SECURITY OF CYBER-PHYSICAL SYSTEMS AGAINST ACTIVE AND PASSIVE NETWORK ATTACKS: A DISCRETE EVENT SYSTEM APPROACH

Publio Macedo Lima

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia Elétrica.

Orientadores: Marcos Vicente de Brito Moreira
Lilian Kawakami Carvalho

Rio de Janeiro
Agosto de 2021

SECURITY OF CYBER-PHYSICAL SYSTEMS AGAINST ACTIVE AND PASSIVE NETWORK ATTACKS: A DISCRETE EVENT SYSTEM APPROACH

Publio Macedo Lima

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Orientadores: Marcos Vicente de Brito Moreira
Lilian Kawakami Carvalho

Aprovada por: Prof. Marcos Vicente de Brito Moreira
Profª. Lilian Kawakami Carvalho
Prof. José Eduardo Ribeiro Cury
Prof. Antonio Eduardo Carrilho da Cunha
Prof. Claudio Miceli de Farias
Prof. Gustavo da Silva Viana

RIO DE JANEIRO, RJ – BRASIL
AGOSTO DE 2021

Lima, Publio Macedo

Security of Cyber-Physical Systems Against Active and Passive Network Attacks: A Discrete Event System Approach/Publio Macedo Lima. – Rio de Janeiro: UFRJ/COPPE, 2021.

XII, 85 p.: il.; 29, 7cm.

Orientadores: Marcos Vicente de Brito Moreira
Lilian Kawakami Carvalho

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2021.

Referências Bibliográficas: p. 79 – 85.

1. Security. 2. Discrete event systems. 3. Cyber-Physical Systems. I. Moreira, Marcos Vicente de Brito *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Agradecimentos

Gostaria de agradecer aos meus pais, Mario Jorge Lima e Maria Antônia Macedo Lima, pelo suporte financeiro e emocional. Também gostaria de agradecer a minha irmã, Nádia Helena Lima de Menezes, meus irmãos, Hudson Macedo Lima e Túlio Macedo Lima, e minha namorada, Tamara Bunheirão Monteiro, pelo suporte emocional.

Gostaria de agradecer também aos meus orientadores, Marcos Vicente de Brito Moreira e Lilian Kawakami Carvalho, que estiveram comigo durante a pós-graduação e me ajudaram a chegar a esse ponto.

Agradeço ao Conselho Nacional de Desenvolvimento Tecnológico e Científico (CNPq) pelo suporte financeiro.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

SEGURANÇA DE SISTEMAS CIBER-FÍSICOS CONTRA ATAQUES ATIVOS E PASSIVOS EM REDES: UMA ABORDAGEM DE SISTEMAS A EVENTOS DISCRETOS

Publio Macedo Lima

Agosto/2021

Orientadores: Marcos Vicente de Brito Moreira
Lilian Kawakami Carvalho

Programa: Engenharia Elétrica

Um dos maiores desafios para aplicações de sistemas ciber-físicos (SCF) em indústrias inteligentes é garantir a segurança contra ataques cibernéticos, que podem ser classificados como ativos ou passivos. Neste trabalho é considerado que um SCF pode ser descrito como um sistema a eventos discretos (SED). Do ponto de vista de ataques ativos, são considerados ataques do tipo man-in-the-middle nos canais de sensor e/ou canais de controle, em que o atacante pode ler, apagar, inserir ou modificar dados nos canais atacados. Para proteger contra esse tipo de ataque é proposta a criação de dois módulos de segurança: (i) um baseado em detecção do ataque e; (ii) um módulo maximamente permissível, ou seja, que restringe o mínimo possível o funcionamento do sistema atacado. Para mitigar ataques passivos, em que o atacante espiona o canal de comunicação do sistema, é proposto um novo esquema de criptografia baseado em eventos. Além disso, é definida a propriedade de confidencialidade para SEDs, associada à capacidade de garantir que somente o emissor e o receptor de uma mensagem transmitida sejam capazes de entendê-la. Condições necessárias e suficientes para garantir essa propriedade são apresentadas, bem como métodos para gerar uma criptografia satisfazendo a confidencialidade do sistema.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

SECURITY OF CYBER-PHYSICAL SYSTEMS AGAINST ACTIVE AND PASSIVE NETWORK ATTACKS: A DISCRETE EVENT SYSTEM APPROACH

Publio Macedo Lima

August/2021

Advisors: Marcos Vicente de Brito Moreira
Lilian Kawakami Carvalho

Department: Electrical Engineering

One of the most important challenges for the application of cyber-physical systems (CPS) in smart industries is ensuring its security against cyber attacks that can be classified as passive or active. In this work it is considered that the CPS can be abstracted as a Discrete-Event System (DES). Relating to active attacks, the protection against man-in-the-middle attacks in the sensor and/or control channel are considered, where the attacker can read, delete, insert or modify data in attacked channels. In order to protect from this type of attack it is proposed the creation of two security module: (i) a module based on the detection of attacks, and (ii) a maximally permissive module, *i.e.*, a module that restricts the minimum possible amount the operation of the attacked system. In order to mitigate passive attacks, where the attacker eavesdrops the communication channel of the system, it is proposed a new cryptographic scheme based on events. We also define the property of confidentiality of DESs, which is associated with the capability of ensuring that only the sender and the intended receiver are able to understand the transmitted message. Necessary and sufficient conditions to ensure this property, and methods to generate the cryptographic schemes ensuring confidentiality of the system are presented.

Contents

List of Figures

List of Tables

List of Symbols

F_E	Event-Based Encryption Function, p. 61
F_T	Transition-Based Encryption Function, p. 63
G	Deterministic automaton, p. 8
$G_1 \parallel G_2$	Parallel composition between automata G_1 and G_2, p. 13
$L(G)$	Language generated by automaton G, p. 8
L/s	Post language of L after s, p. 8
$Obs(G, \Sigma_o)$	Observer of automaton G considering the set of observable events Σ_o, p. 10
$P_o(s)$	Natural Projection of sequence s, p. 8
$UR(x)$	Unobservable reach of a state x, p. 10
Υ	Attackable Networked System, p. 19
$\mathcal{E}(G)$	Current-state estimator of automaton G, p. 11
$\Gamma_G(x)$	Feasible event function of on G in state x, p. 9
$\Sigma^\star$	Kleene-closure of Σ, p. 7
Σ_c	Set of controllable events, p. 9
Σ_o	Set of observable events, p. 9
Σ_{uc}	Set of uncontrollable events, p. 9
Σ_{uo}	Set of unobservable events, p. 9
$\|s\|$	The length of a sequence s, p. 7
$\mathcal{G}$	Nondeterministic automaton, p. 8
ε	Empty sequence, p. 7

Chapter 1

Introduction

Security in the context of communication networks has been addressed in the literature with different definitions. In the context of Information Technology (IT), it is defined as the protection of assets (PFLEEGER and PFLEEGER, 2002), the collection of tools designed to protect data (STALLINGS, 2006), or the guarantee of certain specifications (KUROSE and ROSS, 2011). Moreover, three desired properties for security communications are defined as: Confidentiality, Integrity and Availability, denoted as the CIA triad (KUROSE and ROSS, 2011; PFLEEGER and PFLEEGER, 2002; STALLINGS, 2006). Confidentiality represents that only the sender and the intended receiver should be able to understand the contents of the transmitted message. Integrity assures that messages are received as sent, with no duplication, insertion, modification, reordering, or replays. Availability is defined as the property of a system, or a system resource, being accessible and usable upon demand by an authorized system entity. These properties are also defined in the X-800 and the RFC2828 standards (SHIREY, 2000; X-800, 1991). Aside from these three properties for communication security, KUROSE and ROSS (2011) also define operational security, which is associated with the protection of the operation of a system, such as preventing harm in industrial sites.

Most computer security activity is designed to prevent malicious human-caused harm (PFLEEGER and PFLEEGER, 2002). Since malicious agents want to cause harm, it is used the term attack for malicious actions. Attacks can be classified in two major categories: passive attacks and active attacks (WU *et al.*, 2007). An attack is classified as passive if it attempts to learn or make use of information from the system without affecting system resources. On the other hand, an active attack attempts to affect their operation or alter system resources (STALLINGS, 2006).

The advance of globalization brought new challenges to the industry, forcing processes to become more efficient and flexible. From these requirements a new concept was created for the industry, denoted by Smart Factory, or alternatively, denoted by Industry 4.0 (GILCHRIST, 2016; WANG *et al.*, 2016). This concept is used

to designate the current tendency in automation systems to exchange informations in manufacture technologies. With the new concept of smart factories, automation systems are more than ever using network communications, even in the lower levels of the industry (GUBBI *et al.*, 2013). The use of this type of communication in Automation Technology (AT) networks also increases the vulnerability to cyber attacks (DA XU *et al.*, 2014; GOU *et al.*, 2013; HE *et al.*, 2016; SINGH and SINGH, 2015). Therefore, in the context of AT, security is becoming a major concern (BENCSÁTH *et al.*, 2012; COUTO *et al.*, 2020; FARWELL and ROHOZINSKI, 2011; MOHAMMAD and LAKSHMISRI, 2018).

