and expand the basic design rules of insertion functions which has been presented in Wu and Lafortune (2014). Due to different objectives, f_{IS} is designed to confuse intruder by constructing a non-secret behaviors from a unsafe one, and the division of f_{IA} is to polish the modified output behaviors back to normal. We define the property of "I_2–*Enforceability*" for dual layer insertion functions if they have the ability to enforce opacity. By a given mission map of SIS and the marked secret missions, we propose an algorithm to select f_{IS} and compute its matchable f_{IA} and then the DCUs which should be activated to release non-essential data flow in each step is calculable.

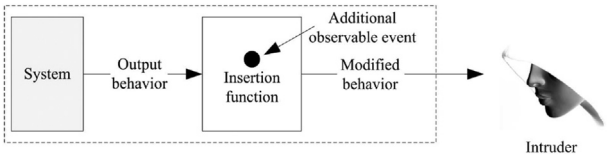


Fig. 1. The insertion mechanism.

To the best of our knowledge, security and opacity issues of ship mission state in SIS have never been addressed so far.

The rest of the paper is organized as follows. Section 2 introduces the typical SIS mission model, and defines the enforcement opacity problem which is introduced in Section 3. Section 4 presents the proposed dual insertion function structure which used in the enforcement of opacity. Section 5 concludes the paper.

2. Structure of SISM

2.1. Basic structure of SIS

As shown in Fig. 2, the structure of SIS we proposed here is similar to Raytheon Companys Total Ship Computing Environment (TSCE) which is one of late-model for USS Zumwalt (DDG-1000). The SIS network is a Double-loop Broadcast Network which connect C2I (Command, Control & Intelligence) system, 3 parallel-hosts monitoring system, Human–Computer Interface (HCI) and dozens of DCUs which are located throughout the ship. And every sensor and actuator which belongs to different systems in the ship are added in this network through remote terminal units (RTUs) with the most nearby DCU. That means a data flow D_m released by DCU_m may include kinds of information that are needed in different systems. As a Broadcast Network in this structure, for each component in SIS, to detect the releasing of data flow is much easier than detecting receiving actions. And according to different sequences of releasing action from each DCU, the on-going mission of the ship is able to be determined.

2.2. SIS mission (SISM) models

If we consider each releasing action as an controllable event, the mission states of SIS can be modeled as deterministic finite state automaton, $G = (X, E, E_{int}, E_{host}, f, f_c, X_0)$, which includes a set of mission states X, a set of release action

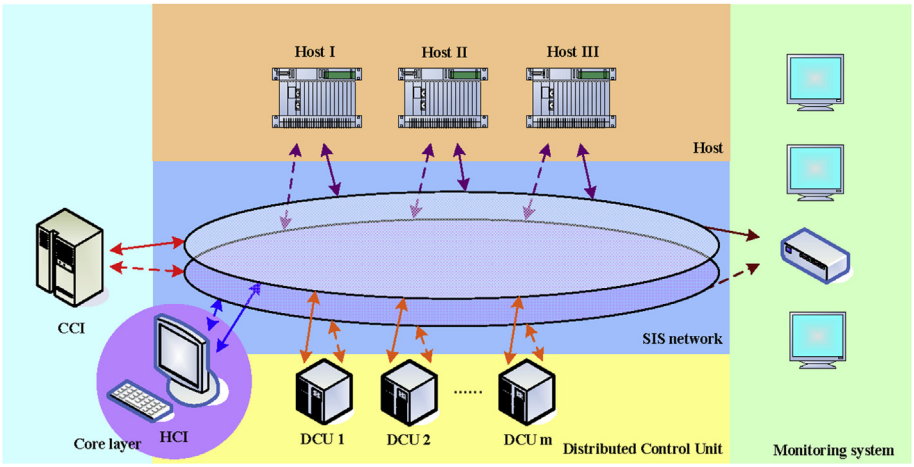


Fig. 2. Basic structure of SIS.

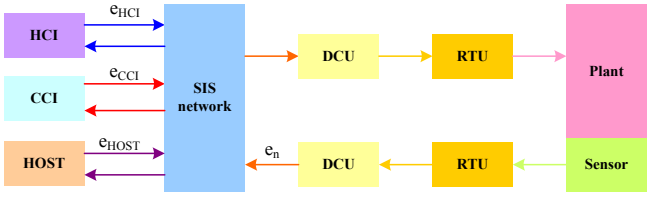


Fig. 3. Basic structure of SISM.

events E, partial transition function f: $X \times E \rightarrow X$, where $f(x, e_i) = y$ means when DCU_i release its data flow to SIS network while mission x is running, the ship is proofed to run mission y instead, $E_{int} = \{e_{CCI}, e_{HCI}\}$ denotes the data flow releasing of CCI and HCI, respectively, similarly $E_{host} = \{E_{host1}, E_{host2}, E_{host3}\}$ means each releasing action from host, and a set of initial state X_0. In addition, partial transition function is defined as f: $X_0 \times E_{host} \rightarrow X_0$, where $f(x, e_{hosti}) = y$ means $HOST_i$ joining the calculating process of initial mission state y from x. And which should be mentioned, not all of mission states in X have actual definitions, some of them may be treated as staged mission states.

The running step of a normal SISM is shown in Fig. 3.

As a beginning flag of a SISM, the task is published by the user from HCI or SIS core from CCI which means e_{CCI} or e_{HCI} is one mark of setting a new mission starting at X_0. Then the host analyzes this mission and decomposes it into several small submissions for each related DCU. It is a typical Hierarchical-cooperative Structure (HCS). Once a DCU receive its submission, it starts to control the plant through mounted RTUs. Accordingly, RTUs will collect data from correlated sensors and send them to DCU. In order to form a closed loop, according to the command from each submission, DCU_n would only put necessary sensor data in the data flow, and the releasing action is denoted as e_n. And we call mission state y as a cooperative mission by DCU_{m1} and DCU_{m2}, if $f(x, e_{m1}e_{m2}) = f(x, e_{m2}e_{m1}) = y$ is satisfied. Meanwhile, DCU_{n1} is said to build a redundant structure with DCU_{n2} for mission state y, if there is $f(x, e_{n1}) = f(x, e_{n2}) = y$.

Let E^* be denoted as the set of all finite strings over X plus the empty string ε, then f can be extended to $X \times E^*$ in the usual manner. For a given state $S \in X$, the language generated by G from S is given by $L(G, S) := \{s \in E^* : \exists x \in S s.t. f(x, s)!\}$ where the meaning of ! is "be defined". For simplicity, when $S = X_0$, the prefix-closed language $L(G, X_0)$ is denoted as $L(G)$. Hence, the event set are partitioned into the set of observable event E_o and the set of unobservable event E_{uo}, i.e., $E = E_o \cup E_{uo}$. The presence of partial observation can be captured by the natural projection $P : E^* \rightarrow E_o^*$ which is defined as

$$P(\varepsilon) = \varepsilon \; and \; P(s\sigma) = \begin{cases} P(s)\sigma & if \; \sigma \in E_o \\ P(s) & if \; \sigma \in E_{uo} \end{cases} \quad (1)$$

Which should be mentioned, the SIS mission (SISM) model just bring a minimal necessary releasing action strings for each mission. If the SIS is in a multi-task mode, the mission state y can not happen from x when DCU_m releases its data flow without adding in all necessary information for y, even if there is $f(x,e_m) = y$. Multi-task mode is a typical working style, due to the high computation ability of DCUs. In most cases, a DCU can add all necessary information it has for every on going missions in one data flow. So we assume that if there is $f(x,e_m) = y$, once DCU_m sends a data flow at mission state x, the mission state is trusted to be y. Imperfect information adding issue will be considered in the future. For now, the way to determine each mission state in a multi-task process is shown in Algorithm 1.