The framework used for Industry 4.0 is based on Cyber-Physical systems (CPS) (JIRKOVSKY *et al.*, 2017), which are systems that integrate computing and communication capabilities to monitor and control physical processes (CASSANDRAS, 2016; LEE, 2008). It is also important to remark, that the security of CPS can be addressed by considering different model formulations such as: (i) discrete linear time-invariant systems MO and SINOPOLI (2010, 2012); (ii) continuous linear time-invariant systems HUANG and DONG (2018); (iii) discrete event systems; or (iv) considering the communication network properties, *e.g.,* considering the physical layer JADOON *et al.* (2021), or proposing the creation of a demilitarized zone (DMZ) COUTO *et al.* (2020). Several overviews of security for cyber-physical systems have been presented in the literature (ASHIBANI and MAHMOUD, 2017; CAO *et al.*, 2020; HUMAYED *et al.*, 2017; IGURE *et al.*, 2006).

In this work, the system's logical behavior of the CPS is modeled as a Discrete Event System (DES), which are systems with discrete state space, and whose evolution is driven by the occurrence of events (CASSANDRAS and LAFORTUNE, 2008). Events are representations of instantaneous occurrences, such as the pressing of a button or a sensor state change. DESs can be used to model several systems, such as industrial networks, manufacturing systems, traffic supervision, logistic, or scheduling problems.

Several works in the literature propose strategies to model and thwart active attacks in automation networks using the framework of Discrete Event Systems (CARVALHO *et al.*, 2018; GOES *et al.*, 2017, 2020; LIMA *et al.*, 2017; SU, 2018; THORSLEY and TENEKETZIS, 2006; ZHANG *et al.*, 2018), and address the problem of ensuring operational security, which can be interpreted as the prevention of damages caused to the system. The first work that addresses active attacks in the context of Stochastic Discrete Event Systems is presented by THORSLEY and TENEKETZIS (2006), where the main objective is to design a supervisor that achieves the specification in normal operation and after an attack. The attacks considered in THORSLEY and TENEKETZIS (2006) can be interpreted as an interference in the communication channel between the supervisor and the system, that could be

caused by an intruder attempting to damage the system. Latter, in SU (2018), a robust supervisor is designed to enforce the control specification language under sensor channel attacks, where it is considered that the attacker can replace the observation of an event with a bounded sequence of events. In SU (2018), attacks in actuator channels are not considered. In GOES *et al.* (2017, 2020) and ZHANG *et al.* (2018) the authors consider stealthy attacks in sensors. The authors, in LIMA *et al.* (2017) and CARVALHO *et al.* (2018), deal with attacks in both sensors and actuators. Both in CARVALHO *et al.* (2018) and in LIMA *et al.* (2017), the authors use a conservative strategy, in which all controllable events are disabled when the attack is detected and it can lead the system to an unsafe state, *i.e.*, an state that if reached presents a treat to the system.

In the context of DESs, passive attacks has been addressed as the problem of opacity (BARCELOS and BASILIO, 2018, 2021; GUO *et al.*, 2020; JI, 2019; LIN, 2011; MAZARÉ, 2004; SABOORI and HADJICOSTIS, 2007; WU and LAFORTUNE, 2014). Opacity is defined as the capability of hiding a secret. This secret can be a secret language (Language-based opacity) in a partial observation system such that the attacker must always be unsure if a sequence in the secret language has occurred, or a set of states (State-based opacity) where the attacker must be unsure if the system is or has been in a secret state, based on the observation transmitted by the system. The State-based opacity include initial-state opacity (SABOORI and HADJICOSTIS, 2008), current-state opacity (SABOORI and HADJICOSTIS, 2013; TONG *et al.*, 2018), k-step opacity (SABOORI and HADJICOSTIS, 2011b) and infinite-step opacity (SABOORI and HADJICOSTIS, 2011a; YIN *et al.*, 2019). In general, a system is said to be opaque if given any secret sequence, there exists a non-secret sequence that is transmitted in the same way (WU and LAFORTUNE, 2014). In addition, the authors in LIN (2011) show that several different discrete event systems problems can also be reduced to the opacity problem, namely, anonymity, secrecy, observability, diagnosability, and detectability. In MAZARÉ (2004) it is proposed the use of similarity in cryptographic protocols to enforce opacity. It is important to remark that, in MAZARÉ (2004), the authors still assume that the transmitted data is the same for secret or non-secret sequences, in order to ensure opacity. This is not a problem if the secret information is not important for the intended receiver. In this context, emerge in the literature the notions of utility and privacy (WU *et al.*, 2018), where utility refers to the generation of sequences of events that are useful by the intended receiver and must not be confused with other sequences, and privacy refers to hiding a secret behavior from an intruder for example. However, if the useful information for the receiver must also be kept secret from an intruder, then it is necessary to introduce a new notion that encompasses both properties.

One method to ensure the security of data is cryptography. In the context of DESs, cryptography is considered in FRITZ and ZHANG (2018); FRITZ *et al.* (2019) in order to improve the network security. In FRITZ and ZHANG (2018) the authors consider two types of active attacks, namely, replay attack and covert attack, and propose a method for identifying an attacker by changing the transition matrices of the Petri net model. However, this implementation may increase the complexity of the transmission, since it needs the synchronization of the sender and the receiver. In FRITZ *et al.* (2019) the authors relate to the passive attack problem and adapt the Somewhat Homomorphic Encryption (SWHE) scheme (DYER *et al.*, 2017). The method proposed in FRITZ *et al.* (2019) replaces every transmitted bit with a large integer, due to the public-key encryption proposed using large prime numbers and relating to Euler's totient function (LEHMER *et al.*, 1932) to avoid cryptanalysis, and therefore, making the transmission more expensive.

In this thesis, we present discussions and solutions for problems regarding cyber attacks in industrial networks for active and passive attacks. We initially address the problem of protecting the system against active cyber attacks, for which a model of the attacked system is presented and a defense strategy is proposed, where a security supervisor is created in order to prevent the system from reaching unsafe states. In this regard, a security supervisor that disables all controllable events if the system is under a detected attack and an unsafe state may be reached is proposed. The property of NA-Safe Controllability is also proposed, which is related to the existence of a security supervisor capable of protecting the system. However, this approach may not be maximally permissive, *i.e.*, may not disable events only when necessary to prevent damage. For this reason, we also propose the creation of another security supervisor, that disables only the events that leads the system to states where damage cannot be prevented. This approach is related to the supervisor range control problem, which is a known problem in the supervisor control theory and can be solved, if a solution exists, in exponential time. We define then, a class of systems, called NA-Secure Systems, for which a solution for a maximally permissive security supervisor can be computed in polynomial time.

In this thesis, we also address the problem of security of CPSs against passive attacks in the network. In this regard, a model for an eavesdropping attacker estimation is presented. We also present the property of confidentiality of discrete event systems, related to the capability of the system to prevent the attacker from correctly estimating the occurrence of secret sequences in the system from the transmitted message. In order to do so, we propose a defense strategy based on cryptography for the transmission in an industrial communication channel, where the message, can be modified by an encryption function before transmission and recovered by the authorized receiver. We present a method for verification of confidentiality of DES

given an encryption function, and present a method for the creation of an encryption function that ensures confidentiality of DESs for a given secret language with bounded sequences.

This thesis is organized as follows, in Chapter 2 we present some basic concepts used throughout the thesis. We address the problem of security in CPSs against active attacks in Chapter 3, where we consider a detection based approach in Section 3.3. Then, in Section 3.4, we then propose a maximally permissive approach. In Chapter 4 We propose an event-based cryptographic scheme to mislead an attacker that knows the internal model of the system, and define the property of Confidentiality for discrete event systems. Finally in Chapter 5 the conclusions are drawn.

Chapter 2

Basic concepts

In this work we propose methods for improving the security of cyber-physical systems (CPS) modeled as discrete-event systems (DES). Thus, in Section 2.1 we introduce the notations and the background of DES used in this work. Since we also use concepts of Information Technology (IT) to improve the security of CPS, we present some background on the security of networks in Section 2.2.

2.1 Discrete event systems

Discrete event systems can be modeled by using different formalisms, such as Petri nets and automata (CASSANDRAS and LAFORTUNE, 2008; HOPCROFT *et al.*, 2006; LAWSON, 2003). In this work, all systems are modeled by automata. Thus, in this section we present the models of a deterministic and a nondeterministic automaton. We also present the language of these automata, and some basic operations using them.

Before introducing automata, it is important to define languages, which is a formal way to represent DESs (CASSANDRAS and LAFORTUNE, 2008). DESs are associated with an event set Σ, that can be interpreted as an "alphabet", such that the concatenation of events forms sequences that can be interpreted as "words" of a language (notice that in the literature a sequence is also called "string" or "trace") (CASSANDRAS and LAFORTUNE, 2008). Thus, a language is a set of sequences formed with the elements of Σ.

The length of a sequence s is the number of events that form it, counting multiple occurrences of the same event, and it is denoted by $|s|$. The sequence with zero length is called empty sequence and is denoted by ε. The language formed of all sequences of finite length obtained from Σ plus the empty sequence ε is call the kleen-closure of Σ denoted by $\Sigma^\star$.

Definition 1 (Language) *(CASSANDRAS and LAFORTUNE, 2008) A language defined over an event set Σ is a set of finite length sequences formed with events in Σ.* □

Notice that, according to definition 1, any language L defined over Σ is a subset of $\Sigma^\star$, *i.e.*, $L \subseteq \Sigma^\star$.

A sequence s can be partitioned as $s = tuv$, where t and v are called a prefix and a suffix of s, respectively. Notice that ε is always a prefix and a suffix of any sequence. We can also define the post language after a sequence s in a language L, L/s, as the set formed of the continuations of all sequences of L after s, *i.e.*, $L/s = \{t \in \Sigma^\star : st \in L\}$.

The set of all possible prefixes of the strings in a language $L \subseteq \Sigma^\star$ is called prefix-closure of L, and is defined as $\overline{L} = \{s \in \Sigma^\star : (\exists t \in \Sigma^\star)[st \in L]\}$.

The natural projection is an important operation defined for sequences. Let $\Sigma_o \subset \Sigma$ be a set of events. Then, the projection $P_o : \Sigma^\star \to \Sigma_o^\star$ is defined as $P_o(\varepsilon) = \varepsilon$, $P_o(\sigma) = \sigma$, if $\sigma \in \Sigma_o$ or $P_o(\sigma) = \varepsilon$, if $\sigma \in \Sigma \setminus \Sigma_o$, and $P_o(s\sigma) = P_o(s)P_o(\sigma)$, for all $s \in \Sigma^\star$ and $\sigma \in \Sigma$. Notice that the projection $P_o(s)$ erases events that do not belong to Σ_o in $s \in \Sigma^\star$. The projection operation can be extended to languages by applying them to all sequences in the language (CASSANDRAS and LAFORTUNE, 2008), $P_o(L) = \{t \in \Sigma_o^\star : (\exists s \in L)[P_o(s) = t]\}$.

In the sequel we present the definition of a deterministic automaton.

Definition 2 *A deterministic automaton is a five-tuple, $G = (X, \Sigma, f, x_0, X_m)$, where X is the set of states, Σ is the finite set of events, $f : X \times \Sigma \to X$ is the transition function, $x_0 \in X$ is the initial state of the system, and $X_m \subseteq X$ is the set of marked states.*

In this work we will not consider the set of marked states, *i.e.*, we will define $X_m = \emptyset$ for all automata, and therefore, a deterministic automaton can be represented by the four-tuple $G = (X, \Sigma, f, x_0)$.

The domain of the transition function f can be extended to $X \times \Sigma^*$, where $\Sigma^\star$, as usual: $f(x, \varepsilon) = x$, and $f(x, s\sigma) = f(f(x, s), \sigma)$, for all $s \in \Sigma^*$ and $\sigma \in \Sigma$. The language generated by G is defined as $L(G) = \{s \in \Sigma^\star : f(x, s) \text{ is defined}\}$.

Definition 3 *A nondeterministic automaton is the five-tuple $\mathcal{G} = (X, \Sigma \cup \{\varepsilon\}, f_{nd}, x_0, X_m)$, where the elements of $\mathcal{G}$ have the same interpretation as in the deterministic automaton G, with the exception of the nondeterministic transition function $f_{nd} : X \times \Sigma \cup \{\varepsilon\} \to 2^X$. The initial set of states x_0 in a nondeterministic automaton can be a subset of the set of states X.*

A nondeterministic automaton can be represented by the four-tuple $\mathcal{G} = (X, \Sigma \cup \{\varepsilon\}, f_{nd}, x_0)$ when there is no marked states, *i.e.*, $X_m = \emptyset$.

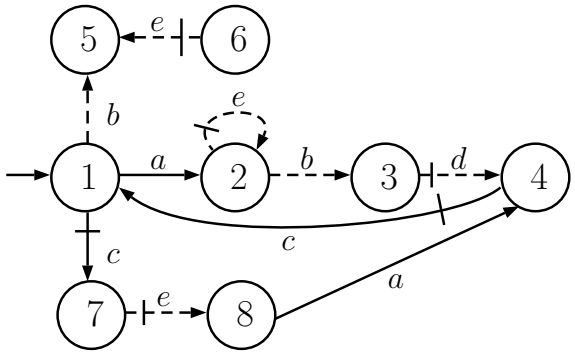

Figure 2.1: Example 1: Automaton G.

In order to define the language generated by $\mathcal{G}$, it is necessary to extend the domain of f_{nd} to $X \times \Sigma^*$, obtaining the extended transition function f^e_{nd}. Let $\varepsilon R(x)$ denote the ε-reach of a state x, *i.e.*, the set of states reached from x by following transitions labeled with ε, including state x (CASSANDRAS and LAFORTUNE, 2008). The ε-reach can be extended to a set of states $B \subseteq X$ as $\varepsilon R(B) = \cup_{x \in B} \varepsilon R(x)$. The extended nondeterministic transition function $f^e_{nd} : X \times \Sigma^* \to 2^X$, can be defined recursively as $f^e_{nd}(x, \varepsilon) = \varepsilon R(x)$, and $f^e_{nd}(x, s\sigma) = \varepsilon R[\{z : z \in f_{nd}(y, \sigma) \text{ for some state } y \in f^e_{nd}(x, s)\}]$. Thus, the language generated by $\mathcal{G}$ can be defined as $L(\mathcal{G}) = \{s \in \Sigma^* : (\exists x \in x_0)[f^e_{nd}(x, s) \text{ is defined}]\}$.

The feasible event function of an automaton $\mathcal{G}$ is defined as $\Gamma_{\mathcal{G}} : X \to 2^{\Sigma}$, where $\Gamma_{\mathcal{G}}(x) = \{\sigma \in \Sigma : f_{nd}(x, \sigma) \text{ is defined}\}$, for all $x \in X$.

A path of length k_1 of G is defined as $p = (x_1, \sigma_1, x_2, \ldots, \sigma_{k_1-1}, x_{k_1})$, where $x_i \in X$, for $i = 1, \ldots, k_1$, $\sigma_i \in \Sigma$, for $i = 1, \ldots, k_1 - 1$, and $f(x_i, \sigma_i) = x_{i+1}$, for $i = 1, \ldots, k_1 - 1$. The concatenation of two paths p and $p' = (x'_1, \sigma'_1, x'_2, \ldots, x'_{k_2})$, where $x'_i \in X$, for $i = 1, \ldots, k_2$, and $\sigma'_i \in \Sigma$, for $i = 1, \ldots, k_2 - 1$, is possible if the last state of p is equal to the first state of p', *i.e.*, $x_{k_1} = x'_1$. The concatenation operation is defined as $p \cdot p' = (x_1, \sigma_1, x_2, \ldots, \sigma_{k_1-1}, x_{k_1}, \sigma'_1, \ldots, \sigma'_{k_2-1}, x'_{k_2})$.

The set of events Σ can be partitioned in two different manners, as $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$, where Σ_c and Σ_{uc} are, respectively, the sets of controllable and uncontrollable events of the system, and also as $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, where Σ_o and Σ_{uo} denote, respectively, the sets of observable and unobservable events of the system (CASSANDRAS and LAFORTUNE, 2008). With a view to facilitating the visualization of the controllable and/or observable events of the system, we adopt in this work the following graphical representation for the arcs in the state transition diagram of an automaton: (*i*) observable events are associated with solid arcs, and unobservable events with dashed arcs; (*ii*) controllable events are associated with arcs with a tick, and uncontrollable events are those associated with arcs without ticks.

Example 1 *Let us consider the automaton* $G = (X, \Sigma, f, x_0)$*, depicted in Figure 2.1. In this case,* $X = \{1, 2, 3, 4, 5, 6, 7, 8\}$, $\Sigma = \{a, b, c, d, e\}$, $f(1, a) =$

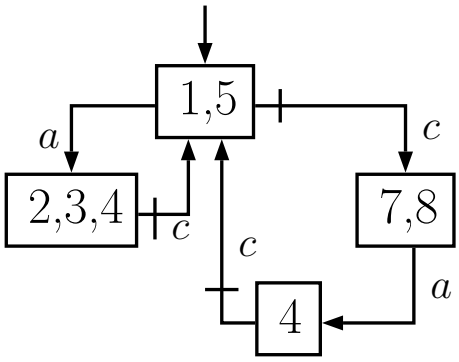

Figure 2.2: Example 2: Observer $G_{obs} = Obs(G, \Sigma_o)$.