Algorithm 1 Determining method of multi-task

Require: input event q a multi-task string set $L = l_1 l_2 \cdots l_p$

1: **if** $q \in E_{INT}$ **then**
2: $p = p + 1$;
3: initialize string $l_p = q$ and add l_p in L;
4: **else if** $q \in E_{HOST}$ **then**
5: **for** $i = 1$ to $i = p$ **do**
6: **if** $\exists x_{0j} \in X_0$ such that $f_c(x_{0j}, l_i q)!$ **then**
7: $l_i = l_i q$;
8: **end if**
9: **end for**
10: **else if** $q \in E$ **then**
11: **for** $i = 1$ to $i = p$ **do**
12: **if** $\exists x_{0j} \in X_0$ such that $f(x_{0j}, l_i q)!$ **then**
13: $l_i = l_i q$;
14: **end if**
15: **end for**
16: **end if**
17: **return** $L = l_1 l_2 \cdots l_p$;

The algorithm takes the observable release action q and a multi-task string set which include every on-going mission strings as input. Line 1–3, as $q \in E_{INT}$, which means a new mission has been published by HCI or CCI. As shown in Line 4–9, q is a mark of further computing by HOST. According to the analysis above, in Line 10–15, if this data flow is released by a DCU, every mission which need this data would be to next mission state.

Due to the calculations of Algorithm 1 and the hypothesis of DCU's working mode in SIS, the releasing action q would result in a new mission state for any task which needs the data from DCU_q. Therefore, even if SIS is in a multi-task mode, every mission state is able to be determined which means a potential unsafe action string is able to be screened out from a multi-task string.

As for a middle scale SIS, there exist thousands of mission states and hundreds of DCUs, here in Fig. 4, we only recite a partial example of SISM. According to the definition of $G = (X, E, E_{int}, E_{host}, f, X_0)$, there is $X = \{x_0, x_1, \cdots, x_{21}\}$, $E = \{e_1, e_2, e_3, e_4\}$, $E_{int} = \{e_{HCI}\}$, $E_{host} = \{e_{host1}, e_{host2}\}$ and $X_0 = \{x_0, x_1, x_2\}$. It can be seen that the general missions in G is published by HCI, and decomposed by HOST1. HOST2 can be regarded as a parallel computer of HOST1 for x_2. DCU_4 is only a redundant device of DCU_1 during the running process to x_{13}, besides DCU_2 and DCU_3 work cooperatively from mission sate x_{13} to x_{16}.

Choosing x_{22} for example, the minimal necessary release action string is denoted as $e_{HCI}e_{HOST1}e_{HOST2}e_3e_2(e_1/e_4)[e_2e_3]e_1^*e_2e_1e_3e_1e_2^*$, where (e_1/e_4) means redundant and $[e_2e_3]$ means cooperative relation. For a multi-task case, if the action string is $e_{HCI}e_{HOST1}e_1e_{HCI}e_{HOST1}e_{HOST2}e_3e_1e_2$, according to Algorithm 1, there are two mission action strings, one is $e_{HCI}e_{HOST1}e_1e_3e_1$ at state x_8, the other is $e_{HCI}e_{HOST1}e_{HOST2}e_3e_2$ at state x_{12}.

In order to state the proposed enforcing opacity structure completely and comprehensively, without losing generality, in the following of this paper, every example during the analysis is based on Fig. 4.

2.3. Secrets in SIS mission (SISM) models

The "secret mission" in a SIS is such a kind of information (such as activation of fire control radar, fault report of power system, etc.) which SIS wants to "hide" from an

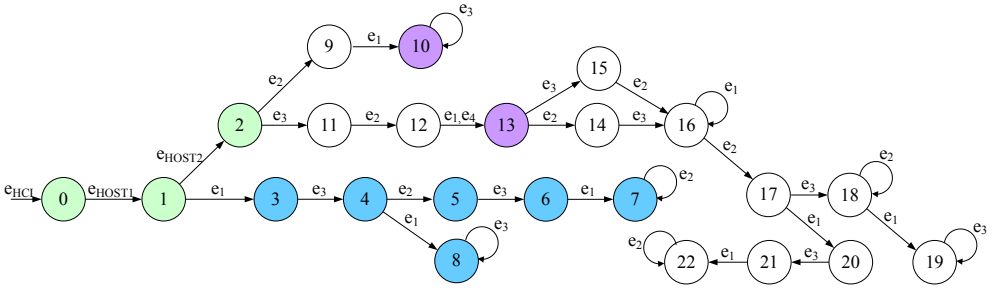


Fig. 4. A typical SISM.

Table 1
Definition of meaningful states in G.

State	Definition
x_3	Power failure in area 1
x_7	Power supplied by sub-grid 2 for area 1
x_8	Power supplied by sub-grid 3 for area 1
x_{10}	Fire-fighting monitoring for area 3
x_{13}	Activation of navigation system
x_{16}	Activation of path planning mission
x_{19}	Control of fin stabilizer
x_{22}	Patrolling mission

external observer, named as "intruder". These missions should not be restricted if they are on the list to be happened.

Here we purpose a sub system of $G = (X, E, E_{int}, E_{host}, f, f_c, X_0)$ which is named as suffix subsystem $G_{suf}(x_n) = (X_{suf-n}, E_{suf-n}, f_{suf-n}, x_n, X_{suf-n0})$ where x_n is the initial state and $x_n \in X$. $L(G_{suf}(x_n))$ is the set of all substrings of $L(G)$ which start from state x_n. And there is $X_{suf-n} \subseteq X$ and $E_{suf-n} \subseteq E$, which means for $\forall x_p \subseteq X_{suf-n}$, there always exists a $x_p = f_{suf-n}(x_n, s_p)!$ where $s_p \in E^*_{suf-n}$.

For a SIS mission model $G = (X, E, E_{int}, E_{host}, f, f_c, X_0)$, if x_{sj} is a secret mission state in a secret set X_s, every state in $G_{suf}(x_{sj})$ is also belonged to X_s. And we define such secret x_{sj} an original secret, if there $\exists x_{ns} \in X/X_s$ and $e \in E$, such that $x_{sj} = f(x_{ns}, e)$ is satisfied, and the set of original secrets in G is denoted as X_{os}. Without any need of proof, G is safe if every original secrets in G are hidden from the intruder.