$2, f(1,b) = 5, f(1,c) = 7, f(2,b) = 3, f(2,e) = 2, f(3,d) = 4, f(4,c) = 1, f(6,e) = 5, f(7,e) = 8, f(8,a) = 4$, *and* $x_0 = 1$, *and* $X_m = \emptyset$, *and therefore not represented. Moreover,* Σ *can be partitioned as* $\Sigma_o = \{a, c\}$ *and* $\Sigma_{uo} = \{b, d, e\}$, *or as* $\Sigma_c = \{c, d, e\}$ *and* $\Sigma_{uc} = \{a, b\}$. *In addition the function* Γ_G *can be defined from* f *as* $\Gamma_G(1) = \{a, b, c\}, \Gamma_G(2) = \{b, e\}, \Gamma_G(3) = \{d\}, \Gamma_G(4) = \{c\}, \Gamma_G(5) = \emptyset, \Gamma_G(6) = \{e\}, \Gamma_G(7) = \{e\},$ *and* $\Gamma_G(8) = \{a\}$. □

It is important to remark that it is always possible to compute from a nondeterministic automaton $\mathcal{G}$, a deterministic automaton whose generated language is equal to $P_o(L)$, where $L = L(\mathcal{G})$ and P_o is computed considering the set of unobservable events Σ_{uo}. We adopt, in this work, the observer automaton presented in CASSANDRAS and LAFORTUNE (2008), denoted as $Obs(\mathcal{G}, \Sigma_o) = (X_{obs}, \Sigma_o, f_{obs}, x_{0,obs})$. In order to compute G_{obs}, we first need to define the operation of obtaining the unobservable reach of a state $x \in X$, which is a generalization of the notion of $\varepsilon - reach$ (CASSANDRAS and LAFORTUNE, 2008).

Definition 4 (Unobservable reach) *The unobservable reach of a state* $x \in X$, *denoted by* $UR(x)$, *is defined as:*

$$UR(x) = \{y \in X : (\exists t \in \Sigma_{uo}^{\star})(f(x,t) = y)\}. \tag{2.1}$$

The unobservable reach can also be defined for a set of states $B \in 2^X$ *as:*

$$UR(B) = \bigcup_{x \in B} UR(x). \tag{2.2}$$

The observer automaton can be then computed by following the steps of algorithm 1.

Example 2 *Let us consider once again automaton G from Example 1, depicted in Figure 2.1, where* $\Sigma_o = \{a, c\}$. *By following the steps from algorithm 1 is computed the observer automaton* $G_{obs} = Obs(G, \Sigma_o)$, *depicted in Figure 2.2.* □

Algorithm 1: Observer automaton

Input : $G = (X, \Sigma, f, x_0)$, Σ_o.

Output: Observer automaton $G_{obs} = (X_{obs}, \Sigma_o, f_{obs}, x_{0,obs})$.

1 Define $x_{0,obs} = UR(x_0)$.

2 $X_{obs} = \{x_{0,obs}\}$ and $\widetilde{X}_{obs} = X_{obs}$.

3 $\bar{X}_{obs} = \widetilde{X}_{obs}$, $\widetilde{X}_{obs} = \emptyset$.

4 **for** *each* $B \in \bar{X}_{obs}$ **do**

5 $\quad \Gamma_{G_{obs}}(B) = \left(\bigcup_{x \in B} \Gamma_G(x)\right) \cap \Sigma_o$.

6 $\quad$ **for** *each* $\sigma \in \Gamma_{G_{obs}}(B)$ **do**

7 $\quad\quad f_{obs}(B, \sigma) = UR(\{x \in X : (\exists y \in B)[x = f(y, \sigma)]\})$.

8 $\quad$ **end**

9 $\quad \widetilde{X}_{obs} \leftarrow \widetilde{X}_{obs} \cup f_{obs}(B, \sigma)$.

10 **end**

11 $X_{obs} \leftarrow X_{obs} \cup \widetilde{X}_{obs}$.

12 Repeat steps 3 to 11 until all accessible part of G_{obs} is constructed.

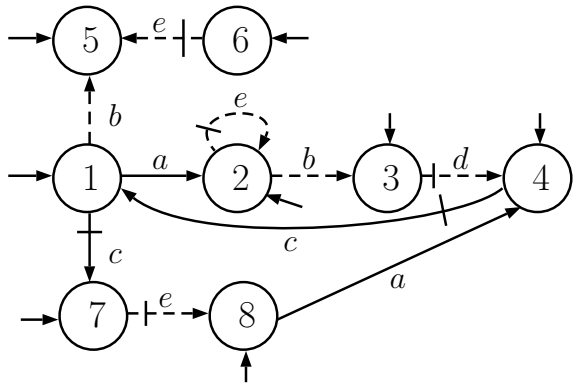

Figure 2.3: Example 3: Modified automaton $\mathcal{G}$.

We can then compute the current-state estimator of an automaton G, denoted as $\mathcal{E}(G) = (X_{\mathcal{E}}, \Sigma, f_{\mathcal{E}}, x_{0,\mathcal{E}})$ SABOORI and HADJICOSTIS (2011a). The current-state estimator of $G = (X, \Sigma, f, x_0)$ is computed in two steps: (i) obtain the nondeterministic automaton $\mathcal{G} = (X, \Sigma, f, X)$ from G, by defining the initial state of $\mathcal{G}$ as the set X; and (ii) compute $\mathcal{E}(G) = Obs(\mathcal{G}, \Sigma)$.

Example 3 *Let us consider again automaton $G = (X, \Sigma, f, x_0)$ from Example 1, depicted in Figure 2.1. We can then compute the current-state estimator of G, by firstly computing automaton $\mathcal{G} = (X, \Sigma, f, X)$, depicted in Figure 2.3, and then by following the steps from algorithm 1 considering all events as observable $\mathcal{E}(G) = Obs(\mathcal{G}, \Sigma)$, depicted in Figure 2.4.* □

The uncontrollable reach of a state $x \in X$ with respect to an automaton $\mathcal{G}$ and a set of uncontrollable events Σ_{uc} is defined as $R_{uc}(x, \mathcal{G}, \Sigma_{uc}) = \{y \in X : (\exists t \in \Sigma_{uc}^{\star})[f_{nd}^{e}(x, t) = y]\}$. For a set of states $B \subseteq X$, the uncontrollable reach is extended as $R_{uc}(B, \mathcal{G}, \Sigma_{uc}) = \bigcup_{x \in B} R_{uc}(x, \mathcal{G}, \Sigma_{uc})$.

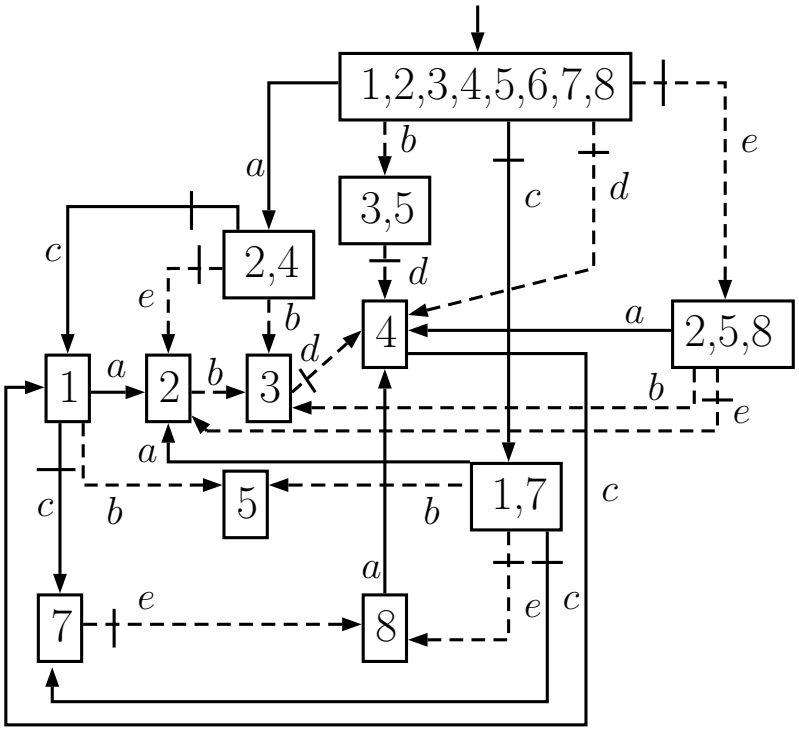

Figure 2.4: Example 3: Estimator $\mathcal{E}(G)$.

The unary operation of taking the accessible part of an automaton is defined as follows.

Definition 5 *The operation of taking the accessible part of an automaton* $G = (X, \Sigma, f, x_0)$ *is given by:*

$$Ac(G) := (X_{ac}, \Sigma, f_{ac}, x_0)$$

where

$$X_{ac} = \{x \in X : (\exists s \in \Sigma^\star)[f(x_0, s) = x]\},$$

$$f_{ac} = f \mid_{X_{ac} \times \Sigma \to X_{ac}},$$

where $f_{ac} = f \mid_{X_{ac} \times \Sigma \to X_{ac}}$, *denotes the transition function* f *restricted to domain* $X_{ac} \times \Sigma$.

Example 4 *Let us consider Automaton* G *from Example 1. The accessible part of Automaton* G *is then presented in Figure 2.5. Notice that, state* 6 *is not accessible since there is no sequence* $s \in \Sigma^*$ *that leads the system from the initial state* 1 *to state* 6*, and therefore, it is removed from the accessible automaton* $Ac(G)$. □

Let $G_1 = (X_1, \Sigma_1, f_1, x_{0,1})$ and $G_2 = (X_2, \Sigma_2, f_2, x_{0,2})$ be two deterministic automata. Then, the following parallel composition operation can be defined CASSANDRAS and LAFORTUNE (2008).

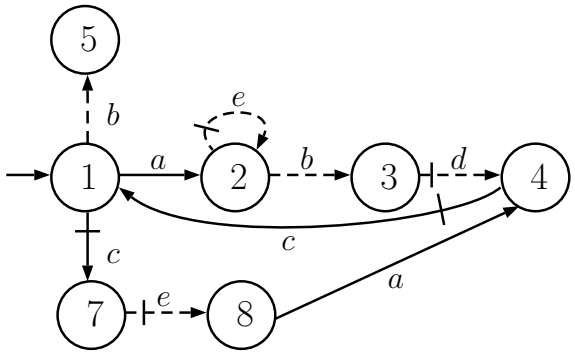

Figure 2.5: Example 4: Accessible part of automaton G, $Ac(G)$.

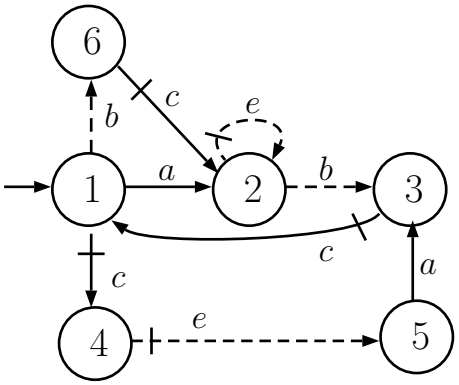

Figure 2.6: Example 5: Automaton G_1.

Definition 6 *The parallel composition of G_1 and G_2 is given by:*

$$G_1 \parallel G_2 = Ac(X_1 \times X_2, E_1 \cup E_2, f_{1\parallel 2}, (x_{0,1}, x_{0,2}))$$

where

$$f_{1\parallel 2}((x_1, x_2), \sigma) := \begin{cases} (f_1(x_1, \sigma), f_2(x_2, \sigma)), \text{ if } \sigma \in \Gamma_{G_1}(x_1) \cap \Gamma_{G_2}(x_2) \\ (f_1(x_1, \sigma), x_2), \text{ if } \sigma \in \Gamma_{G_1}(x_1) \setminus \Sigma_2 \\ (x_1, f_2(x_2, \sigma)), \text{ if } \sigma \in \Gamma_{G_2}(x_2) \setminus \Sigma_1 \\ \text{undefined, otherwise.} \end{cases}$$

Example 5 *Let us consider automata G_1 and G_2 depicted in Figure 2.6 and 2.7 respectively. Then the parallel composition $G = G_1 \parallel G_2$ is depicted in Figure 2.8. It is important to remark that, State $(2,2)$ of G represents that the current state of automaton G_1 is 2, and since event e is not defined in G_2 and is feasible in state 2 of G_1, then e is also feasible in state $(2,2)$ of G. On the other hand, state $(6,5)$ of G represents that the current state of automaton G_1 is 6 and the current state of automaton G_2 is 5. From automaton G_1 $\Gamma(6) = \{c\}$ and from automaton G_2 $\Gamma(5) = \{a\}$, then, since a and c are defined in both G_1 and G_2, no events are feasible in state $(6,5)$ of automaton G.* □

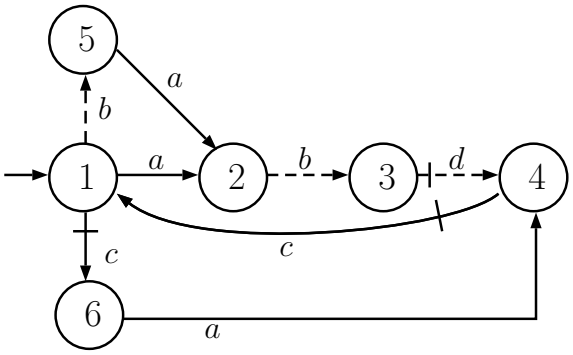

Figure 2.7: Example 5: Automaton G_2.

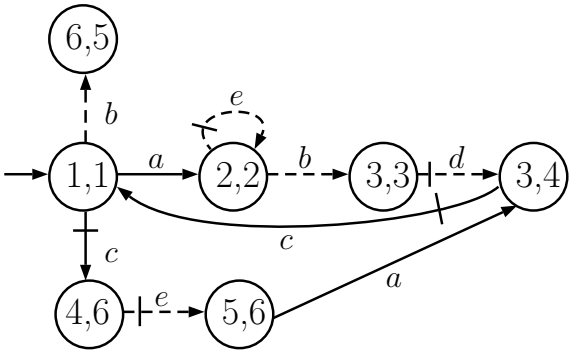

Figure 2.8: Example 5: Parallel composition $G = G_1 \parallel G_2$.

The reversed automaton of $\mathcal{G} = (X, \Sigma \cup \{\varepsilon\}, f_{nd}, x_0)$ is the automaton obtained by reversing all of its transitions, formally defined as $\mathcal{G}^r = (X, \Sigma \cup \{\varepsilon\},\ f^r_{nd}, x^r_0)$, where $y \in f^r_{nd}(x, \sigma)$ if, and only if, $x \in f_{nd}(y, \sigma)$, for all $x, y \in X$ and $\sigma \in \Sigma \cup \{\varepsilon\}$, and x^r_0 is undefined WU and LAFORTUNE (2013).

Example 6 *Considering automaton G from example 5, depicted in Figure 2.8. The reverse automaton G^r is depicted in Figure 2.9. Notice that, in the reversed automaton the initial state is undefined.* □

Let T be a rooted tree. Then, the *height* of T is the maximum length of a path that starts at the root and proceeds downward through descendants to a leaf. The

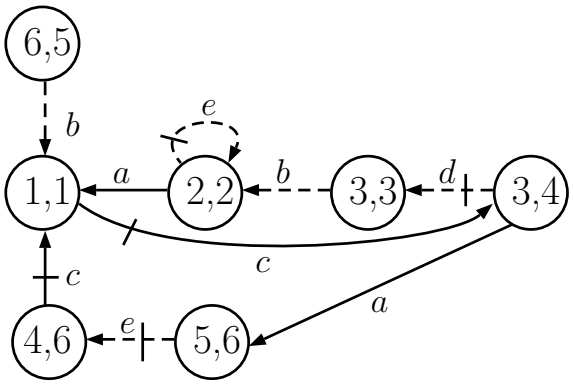

Figure 2.9: Example 6: Reverse automaton G^r.

depth of a node y of T, is the length of the simple path from the root to y COMER (2009); HOPCROFT *et al.* (2006).

2.2 Information Security

The basic principles of security for CPSs are confidentiality, integrity, and availability KUROSE and ROSS (2011). These concepts are better described in the sequel.

- Confidentiality

  Confidentiality refers to the capability of the system to prevent unauthorized entities to access information transmitted in a channel, more specifically, a system is confidential if only the sender and the intended receiver can understand the data transmitted in a channel. Since eavesdroppers can intercept messages transmitted in a channel, in order to ensure confidentiality, some kind of encryption must be used KUROSE and ROSS (2011).

- Integrity

  Integrity refers to data and resources of the system not being modified or deleted without authorization. Integrity is violated when an attack or a fault modify information transmitted in a channel.

- Availability

  Availability refers to any resource of the system being available when requested by an authorized entity. Availability is in general compromised by denial of service (DOS) attacks. In this work we do not consider availability problems.

The concept of cryptography is widely used in IT to improve security of systems KUROSE and ROSS (2011); PFLEEGER and PFLEEGER (2002); STALLINGS (2006). In the sequel, we present the background of cryptography necessary to understand this work.

In this work, We denote an original message by plain text and a ciphered message by cipher text, and the method to convert a plain text into a cipher text is called encryption. The method to recover a plain text from a cipher text is called decryption. Thus, cryptography can be defined as the study of methods of encryption and decryption of messages STALLINGS (2006).

The messages transmitted from the sender to the receiver are modified in the cryptographic system using a secret information. We call this secret information 'key' STALLINGS (2006). The study of retrieving the secret key by an attacker is called cryptanalysis. In general, cryptanalysis uses some knowledge of the algorithm

or the system to recover the key. When a cryptanalysis try every possible key in order to recover the plain text, it is called brute-force attack STALLINGS (2006).

Cryptography is classified into three independent dimensions STALLINGS (2006). Firstly, it can be classified by the way it processes a plain text, where cryptographic systems are divided in stream ciphers and block ciphers. Stream ciphers are encryption methods that cipher one element of the plain text at a time, *i.e.*, it processes the plain text continuously. On the other hand, in block ciphers, the system processes a block of plain text at a time, *i.e.*, it waits a certain amount of inputs before processing the plain text. Secondly, cryptographic systems can be classified by the number of keys used, *i.e.*, if the algorithm to encrypt uses the same key as the decryption algorithm, the system is classified as symmetric cipher, or single-key cipher. However, if the key to encrypt is different from the key to decrypt, the system is known as asymmetric cipher, or public-key cipher. Finally, based on the algorithms used, it is considered that encryption algorithms are based on two general principles, transposition, in which elements in the plain text are rearranged, and substitution, in which each element in the plain text is mapped into another element. A system can also be classified as a product system if its algorithm uses both substitution and transposition. Independently on the classification, it is fundamental for a cryptographic system that no information is lost, *i.e.*, the encryption must be revertible.