Based on Fig. 4, the meaningful states of G are listed in Table 1. And DCU_1, DCU_2 and DCU_3 are located in the ship motion control system.

The secret we want to hide is x_3, which means every state in X_{suf-3} should be hidden from the intruder. If the x_3 is protected completely, every state in X_{suf-3} is sensibly safe.

3. Preliminaries of enforcing opacity

3.1. Screening - holding projection

As defined in Wu and Lafortune (2012), the insertion function has ability to insert any event in E_o which looks identical to a system observable event. In order to distinguish between observable event and inserted event in this paper, we denote inserted event by attaching insertion label (subscript i or dashed feature), resulting in $E_i = \{e_i : e \in E_o\}$. Projection P_{und} is a natural projection that treats insertion transitions as unobservable. There is $P_{und}(e_i) = \varepsilon$, $e_i \in E_i$ and $P_{und}(e) = e$, $e \in E_o$ Similarly, projection P_{oi} is used in the exclusion of event which is neither observable nor inserted, that means, given $e \in E \cup E_i$, there is:

$$P_{oi}(e) = \begin{cases} e & if\ e \in E_o \cup E_i \\ \varepsilon & if\ e \in E_{uo} \cup \{\varepsilon\} \end{cases} \tag{2}$$

M_i is a mask which is not able to distinguish between inserted event and observable event. Given an event, there results $M_i(e_i) = e$, $e_i \in E_i$ and $M_i(e) = e$, $e \in E_o$.

Here we propose another projection named screening-holding projection P_{s-h} to analyze admissible strings. P_{s-h} is a nature projection that treat queue jumping transitions as unobservable. Two strings $r = r_1 r_2 \cdots r_m$ and $t = t_1 t_2 \cdots t_n$ are given. The definition of $P_{s-h}((r)/(t))$ is shown as follows:

$$P_{s-h}((r)/(t)) = \begin{cases} \varepsilon & if\ t_1 \notin r \\ t_1 & if\ t_1 = r_{i1} \in r \\ & butt_2 \notin r_{i1} \cdots r_m \\ t_1 t_2 & if\ t_1 = r_{i1} \in r, t_2 = r_{i2} \in r_{i1} \cdots r_m \\ & butt_3 \notin r_{i2} \cdots r_m \\ \vdots & \vdots \\ t & if\ t_1 = r_{i1} \in r, \cdots, t_{n-1} = r_{i(n-1)} \in r_{i(n-1)} \cdots r_m \\ & and\ t_n = r_{in} \in r_{i(n-1)} \cdots r_m \end{cases} \tag{3}$$

And for two strings $r = r_1 r_2 \cdots r_m$ and $t = t_1 t_2 \cdots t_n$, if there exists $r_m = t_n$ and $P_{s-h}((r)/(t)) = t$, we reformulate it as $P_{s-h}((r)/(t)) \doteq t$.

For given strings r and t, the screening-holding projection $P_{s-h}((r)/(t))$ can be counted by Algorithm 2 which operates as follows.

Algorithm 2 Computing method of $P_{s-h}(r/t)$

Require: two strings $r = r_1 r_2 \cdots r_m$ and $t = t_1 t_2 \cdots t_n$
initialize string $u = r$, $j = 1$;
for $i = 1$ to $i = m$ **do**
  if $j \leqslant n$ **then**
    if $r_i = t_j$ **then**
      $u_i = t_j$;
      $j = j + 1$;
    else
      $u_i = \varepsilon$;
    end if;
  else
    break;
  end if
end for
return $P_{s-h}(r/t) = P(u)$;

Theorem 1. For four strings r_a, r_b, t_a and t_b, if $P_{s-h}((r_a)/(t_a)) \doteq t_a$, there is $P_{s-h}((r_a r_b)/(t_a t_b)) = t_a P_{s-h}(r_b/t_b)$.

Proof. We assume that $r = r_a r_b = r_{a1} r_{a2} \cdots r_{am} r_{b1} r_{b2} \cdots r_{bp}$, $t = t_a t_b = t_{a1} \quad r_{a2} \cdots t_{an} r_{b1} r_{b2} \cdots r_{bp}$. And $P_{s-h}(r/t) = P_{s-h}((r_{a1} \quad r_{a2} \cdots r_{am} r_{b1} \quad r_{b2} \cdots r_{bp})/(t_{a1} r_{a2} \cdots t_{an} r_{b1} r_{b2} \cdots r_{bp}))$. Based on Equation (3), if there exists $r_{a(s-1)} = t_{t_a 1}$ and $r_{as} \neq r_{a2}$, $P_{s-h}(r/t)$ can be simplified as $P_{s-h}(r/t) = t_{a1} P_{s-h}((r_{as} \; r_{a(s+1)} \cdots r_{am} r_{b1} r_{b2} \cdots r_{bp}) \quad \cdots t_{an} r_{b1} r_{b2} \cdots r_{bp}))$, such $r_{a(s-1)}$, as $P_{s-h}(r_a/t_a)$, such $r_{a(s-1)}$ can be found in t_a apparently. Further more, as $r_{am} = r_{bn}$, $P_{s-h}(r/t)$ can be simplified step by step into $P_{s-h}(r/t) = r_{a1} r_{a2} \cdots r_{am} \quad P_{s-h}((r_{b1} r_{b2} \cdots r_{bp})/(r_{b1} r_{b2} \cdots r_{bp})) = t_a P_{s-h}(r_b/t_b)$.

Theorem 2. For two strings r and t, if $P_{s-h}((r)/(t)) \doteq t$, for any prefix of t denoted as t_q, there must exist a prefix of r named as r_p, which leads to $P_{s-h}((r_p)/(t_m)) \doteq t_m$.Proof. Here we assume that $t = t_m t_n$ (the last event in string t_m is denoted as t_{mj}). As $P_{s-h}((r)/(t)) \doteq t$, based on Equation (3), there is $P_{s-h}((r)/(t_m)) = t_m$. Furthermore, r can be divided into two parts (denoted as $r = r_p r_q$, the ending event of r_p is marked as r_{pi}) where $t_{mj} \notin r_q$ and $r_{pi} = t_{mj}$. Therefore, $P_{s-h}((r_p r_q)/(t_m)) = P_{s-h}((r_p)/(t_m)) \doteq t_m$..

Theorem 3. For two strings r and t, if $P_{s-h}((r)/(t)) = t$, there must exist a prefix of r, denoted as r_p, such that $P_{s-h}((r_p)/(t)) \doteq t$

Proof. Based on t_q we expand a string t_q which as a result $P_{s-h}(r/t_q) \doteq t_q$, here t is a prefix of t_q. As what has been proofed in Theorem 2, for the two prefixes t and r_p, there must be $P_{s-h}((r_p)/(t)) \doteq t$.