Chapter 3

Security against active communication network attacks

Initially, we consider the problem of intrusion detection and prevention of damages caused by attacks in the context of supervisory control of Discrete Event Systems (DESs). In LIMA *et al.* (2017) and LIMA (2017), we proposed automaton models of the plant and supervisor subject to man-in-the-middle attacks in the sensor and/or communication channels. In addition, we propose the use of a module to ensure the security of the system, *i.e.*, a module capable of preventing the system from reaching states that may cause harm to the system without altering the behavior of the non-attacked system. This module is called security supervisor.

In Section 3.3, the property of NA-Safe controllability, that is related with the capability of detecting intrusions and preventing damages caused by man-in-the-middle attacks, is introduced, and a verification algorithm was proposed. In LIMA *et al.* (2017) it is not presented how the security supervisor can be implemented in order to detect and prevent damages caused by cyber-attacks. In Section 3.3, we extended the work presented in LIMA *et al.* (2017) as follows: (*i*) we prove the correctness of the NA-Safe controllability verification algorithm proposed in LIMA *et al.* (2017); (*ii*) we show how to implement the security supervisor against attacks in the communication channels of CPSs; (*iii*) we show that NA-Safe controllability is a necessary and sufficient condition for the existence of the security supervisor; and (*iv*) we present a practical example, that has not been used in LIMA *et al.* (2017), to illustrate the results of the work. The results presented in Section 3.3 are published in LIMA *et al.* (2019).

In Section 3.4, we propose a new method for the design of the security supervisor, where we add the objective of the security supervisor to be maximally permissive, *i.e.*, the security supervisor must disable events only when it is necessary to prevent the system from being damaged. Then, a class of system in which the maximally permissive security supervisor can be computed in polynomial time complexity is

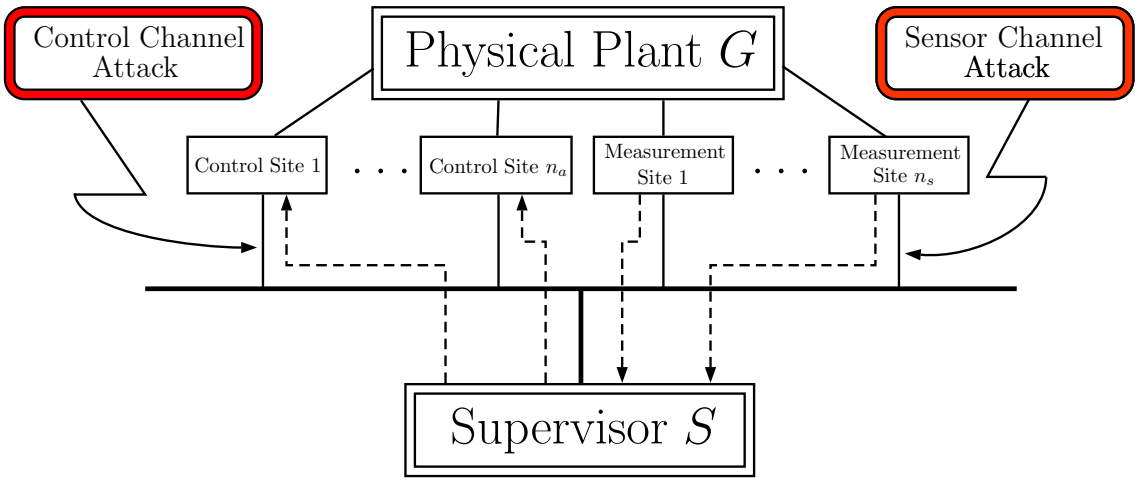

Figure 3.1: Attackable Networked System (ANS).

presented, called NA-Secure Systems. A verification method for confirming if a system is NA-Secure is presented, and a method for the online implementation of a maximally permissive securty supervisor is proposed. Finally, the complexity of the algorithms proposed in this section are discussed. The results presented in Section 3.4 are published in LIMA *et al.* (2021).

This chapter is organized as follows. In Section 3.1, we present models for the attacked system. In Section 3.2 we formulate the problem of intrusion detection and prevention of damages caused by man-in-the-middle attacks in CPSs. In Section 3.3, we define NA-Safe controllability, and present the verification algorithm with its proof of correctness. In Section 3.3.3, we present a necessary and sufficient condition for the existence of the security supervisor, and a method for its online implementation. We also present in Section 3.3.3 a practical example, from modeling to implementation of the security supervisor. In Section 3.4 we present the condition of maximally permissibility. In Section 3.4.1 we define NA-Security, and present an algorithm to verify this property. In Section 3.4.2, we present the algorithm to implement the security supervisor for the class of NA-Secure systems. In Section 3.4.3, we present the computational complexity of the methods proposed. Finally, in Section 3.5, the conclusions of this chapter are drawn.

3.1 Communication Network Attacks

3.1.1 System and Security Structure

In this chapter, we consider a cyber-physical system composed of a physical plant controlled by a supervisor as shown in Figure 3.1. We assume that the communication between supervisor and plant is carried out by using several different sensor channels and supervisory control channels. The plant is modeled by a deterministic

automaton $G = (X, \Sigma, f, x_0)$, the realization of the supervisor S is modeled by a deterministic automaton $H = (X_H, \Sigma, f_H, x_{0H})$, and the closed-loop system model $T = (X_T, \Sigma, f_T, x_{0T})$ is obtained by the parallel composition $T = G \| H$. We also refer to unsafe states, denoted by $X_{US} \subset X$, where unsafe states are states of the plant that represent risk to operators or equipments, or can be associated with an undesirable behavior. We assume that the supervisor is designed to avoid the plant from reaching any unsafe state $x \in X_{US}$.

Let us consider that at least one communication channel in the networked supervisory control system of Figure 3.1 is vulnerable to man-in-the-middle attacks, *i.e.*, the attacker can observe, hide, create or change the information that transits in the attacked communication channel. Let us also assume that both sensor communication channels and supervisory control communication channels can be attacked. Thus, the intruder can hide, insert or change events whose occurrence is detected by sensors in the plant, and can modify the enabling action commanded by the supervisor to the actuators of the plant, with the objective of driving the system to reach unsafe states. We denote the sensor channels and the supervisory control channels vulnerable to man-in-the-middle attacks, respectively, as ch_{s_i} and ch_{a_j}, for $i = 1, \ldots, n_s$ and $j = 1, \ldots, n_a$, where n_s is the number of vulnerable sensor channels and n_a is the number of vulnerable supervisory control channels. The set of observable events transmitted through channel ch_{s_i} is denoted as $\Sigma_{s_i} \subset \Sigma_o$ and the set of controllable events enabled through channel ch_{a_j} is denoted as $\Sigma_{a_j} \subset \Sigma_c$. Then, the set of events associated with the vulnerable sensor channels is defined as $\Sigma_{vs} = \bigcup_{i=1}^{n_s} \Sigma_{s_i}$, and the set of events associated with the vulnerable supervisory control channels is defined as $\Sigma_{va} = \bigcup_{j=1}^{n_a} \Sigma_{a_j}$.

We formally define an Attackable Networked System (ANS) as follows.

Definition 7 *An Attackable Networked System is a tuple* $\Upsilon = (G, H, \Sigma_{vs}, \Sigma_{va})$, *where* G *is the automaton model of the plant,* H *is the realization of supervisor* S, *and* Σ_{vs} *and* Σ_{va} *are the sets of events associated with the vulnerable sensor channels and the vulnerable supervisory control channels, respectively.* □

The following assumption is considered for the correct modeling of the plant and supervisor under network attacks.

A1. *The sets of controllable and observable events are disjoint, i.e.,* $\Sigma_o \cap \Sigma_c = \emptyset$.

Notice that Assumption **A1** is not restrictive since any controllable and observable event σ can be represented as a sequence $\sigma_c \sigma_o$, where σ_c is controllable and unobservable, and σ_o is observable and uncontrollable. In this case, event σ_c represents the occurrence of event σ and σ_o represents the successful communication to supervisor S of the occurrence of σ as shown in LIMA (2017).

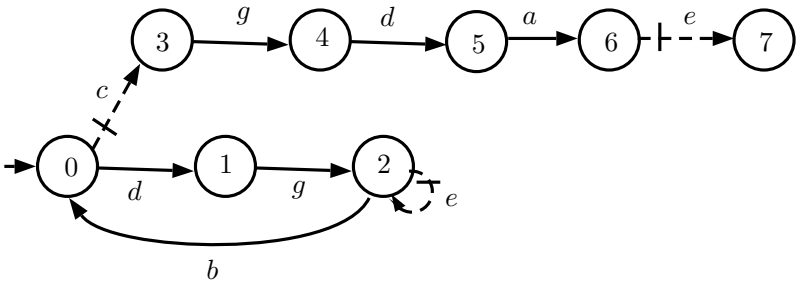

Figure 3.2: Model of the plant G.

3.1.2 Model of the plant subject to sensor channel attacks

In this section, we propose a nondeterministic automaton $\mathcal{G}_a$, obtained from automaton G and the set of events associated with the vulnerable sensor channels Σ_{vs}, that generates all possible sequences modified by the attacks in the sensor channels LIMA *et al.* (2018).