3.2. Insertion functions

An insertion function in this paper has the ability to take an observed event from the system, possibly inserts extra observable events, and then outputs the resulting string. In terms of the intruder, the extra inserted events are indistinguishable from genuine observable events. That is, the intruder does not recognize the virtual insertion label. Therefore, the intruder, observing at the output of the insertion function, cannot tell if the observed string includes inserted events or not. Given a modified string $s \in (E_o \cup E_i)^*$ from the insertion function, the intruder observes s under mask M_i. We can also reconstruct the genuine string by applying projection P_{und} to s. Which should be mentioned that the intruder is assumed to have no knowledge of the insertion function at the outset but it can detect the existence of insertion function by sensing unreasonable output behaviors.

The basic structure of insertion function we used in this paper is defined as $f_I : E_o^* \times E_o \to E_o^* E_o$ which outputs a string with insertion parts based on the knowledge of system's historical and current observed behavior. Given a string t where $P(t) = s$ that has been executed in the system and the current observed event $e_o \in E_o$, the insertion function is defined as $f_I(s, e_o) = s_I e_o$ which means we select string $s_I \in e_i^*$ to insert before e_o. And a pair (s, e_o) is used to denote the systems historical and current observed behavior which determined the output of f_I. Here we assume that no s_I is of unbounded length. The function f_I defines the instantaneous insertion for every (s, e_o). To determine the completely modified string from the insertion function, we define an equivalent string-based insertion function f_I^{str} from $f_I : f_I^{str}(\varepsilon) = \varepsilon$ and $f_I^{str}(s_n) = f_I(\varepsilon, e_1) f_I(e_1, e_2) \cdots f_I(e_1 e_2 e_{n-1}, e_n)$ where $s_n = e_1 e_2 \cdots e_n \in E_o^*$. Given a G as the modified language output from the insertion function which is $f_I^{str}(P[L(G)]) = \{\tilde{s} \in (E_i^* E_o)^* : \tilde{s} = f_I^{str}(s_n) \wedge s \in P[L(G)]\}$ and is denoted hereafter by $L_o ut$.

According to the definition of $P_{s-h}((r)/(t))$ and insertion functions, such substrings of r which cannot follow the event queue of t would be eliminated by $P_{s-h}((r)/(t))$. Hence we have the following result:

Lemma 1. For two strings r and t there must exist such insertion functions mechanism which as result is $f_I^{str}(t) = r$ if $P_{s-h}((r)/(t)) \doteq t$.

Proof. As $P_{s-h}((r)/(t)) = t$ which means r can be rewritten as $r = r_1 \cdots r_{i1} \cdots r_{in} \cdots r_m = r_1 \cdots t_1 \cdots t_n \cdots r_m$. And the insertion function can be designed as follows: $f_I^{str}(t) = f_I(\varepsilon, t_1) \; f_I(t_1, t_2) \cdots f_I(t_1 t_2 t_{n-1}, t_n) = r_1 \cdots r_{i1-1}, t_1, \quad r_{i1+1} \cdots r_{i2-1}, t_2 \cdots, \quad r_i (n-1)+1 \cdots r_{in-1}, t_n = r_1 \cdots r_{in} \in r_m$. The computing method of insertion function set $F_I(r, t)$ when $P_{s-h}(r/t) = t$ is shown in algorithm 3. According to Line 3 and 4, the computational complexity of algorithm 3 is $O(nm)$.

3.3. Problem definition

Normally, designers preset (predefine) part of system's representation as secrets by its importance or possibility of intruder's attention. Due to the particularity of SISM, in this paper, we only consider about CSO, which is:

Algorithm 3 Computing method of insertion function set $F_I(r, t)$ when $P_{s-h}(r/t) = t$

Require: two strings $r = r_1 r_2 \cdots r_m$ and $t = t_1 t_2 \cdots t_n$
1: initialize a insertion function set $F_I(r, t)$ and two string set $U = (u_1 u_2 \cdots u_n)$, $V = (v_1 v_2 \cdots v_n)$;
2: initialize $j = 1$;
3: **for** $i = 1$ to $i = n$ **do**
4: **while** $(r_j \neq t_i || j \leqslant m)$ **do**
5: $v_i = v_i + r_j$;
6: $j = j + 1$;
7: **end while**
8: add $f_I(u_i, t_i) = v_i t_i$ in $F_I(r, t)$;
9: $j = j + 1$;
10: **end for**
11: **return** $F_I(r, t)$;

Definition 1. (*Current-State-Opacity*, *CSO*) A system $G = (X,E,f, X_0)$ is current-state opaque w.r.t. $X_S \subseteq X$ *and* $E_o \subseteq E$ *if* $\forall i \in X_0$ such that $f(i,t) \subseteq X_S$, $\exists j \in X_0$, $\exists t\prime \in L(j,t\prime)$ such that: (i) $f(j,t\prime) \subseteq X \setminus X_S$ and (ii) $P(t) = P(t\prime)$.Based on definition 1, it is known that opacity will hold if the intruder's observation is always belonged to safe strings denoted as l_s, and there is $l_s \subseteq L_s(G)$ where $L_s(G)$ is a sublanguage of $L(G)$ called safe language.As the basic rule of design insertion functions which is to add extra observable events which disguises every unsafe string $l_{usi} \subseteq L(G)/L_s(G)$ in $L(G)$ as a safe and existing string $l_{si} \subseteq L_s(G)$ from the intruder's observation. And if such insertion functions exist, the system is said to be enforced opacity.

According to Definition 1, Algorithm 1 and 2, here we have:

Definition 2. (Enforcing opacity) A unsafe system $G = (X,E,f,X_0)$ with secret set X_s is able to be enforced opacity, if $\forall l_{usi} \in L(G)/L_S(G)$, there exists $l_{si} \in L_S(G)$, such that $P_{s-h}((l_{si})/(l_{usi})) \doteq l_{usi}$ is satisfied and the insertion functions which modified the output behavior as $f_I^{str}(l_{usi}) = l_{si}$ are stored in $F_I(l_{si},l_{usi})$.So the core problem of enforcing opacity in this paper is for a given l_{usi}, how to determine the corresponding l_{si} which leads $P_{s-h}((l_{si})/(l_{usi})) \doteq l_{usi}$. And based on Theorem 1, we can consider this problem gradationally. Based on Definition 1, it is shown that the system G with $E_o = \{e_1,e_2,e_3\}$ and $X_s = X_{suf-3}$ in Fig. 4 is not CSO. And the main work in this paper is how to enforce opacity for such SIS just like G.

4. Dual insertion function structure

In this paper, we propose a new opacity enforcement structure based on dual insertion functions named as Safety-assured insertion function and Admissibility assured insertion function respectively to protect the secrets in SIS.

For such a non-safety SIS, according to this dual insertion function structure, some unnecessary DCUs are activated and release their data flow in SIS during the unsafe strings to change the output behaviors from the intruder. As we mainly consider such data flow from DCUs, here we simplify G as $G = (X,E,f,X_0,X_{ms})$, where X_{ms} is the marked states set (which is also the secret set) of G.

4.1. Definition of f_{IS},f_{IA} and I_2−Enforceability

Definition 3. (Safety-assured insertion function set, $F_{IS}(l_{cni},s_n)$) Given system $G = (X,E,f,X_0,X_{ms})$, if that for $s_n \in L_m(G)$, $\exists l_{cni} \in L(G)/L_m(G)$, such that $F_{IS}(l_{cni},s_n)$, such that $\exists M_i$ $(f_{IS}^{str}(s_n)) = l_{cni}$. Here $F_{IS}(l_{cni},s_n)$ is named as Safety-assured insertion function set of s_n.And the collect of all the possible strings l_{cni} which satisfies Definition 3 is named as Candidate language of s_n, and denoted as $L_{c-sn}(G)$. If $L_{c-sn}(G) = \phi$, s_n is impossible to be modified to any safe string by adding events from the output.Which should be mentioned that $L(G_{suf}(X_{ms}))$ can be treated as the suffix set of all strings which would reveal the secret state belonging to X_{ms}. $X_{suf-X_{ms}} \subseteq X$ is the subsequent states set after the happening of secret states.

Definition 4. (Admissibility-assured insertion function set, F_{IA} (l_{cni-k}, l_{s_n-j})). Given a system $G = (X,E,f,X_0,X_{ms})$ which is able to be repaired to admissibility by $F_{IA}(l_{cni-k}, l_{s_n-j})$ after the modification done by $F_{IS}(l_{cni},s_n)$, if for $\forall l_{s_n-j} \in L(G_{suf}(f(x_0,s_n)))$, there exists $l_{cni-k} \in L(G_{suf}(f(x_0,l_{cni})))$, such that $M_i(f_{IA}^{str}(l_{s_n-j})) = l_{cni-k}$. Here $F_{IA}(l_{cni-k}, l_{s_n-j})$ is named as an Admissibility-assured insertion function set paired with $F_{IS}(l_{cni},s_n)$.Based on the definition of $P_{s-h}((r)/(t))$, the two-tiered insertion mechanism which can enforce opacity we proposed in this paper named as enforceability that could be defined as follows:

Definition 5. (Enforceability, I_2−*Enforceability*) Given system $G = (X,E,f, X_0,X_{ms})$, such dual insertion function structure is called I_2−*Enforcing* if $\forall s_n \in L_m(G)$ and $\forall l_{s_n-j} \in L(G_{suf}(f(x_0,s_n)))$, there exists $l_{cni} \in L(G)/L_s(G)$, such that $P_{s-h}((l_{cni})/(s_n)) = s_n$ (Safety) and $\exists l_{cni-k} \in L(G_{suf}(f(x_0,l_{cni})))$ such that $P_{s-h}((l_{cni-k})/(l_{sn-j})) = l_{sn-j}$ (Admissibility).

4.2. Computing method of safety-assured insertion function set

As what has proofed in Lemma 1 and Algorithm 2, for a given string s, we can always modified it to l_i by adding extra events iff there is $P_{s-h}((l_i)/(s)) \doteq s$ and these extra events can be always determined by $F_I(l_i,s)$. Here if s is unsafe but l_i is safe, $F_I(l_i,s)$ would be a suitable safety-assured insertion function set. Which means for a unsafe string s, we can determine $F_{IS}(l_i,s)$ by finding a safe l_i which satisfies $P_{s-h}((l_i)/(s)) \doteq s$. In most case, there may exist several safe strings meet the assumption above and in this paper we put them in candidate language $L_c(G)$. The computing process of $L_c(G)$ is shown in Algorithm 4.

The algorithm takes the initiation unsafe string s and language $L_s(G)$ as input. Line 1 differentiating each string in $L_s(G)$ by adding a subscript, and preset the candidate language $L_c(G)$ which includes every strings of $L_s(G)$. Lines 2–8, show the way to identify the safe strings which can be modified by adding extra events from s based on Lemma 1, and each strings are remarked in Line 9. Lines 10–17, make a further screening of $L_c(G)$ which only reserve the most simplified and satisfactory prefixes of each strings in $L_c(G)$. If $L_{ci}(G) = \phi$ is satisfied, G is impossible to be enforced opacity by adding extra events (Not I_2−*Enforcing*). The output of this algorithm is the candidate language $L_c(G)$. And by choosing the string l_{ci} from $L_c(G)$ such $F_{IS}(l_{ci},s)$ must exist which can prevent the

reveal of secret. According to Fig. 4, it is shown that the initiation unsafe string is $s = e_1$. Based on Line 2–8 in Algorithm 4, $L_c(G)$ in Line 9 is $\{e_2e_1,\ e_3e_2e_1, e_3e_2(e_1/e_4)[e_3e_2]e_1^*,\ e_3e_2(e_1/e_4)[e_3e_2]e_1^*e_2e_1,\ e_3e_2(e_1/e_4)[e_3e_2]e_1^*e_2e_1e_3e_1,\ e_3e_2(e_1/e_4)[e_3e_2]e_1^*e_2e_3e_2^*e_1\}$. And then a further screening of $L_c(G)$ runs, $\{e_3e_2(e_1/e_4)[e_3e_2]e_1^*, e_3e_2(e_1/e_4)[e_3e_2]e_1^*e_2e_1, e_3e_2(e_1/e_4)[e_3e_2]e_1^*e_2e_1e_3e_1,\ e_3e_2(e_1/e_4)[e_3e_2]e_1^*e_2e_3e_2^*e_1\}$ should be deleted from $L_c(G)$ because of the existence of string $e_3e_2e_1$. The result of candidate language in Line 21 is $L_c(G) = \{e_2e_1, e_3e_2e_1\}$. And all the possible safety-assured insertion function set F_{IS} of G which can protect secret x_s are $F_{IS}(e_2e_1, e_1) = \{f_{IS}(\varepsilon, e_1) = e_{2i}e_1\}$ and $F_{IS}(e_3e_2e_1, e_1) = \{f_{IS}(\varepsilon, e_1) = e_{3i}e_{2i}e_1\}$. The next step runs re-examination of these F_{IS}. Due to the nested loop in Line 10 to 15, the computational complexity of algorithm 4 is $O(n^2)$.

Algorithm 4 Candidate language $L_c(G)$

Require: language $L_s(G)$ and initiation unsafe string s
1: mark each string in $L_s(G)$ as $L_s(G) = \{l_1, l_2, \cdots, l_m\}$ and initialize language $L_c(G) = L_s(G)$;
2: **for** $i = 1$ to $i = m$ **do**
3: **if** $P_{s-h}((s)/(l_i)) \doteq s$ **then**
4: keep l_i in $L_c(G)$;
5: **else**
6: delete l_i from $L_c(G)$;
7: **end if**;
8: **end for**;
9: remark each string in $L_c(G)$ as $L_c(G) = \{l_{c1}, l_{c2}, \cdots, l_{cn}\}$;
10: **for** $i = 1$ to $i = n$ **do**
11: **for** $j = 1$ to $j = n$ **do**
12: **if** $l_{ci} \subset l_{cj}$ **then**
13: put l_{cj} in $L_{c1}(G)$;
14: **end if**;
15: **end for**;
16: **end for**;
17: reset $L_c(G) = L_c(G)/L_{c1}(G)$;
18: **if** $L_{ci}(G) = \phi$ **then**
19: **return** NOT $I_2 - Enforcing$;
20: **else**
21: **return** $L_c(G)$;
22: **end if**

4.3. Computing method of admissibility-assured insertion function set

4.3.1. Admissibility-assured insertion function set

After screening every possible strings in $L(G)$ which can be used in the construction of safety-assured insertion function set, and in this subsection we will determine whether a string l_{ci} in $L_c(G)$ is suitable to enforce opacity which is denoted as I_2–*Enforcing* through l_{ci}. We select each candidate string in $L_c(G)$ by turn, and search Admissibility-assured insertion function set, F_{IA}, which can polish every modified output behaviors back to normal strings, meanwhile, collect these two kind of strings in pair in a set $CL(G)$.