Definition 8 (Model of the plant subject to sensor channel attacks) *Let $G = (X, \Sigma, f, x_0)$ and Σ_{vs} be the plant automaton and the set of events associated with vulnerable sensor channels of an ANS Υ, respectively. Then, a nondeterministic automaton that models all possible observations of the events generated by the plant after sensor channel attacks is given by $\mathcal{G}_a = (X, \Sigma \cup \{\varepsilon\}, f_{\mathcal{G}_a}, x_0)$, where $f_{\mathcal{G}_a}$ is defined, for all $x \in X$ and $\sigma \in \Sigma \cup \{\varepsilon\}$, as:*

$$f_{\mathcal{G}_a}(x,\sigma) = \begin{cases} \{f(x,\sigma)\},\ \textit{if } \sigma \in \Gamma_G(x) \cap (\Sigma \setminus \Sigma_{vs}), \\ \{x\} \cup \{f(x,\sigma)\},\ \textit{if } \sigma \in \Gamma_G(x) \cap \Sigma_{vs}, \\ \{x\},\ \textit{if } \sigma \in \Sigma_{vs} \setminus \Gamma_G(x), \\ \{f(x,\sigma_v) : \sigma_v \in \Gamma_G(x) \cap \Sigma_{vs}\},\ \textit{if } \sigma = \varepsilon, \\ \textit{undefined, otherwise.} \end{cases}$$ □

In Definition 8, self-loops are introduced in each state of the plant G, for simulating the ability of the attacker to create a new instance of observation for any attacked event. Transitions labeled with ε are added in parallel to the transitions labeled with attacked events, to simulate the ability to delete the observation of an event. Notice that the ability to modify the observation of vulnerable events is equivalent to deleting one event and creating another one. Thus, the modification of the observation of an event is modeled by the self-loops and ε-transitions added to G.

Example 7 *Consider an ANS $\Upsilon = (G, H, \Sigma_{vs}, \Sigma_{va})$ and $X_{US} = \{7\}$. Consider that G is the automaton model of the plant, shown in Figure 3.2, where $\Sigma = \{a, b, c, d, e, g\}$ and, $\Sigma_o = \{a, b, d, g\}$, and consider that only one sensor channel, that transmits the observation of event g to the supervisor, is attacked, i.e.*

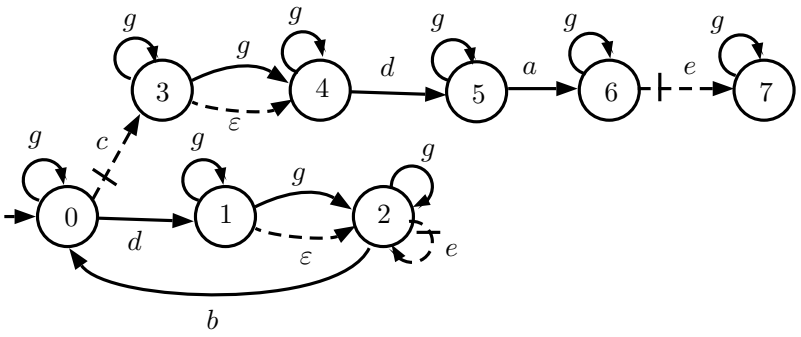

Figure 3.3: Model of the plant subject to sensor channel attacks $\mathcal{G}_a$.

$\Sigma_{vs} = \{g\}$. *Then, we can obtain, in accordance with Definition 8, the model of the plant subject to sensor channel attacks* $\mathcal{G}_a$ *shown in Figure 3.3.*

It is important to notice that $\mathcal{G}_a$ *is a nondeterministic automaton, since it models the false observation of events, when the plant does not change its state (modeled by the self-loops labeled with events in* Σ_{vs}*), and also the addition of transitions labeled with* ε *in parallel with all transitions labeled with an event in* Σ_{vs}*, to represent the deletion of observation of events by the intruder.* □

3.1.3 Realization of the supervisor subject to supervisory control channel attacks

We consider that the intruder has the ability to enable or disable vulnerable controllable events associated with the actuators of the plant. Thus, in order to model the attacks in the supervisory control channels of an ANS $\Upsilon = (G, H, \Sigma_{vs}, \Sigma_{va})$, we modify realization H of supervisor S by allowing the modifications that the intruder can execute in the vulnerable controllable events of Σ_{va}. It is important to remark that when the attacker disables controllable events, it only restricts the closed-loop behavior of the system, since no new behavior of the attacked plant $\mathcal{G}_a$ can be generated. Hence, no unsafe state can be reached. Thus, in the problem considered in this work, it is not important to model the disabling actions of the attack CARVALHO *et al.* (2018); LIMA *et al.* (2018).

Definition 9 (Supervisor realization model after control attacks) *Let* $H = (X_H, \Sigma, f_H, x_{0H})$ *and* Σ_{va} *be the realization of supervisor* S *and the set of events associated with vulnerable supervisory control channels of an ANS* Υ*, respectively. A deterministic automaton that models supervisor* S *in the presence of control channel attacks is defined as* $H_a = (X_H, \Sigma, f_{H_a}, x_{0H})$*, where* f_{H_a} *is defined, for all* $x \in X_H$

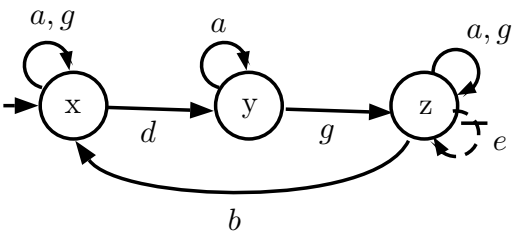

Figure 3.4: Supervisor realization H.

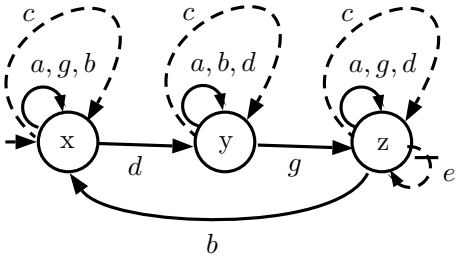

Figure 3.5: Realization of the supervisor after control channel attacks H_a.

and $\sigma \in \Sigma$*, as:*

$$f_{H_a}(x,\sigma) = \begin{cases} f_H(x,\sigma), \text{ if } \sigma \in \Gamma_H(x), \\ x, \text{ if } \sigma \in (\Sigma_{va} \cup \Sigma_{uc}) \setminus \Gamma_H(x), \\ \text{undefined, otherwise.} \end{cases} \qquad \square$$

In Definition 9, we compute the supervisor of an attackable networked system where the malicious agent has the ability to enable vulnerable events, represented by the introduction of self-loops labeled with events in Σ_{va}. Notice that, since the malicious agent can enable attacked events in Σ_{va}, these events become uncontrollable. In addition, since the supervisor must be admissible, we add to all states $x \in X_H$, self-loops labeled with all uncontrollable events $\sigma \notin \Gamma_H(x)$.

Example 8 *Consider the same ANS* $\Upsilon = (G, H, \Sigma_{vs}, \Sigma_{va})$ *of Example 7, whose plant* G *is shown in Figure 3.2, where* $\Sigma_c = \{d, e\}$ *and* $\Sigma_o = \{a, c\}$*, and supervisor realization* $H = (X_H, \Sigma, f_H, x_{0H})$ *is depicted in Figure 3.4. Let* $\Sigma_{va} = \{c\}$ *be the set of vulnerable controllable events. Then, using Definition 9, we obtain the realization of the supervisor subject to control attacks* H_a *shown in Figure 3.5. Notice that* H_a *is obtained from* H *by adding self-loops in the states* $x \in X_H$*, labeled with the events in* Σ_{va} *and* Σ_{uc} *that are not feasible in* x*.* $\square$

3.1.4 Automaton model of the attackable networked system

Given an ANS $\Upsilon = (G, H, \Sigma_{vs}, \Sigma_{va})$. The behavior of Υ in the presence of man-in-the-middle attacks is modeled by nondeterministic automaton $\mathcal{T}_a = \mathcal{G}_a \| H_a = (X_{\mathcal{T}_a}, \Sigma \cup \{\varepsilon\}, f_{\mathcal{T}_a}, x_{0,\mathcal{T}_a})$, referred here to as attacked closed-loop system model.

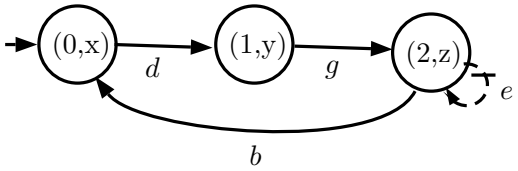

Figure 3.6: Automaton for the closed-loop system (T).