Algorithm 5 Searching algorithmic of $CL(G)$

Require: language$L(G)$, unsafe string s and candidate language$L_c(G)$
1: mark each string in $L_c(G)$ as $L_c(G) = \{l_{c1}, l_{c2}, \cdots, l_{cp}\}$
2: initialize $G_{suf}(x_s) = (X_{suf-s}, E_{suf-s}, f_{suf-s}, x_s)$ and mark each non-prefix string in $G_{suf}(x_s)$ as $L_{n-pre}(G_{suf}(x_s)) = \{l_{s1}, l_{s2}, \cdots, l_{sn}\}$
3: **for** $i = 1$ to $i = p$ **do**
4: define x_{ci}, where$x_{ci} = f(x_0, l_{ci})$;
5: set $G_{suf}(x_{ci}) = (X_{suf-ci}, E_{suf-ci}, f_{suf-ci}, x_{ci})$;
6: mark each string in $G_{suf}(x_{ci})$ as $G_{suf}(x_{ci}) = \{l_{ci1}, l_{ci2}, \cdots, l_{cim_i}\}$;
7: **if** $E_{suf-s} \not\subset E_{suf-ci}$ **then**
8: **return** NOT $I_2 - Enforcing$ through l_{ci};
9: **break**;
10: **else**
11: **for** all $l_{sj} \in Ln - pre(G_{suf}(x_s))$ **do**
12: **if** $\exists l_{cik} \in L(G_{suf}(x_{ci}))$, such that $P_{s-h}((l_{cik})/(l_{sj})) = l_{sj}$ **then**
13: put string pair $l_{sj} - l_{cik}$ in set $CL(G)$;
14: **return** $I_2 - Enforcing$ through l_{ci};
15: **else**
16: **return** NOT $I_2 - Enforcing$ through l_{ci};
17: **break**;
18: **end if**;
19: **end for**;
20: **return** $CL(G)$;
21: **end if**;
22: **end for**;

Algorithm 5 shows the searching algorithmic of $CL(G)$. The algorithm takes as input the initiation unsafe string s and candidate language $L_c(G)$ which is marked in Line 1. Line 2 initializes suffix subsystem $G_{suf}(x_s)$, as x_s is the secret state of G, that means $L(G_{suf}(x_s))$ includes all the strings which can lead $s \cdot l_{sj}$ to a unsafe string. Or said differently, every strings in $L(G)/L_s(G)$ can be denoted as $s \cdot l_{sj}$ where $l_{sj} \in G_{suf}(x_s)$. The Non-prefix $l_{sj} \in L_{pre}(G_{suf}(x_s))$ means l_{sj} is not a prefix of any other strings in $L(G_{suf}(x_s))$. It could consider that language $L(G)$ is build up by several Non-prefix strings and their prefixes, so does the each suffix subsystem of $L(G)$. Based on F_{SA}, when s happens, the intruder is deceived to observe such output l_{ci} which belongs to $L_c(G)$. That is the reason why system could be safe. To enforce admissibility, such insertion function sets should exist that modifies each string $l_{ci} \cdot l_{sj}$ looks normal by the intruder. For a string x_{ci} in $L_c(G)$, Line 4–5 define its suffix subsystem $G_{suf}(x_{ci})$. The intruder can only accept such output behaviors after the happening of s if it has been defined in $G_{suf}(x_{ci})$. As $L(G_{suf}(x_s))$ is fixed, Line 6–27 works like a deeper screening to sort out an acceptable l_{ci} by estimating constructibility from $L(G_{suf}(x_s))$ to $L(G_{suf}(x_{ci}))$.

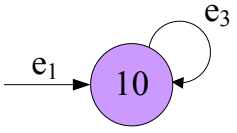


Fig. 5. Suffix subsystem $G_{suf}(x_{10})$.

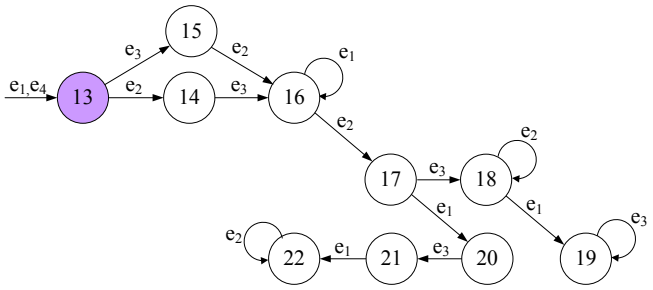


Fig. 6. Suffix subsystem $G_{suf}(x_{13})$.

Based on Lemma 1, Algorithm 3 and Theorem 2, if there exists $l_{ci} \in L_c(G)$ and for every non-prefix $l_{sj} \in L(G_{suf}(x_s))$ in Line 11, there exists $l_{cik} \in L(G_{suf}(x_{ci}))$, such that $P_{s-h}((l_{cik})/(l_{sj})) = l_{sj}$, the admissibility-assured insertion function set $F_{IA}(l_{cik},l_{sj})$ exists. And the correspondences between each l_{cik} and l_{sj} are saved in $CL(G)$. In order to decrease the quantity of calculation Line 7–8, a prejudging method of remove unsuitable candidate strings from $L_c(G)$ is provided. The computational complexity of Algorithm 5 is $O(pw)$, where w is the size of $L_{n-pre}(G_{suf}(x_s))$.

According to Algorithm 4 and Algorithm 5, for a system G, when a unsafe string sl_{sj} is happening, the modified output behaviors received by the intruder are $l_{ci}l_{cik} \in L(G)$ which is determined by $F_{IS}(s,c_i)$ and $F_{IA}(l_{sj},l_{cik})$, the system is I_2–*Enforcing*. And such modification is denoted as $f_{is-ia}(sl_{sj}) = l_{ci}l_{cik}$ and stored in an insertion function set $F_{IS-IA}(G - l_{ci})$.

a. Searching result of $CL(G)$ when choosing l_{c1}

Fig. 5 shows the Suffix subsystem $G_{suf}(x_{10})$. It can been seen that $E_{suf-e2e1} = \{e_3\}$ and $E_{suf-e_1} \not\subset E_{suf-e_2e_1}$. Some strings in $G_{suf}(x_3)$ included specific alphabets that $G_{suf}(x_{10})$ does not have. That means it is impossible to modify every strings in $G_{suf}(x_3)$ to a string which exists in $G_{suf}(x_{10})$ by adding extra events. For this reason, G is not I_2–*Enforcing* through e_2e_1.

b. Searching result of $CL(G)$ when choosing l_{c2}

Then the Algorithm 5 starts to calculate whether l_{c2} is suitable or not.

Fig. 6 shows the Suffix subsystem $G_{suf}(x_{13})$. It can been seen that $E_{suf-e3e2e1} = \{e_1,e_2,e_3\}$ which is equal to E_{suf-e1}, that means l_{c2} passes the prejudging method in Line 7–8. And then the non-prefix string set of $L(G_{suf}(x_s))$ is worked out as $\{e_3e_1e_3^*, e_3e_2e_3e_1e_2^*\}$, the core computing starts. After counting every strings in $L(G_{suf}(x_{13}))$, the result shows that G is I_2–*Enforcing* through $e_3e_2e_1$ and $CL(G) = [e_3e_1e_3^* - [e_3e_2]e_2e_3e_1e_3^*, e_3e_1e_3^* - [e_3e_2]e_1^*e_2e_3e_1e_3^*, e_3e_1e_3^* - [e_3e_2]e_2e_3e_2^*e_1e_3^*, e_3e_1e_3^* - [e_3e_2]e_1^*e_2e_3e_2^*e_1e_3^*, e_3e_1e_3^* - [e_3e_2]e_2e_3e_2^*e_1e_3^*, e_3e_2e_3e_1e_2^* - [e_3e_2]e_1^*e_2e_1e_3e_1e_2^*, e_3e_2e_3e_1e_2^* - [e_3e_2]e_2e_1e_3e_1e_2^*]$.

4.4. Structure of dual insertion function set

The structure of dual insertion functions we proposed in this paper is shown in Fig. 7.

Here we use two kinds of insertion function set F_{IS} and F_{IA} to enforce opacity. For the secret state x_s, the first step is to determine the initiation unsafe string s where $x_s = f(x_0,s)$, and s is the prefix of all the unsafe strings in $L(G)/L_s(G)$ which can reveal the secret x_s. And then, for the second step, we acquire and collect all the possible strings which can be modified by insertion function from s in a subset of $L(G)$ named candidate language $L_c(G)$. And these kind of insertion function which is designed to confuse intruder by constructing a non-secret behaviors from a unsafe one is named as Safety-assured insertion function, f_{IS}. If $L_c(G)$ is a null set, G is impossible to be enforced opacity by adding extra events (Not I_2–*Enforcing*). If $L_c(G)$ is not null, in step 3, we select each candidate string in $L_c(G)$ by turn, and search such insertion functions named as Admissibility-assured insertion functions, f_{IA} which can polish every modified output behaviors back to normal strings, meanwhile, collecting these two kind of strings in pair in a set $CL(G)$ and at this situation G is I_2–*Enforcing*, otherwise it is not.

4.5. Optimizing

Actually, based on the computation mode above, for non-safe system G there may exist several options to enforce opacity. As in this paper, every addition event is physically done by realising data flow in SIS network which would cause a waste of network resources. Therefore, optimization is extremely important to consider. Here we proposed 3 principles to comparing each solution.

Table 2
Result of I_2–*Enforcing* through *ca*.

Unsafe string of G	State	Modified strings	Modified state
e_1	x_3	$e_{HOST2}e_{3i}e_{2i}e_1$	x_{13}
e_1e_3	x_4	$e_{HOST2}e_{3i}e_{2i}e_1e_3$	x_{15}
$e_1e_3e_1$	x_8	$e_{HOST2}e_{3i}e_{2i}e_1e_3e_{2i}e_{2i}e_{3i}e_1$	x_{19}
$e_1e_3e_1e_3*$	x_8	$e_{HOST2}e_{3i}e_{2i}e_1e_3e_{2i}e_{2i}e_{3i}e_1e_3*$	x_{19}
$e_1e_3e_2$	x_5	$e_{HOST2}e_{3i}e_{2i}e_1e_3e_2$	x_{16}
$e_1e_3e_2e_3$	x_6	$e_{HOST2}e_{3i}e_{2i}e_1e_3e_2e_{2i}e_1e_3$	x_{21}
$e_1e_3e_2e_3e_1$	x_7	$e_{HOST2}e_{3i}e_{2i}e_1e_3e_2e_{2i}e_1e_3e_1$	x_{22}
$e_1e_3e_2e_3e_1e_2*$	x_7	$e_{HOST2}e_{3i}e_{2i}e_1e_3e_2e_{2i}e_1e_3e_1e_2*$	x_{22}

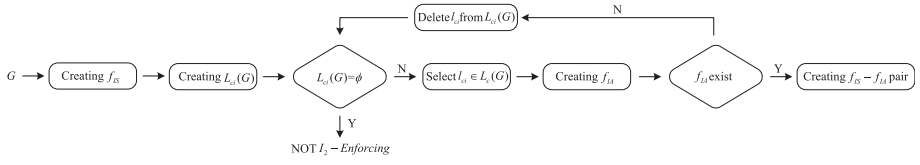


Fig. 7. Structure of dual insertion functions.

a. Selection of candidate string l_{ci}

Rule (a). For two candidate string l_{ca} and l_{cb} which both have the ability to build I_2–*Enforcing* insertion function structure, if $n_{ca}>n_{cb}$, choosing $F_{IS-IA}(G-l_{cb})$, and vice versa, where n_{ca} and n_{cb} are the sum of inserted events for each unsafe string in G based on l_{ca} and l_{cb}, respectively.

According to Algorithmic 4 and 5, l_{ca} and l_{cb} are two basic solution can be used in the computing of corresponding admissibility-assured insertion function set. Rule (a) brings a selection method which can economize valuable network resources with its best from the point of reducing unexpected releasing actions.

b. Simplification of $L_c(G)$

Rule (b). For $l_{sj}-l_{cip}$ and $l_{sj}-l_{ciq}$ in $CL(G)$, if $|l_{cip}|>|l_{ciq}|$, delete $l_{sj}-l_{cip}$ from $CL(G)$ and vice versa.

According to Algorithmic 5, there is $P_{s-h}(l_{cip}/l_{sj})=P_{s-h}(l_{ciq}/l_{sj})=l_{sj}$, which means secret string l_{sj} can be hidden and modified into l_{cip} and l_{ciq} by different insertion functions. Therefore l_{cip} and l_{ciq} are functionally equivalent. As for a l_{sj}, only one corresponding pair is necessary in $CL(G)$, the string with longer length should be deleted to reduce the times of releasing actions.

Due to rule (b) above, the $CL(G)$ is simplified as $CL(G)=[e_3e_1e_3^*-[e_3e_2]e_2e_3e_1e_3^*$, $e_3e_2e_3e_1e_2^*-[e_3e_2]e_2e_1e_3e_1e_2^*]$. As there is only one qualified candidate string remained, this $CL(G)$ is the only solution to enforce opacity for the SISM in Fig. 4.

Ground on Algorithm 2, 3 and Theorem1, 2, 3, there is the safety-assured insertion function set $F_{IS}=[f_{is}(\varepsilon,e_1)]=[(e_{3i}e_{2i}e_1)]$ and the admissibility-assured insertion function set $F_{IA}=[f_{ia}(\varepsilon,\ e_3),\ f_{ia}(e_3,e_1),\ f_{ia}(e_3e_1,e_3),\ f_{ia}(e_3\ \ e_1e_3^*,e_3),\ f_{ia}(e_3,\ \ e_2),\ \ f_{ia}(e_3e_2,\ \ e_3),\ \ f_{ia}(e_3e_2e_3,e_1),\ \ f_{ia}(e_3e_2e_3e_1,e_2),\ f_{ia}(e_3e_2e_3e_1e_2^*,e_2)]=[(e_3),\ \ (e_3e_{2i}e_{2i}e_{3i}e_1),\ \ (e_3e_{2i}e_{2i}e_{3i}e_1e_3),\ (e_3e_{2i}e_{2i}e_{3i}e_1e_3^*),\ \ (e_3e_2),\ \ (e_3e_2e_{2i}e_{1i}e_3),\ \ (e_3e_2e_{2i}e_{1i}e_3e_1),\ (e_3e_2e_{2i}e_{1i}e_3e_1e_2),\ (e_3e_2e_{2i}e_{1i}e_3e_1e_2^*)]$. And the insertion function for each unsafe string in G is confirmable listed in Table 2.

The result shows that the power failure state (x_3) and reconfiguration mission (x_4-x_8) is hidden by normal ship motion control mission safety and admissibility. This SISM is "I_2–*Enforceability*".

5. Conclusion and future work

In this paper, we proposed a dual layer mechanism based on two types of insertion function named Safety-assured insertion function (f_{IS}) and Admissibility-assured insertion function (f_{IA}) to enforce opacity for ship information system where f_{IS} is designed to confuse intruder by constructing a non-secret behaviors from a unsafe one, and the division of f_{IA} is to polish the modified output behaviors back to normal. We define the property of "I_2–*Enforceability*" that dual layer insertion functions has the ability to enforce opacity. Given a SIS mission model, we propose an algorithm to select f_{IS} and compute its matchable f_{IA}. As it is a large and heavy computing work to test the safety and admissibility of each possible insertion function, dislike the opacity enforceability research before, the paramount consideration in this paper is for a non-safe output behavior sl_{sj} which could be modified as safe string $l_{ci}l_{cik}$. Then based on the unsafe string sl_{sj} and its acceptable modification $l_{ci}l_{cik}$, the necessary extra activated DCUs during each secret mission process are able to be determined by the result of $F_{IS-IA}(G-l_{ci})$. Meanwhile, several principles are proposed in the compare of all possible solution to find the easiest method to enforce opacity. The computing process is easy to implement and high-efficiency.

Based on the structure of dual insertion function proposed in this paper, we are interested in the following research points.

5.1. Enforcing opacity for a system with unfixed secret

In previous research, the secret is always given and fixed. However, for some intruders, their interest points may be changed based on the system's previous process (internal incentive). For example, the activation of fire lader is a secret only if some weapons has already activated. Meanwhile, SIS would revise its guard point under certain external circumstances (external incentive, such as cyber-attack, etc.). In view of the above-mentioned incentives, these systems should have the adaptability to protect their unfixed secrets.

5.2. Imperfect information releasing issue in multi-task mode

In this paper, we assume that a DCU will send all required information in a single data flow. However, if the length of each data flow is fixed and limited, it is impossible to add all necessary information at once. Imperfect information releasing issue should be considered.

Acknowledgments

This work was supported by the China Scholarship Council., under the grant 201406680015.

References

Ben-Kalefa, M., Lin, F., 2011. Supervisory control for opacity of discrete event systems. In: Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on, IEEE, pp. 1113–1119.

Bryans, J.W., Koutny, M., Mazaré, L., Ryan, P.Y., 2008. Opacity generalised to transition systems. Int. J. Inf. Secur. 7 (6), 421–435.

Cassez, F., Dubreil, J., Marchand, H., 2012. Synthesis of opaque systems with static and dynamic masks. Formal Methods Syst. Des. 40 (1), 88–115.

Dubreil, J., Darondeau, P., Marchand, H., 2010. Supervisory control for opacity, automatic control. IEEE Trans. 55 (5), 1089–1100.

Falcone, Y., Marchand, H., 2015. Enforcement and validation (at runtime) of various notions of opacity. Discret. Event Dyn. Syst. 25 (4), 531–570.

Jacob, R., Lesage, J.-J., Faure, J.-M., 2015. Opacity of discrete event systems: models, validation and quantification. IFAC-PapersOnLine 48 (7), 174–181.

Kumari, S., Khan, M.K., 2014. More secure smart card-based remote user password authentication scheme with user anonymity. Secur. Commun. Netw. 7 (11), 2039–2053.

Liang, Y., Poor, H.V., et al., 2009. Information theoretic security. Found. Trends Commun. Inf. Theory 5 (4–5), 355–580.

Lin, F., 2011. Opacity of discrete event systems and its applications. Automatica 47 (3), 496–503.

Liu, S., Xing, B., Li, B., Gu, M., 2014. Ship information system: overview and research trends. Int. J. Nav. Archit. Ocean Eng. 6 (3), 670–684.

Mazaré, L., 2004. Using unification for opacity properties. In: Proceedings of the 4th IFIP WG1 7, pp. 165–176.

Rabbachin, A., Conti, A., Win, M.Z., 2015. Wireless network intrinsic secrecy. Netw. IEEE ACM Trans. 23 (1), 56–69.

Takai, S., Oka, Y., 2008. A formula for the supremal controllable and opaque sublanguage arising in supervisory control. SICE J. Control Meas. Syst. Integr. 1 (4), 307–311.

Wang, D., Wang, P., 2014. On the anonymity of two-factor authentication schemes for wireless sensor networks: attacks, principle and solutions. Comput. Netw. 73, 41–57.

Wu, Y.-C., Lafortune, S., 2012. Enforcement of opacity properties using insertion functions. In: Decision and Control (CDC), 2012 IEEE 51st Annual Conference on. IEEE, pp. 6722–6728.

Wu, Y.-C., Lafortune, S., 2013. Comparative analysis of related notions of opacity in centralized and coordinated architectures. Discret. Event Dyn. Syst. 23 (3), 307–339.

Wu, Y.-C., Lafortune, S., 2014. Synthesis of insertion functions for enforcement of opacity security properties. Automatica 50 (5), 1336–1348.

Xing, B., Liu, S., Zhu, W., 2015. Actuator channel setting strategy for ship information systems based on reachability analysis and physical characteristic. In: Environment and Electrical Engineering (EEEIC), 2015 IEEE 15th International Conference on. IEEE, pp. 932–937.

Xing, B., Liu, S., Chen, X., Zhi, P.. Design of sensor data flow for ship information system. Submitted to J. Ship Prod. Des..