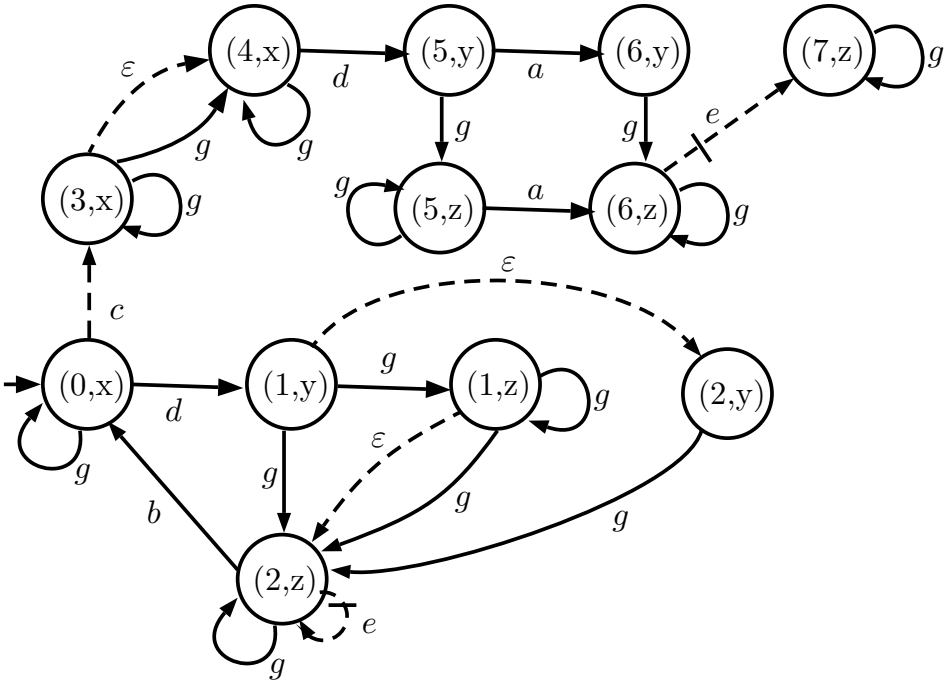

Figure 3.7: Automaton of the closed-loop system subject to network attacks $\mathcal{T}_a$.

Since the malicious agent can enable attacked events in Σ_{va}, these events become uncontrollable. This leads to the following definition of the set of non-vulnerable controllable events $\Sigma_{ca} = \Sigma_c \setminus \Sigma_{va}$, and of uncontrollable events $\Sigma_{uca} = \Sigma_{uc} \cup \Sigma_{va}$ of the attacked closed-loop system model $\mathcal{T}_a$. In addition, the set of unsafe states of $\mathcal{T}_a$ is defined as $X_{US}^{\mathcal{T}_a} = \{(x, y) \in X_{\mathcal{T}_a} : x \in X_{US}\}$.

Example 9 *Consider once again the ANS $\Upsilon = (G, H, \Sigma_{vs}, \Sigma_{va})$ of Example 7, where G be depicted in Figure 3.2, where $\Sigma_o = \{a, b, d, g\}$ and $X_{US} = \{7\}$, and the supervisor realization H be presented in Figure 3.4. Notice that there is no unsafe state in the closed-loop system $T = G \| H$, depicted in Figure 3.6, which shows that the supervisor avoids the reach of unsafe states as expected.*

Consider now that $\Sigma_{vs} = \{g\}$, and $\Sigma_{va} = \{c\}$. Then, the attacked closed-loop system $\mathcal{T}_a = \mathcal{G}_a \| H_a$, depicted in Figure 3.7, has the set of unsafe states $X_{US}^{\mathcal{T}_a} = \{(7, z)\}$. Namely, the intruder is capable of driving the system to reach an unsafe state after cyber-attacks in the channels that communicate the occurrence of g, and the enablement of c. □

3.1.5 Unsafe Region and Unsafe Boundary

In the sequel, we introduce the concepts of unsafe region and unsafe boundary for an ANS, considering the associated attacked closed-loop system model $\mathcal{T}_a$, as follows.

Definition 10 *Let $\mathcal{T}_a$ be the attacked closed-loop system model, where Σ_{ca} and $X_{US}^{\mathcal{T}_a}$ are the sets of controllable events and unsafe states of $\mathcal{T}_a$, respectively. Then, the unsafe region $US_R \subseteq X_{\mathcal{T}_a}$ is defined as $US_R = R_{uc}(X_{US}^{\mathcal{T}_a}, \mathcal{T}_a^r, \Sigma_{uca})$, where $\mathcal{T}_a^r$ is the reversed automaton of $\mathcal{T}_a$.* □

According to Definition 10, for any state $x \in US_R$, there exists a sequence of uncontrollable events in $\Sigma_{uca}^{\star}$ that lead the system from x to an unsafe state. Thus, when the system reaches a state in the unsafe region, the security supervisor cannot prevent it from reaching an unsafe state.

Definition 11 *Let $\mathcal{T}_a$ be the attacked closed-loop system model, and US_R be the unsafe region of $\mathcal{T}_a$. Then, the unsafe boundary $US_B \subset X_{\mathcal{T}_a}$ is defined as $US_B = \{x \in X_{\mathcal{T}_a} \setminus US_R : (\exists \sigma \in \Sigma_{ca}, \exists y \in US_R)[y \in f_{\mathcal{T}_a}(x, \sigma)]\}$.* □

Notice, according to Definition 11, that the unsafe boundary US_B is formed of those states $x \in X_{\mathcal{T}_a}$ that are not in the unsafe region US_R, and there exists a feasible controllable event $\sigma \in \Sigma_{ca}$ that leads the system to the unsafe region US_R.

Example 10 *Let us consider once again the automaton $\mathcal{T}_a$ depicted in Figure 3.7, Considering that the set of unsafe states is $X_{US}^{\mathcal{T}_a} = \{(7, z)\}$, then the unsafe region, can be calculated as $US_R = \{(7, z)\}$. Notice that, state $(6, z)$ is not a part of the unsafe region, since the event e that leads the system from state $(6, z)$ to the unsafe state $(7, z)$ is controllable, i.e., $e \in \Sigma_{ca}$. Moreover, state $(6, z)$ is the only state in $\mathcal{T}_a$ that have a feasible event which leads the $\mathcal{T}_a$ to the unsafe region, therefore, the unsafe boundary can be computed as $US_B = \{(6, z)\}$.*

Based on Definitions 10 and 11 we can partition the language generated by the attacked closed-loop system into five languages according to the detection of the attack and the exposure of the system to threats.

Definition 12 *Let $\mathcal{T}_a$ and T be the attacked and non attacked closed-loop systems of an ANS Υ, respectively. Let, also, US_B, US_R and P_o denote, respectively, the unsafe boundary, the unsafe region and the projection that models the observation of the supervisor. Then, every sequence $s \in L(\mathcal{T}_a)$ can be classified as:*

- *s belongs to the undetectable language, denoted as L_U, if there exists $\omega \in L(T)$ such that $P_o(s) = P_o(\omega)$;*

- *s belongs to the no risk language, denoted as* L_{NR}*, if* $s \notin L_U$ *and,* $\forall t \in L(\mathcal{T}_a)/s, f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, st) \cap US_R = \emptyset$*;*
- *s belongs to the non-imminent risk language, denoted as* L_{NIR}*, if* $s \notin (L_U \cup L_{NR})$*, and* $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s) \cap (US_R \cup US_B) = \emptyset$*;*
- *s belongs to the imminent risk language, denoted as* L_{IR}*, if* $s \notin (L_U \cup L_{NR})$*,* $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s) \cap US_B \neq \emptyset$*, and* $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s) \cap US_R = \emptyset$*;*
- *s belongs to the damage language, denoted as* L_D*, if* $s \notin (L_U \cup L_{NR})$*, and* $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s) \cap US_R \neq \emptyset$*.* □

Notice that each sequence $s \in L(\mathcal{T}_a)$ belongs to a unique language presented in Definition 12, *i.e.*, $L(\mathcal{T}_a) = L_U \dot{\cup} L_{NR} \dot{\cup} L_{NIR} \dot{\cup} L_{IR} \dot{\cup} L_D$. The sequences in the undetectable language L_U represent the behavior of the system without attack, and the behavior after an attack that cannot be distinguished from a sequence in the non-attacked language $L(T)$. The no risk language L_{NR} is formed of the sequences generated after an attack that can be distinguished from the sequences in the non-attacked language, and that does not have any continuation that leads the system to an unsafe state. Thus, the occurrence of sequences in L_{NR} does not represent any risk to the system. The non-imminent risk language L_{NIR} is formed of the sequences that satisfy three conditions: (i) it can be distinguished from the sequences in the non-attacked language; (ii) there exists a continuation that leads the system to an unsafe state; and (iii) if $s \in L_{NIR}$ is executed by the system, then its current state does not belong to the unsafe boundary US_B or unsafe region US_R. On the other hand, if $s \in L_{IR}$ is executed, the attack can be detected, and the system reaches the unsafe boundary; therefore, the security supervisor must act immediately since, as it can be seen from Definition 11, the current state is the last one where damage can be prevented. Finally, if $s \in L_D$ is executed by the system, then a state in the unsafe region is reached, which means that there exists at least one sequence of uncontrollable and/or vulnerable events that leads the system to an unsafe state and, therefore, damage can no longer be prevented.

Example 11 *Consider the ANS* Υ*, where the attacked closed-loop system* $\mathcal{T}_a$ *is depicted in Figure 3.8, where* $\Sigma_o = \{a, c, d, h\}$*,* $\Sigma_{ca} = \{b, e\}$ *and* $\Sigma_{uca} = \{a, c, d, g, h\}$*, and the non-attacked closed-loop system* T*, depicted in Figure 3.9. It can be seen, from Definitions 10 and 11, that* $US_R = \{(9, 2), (9, 3)\}$ *and* $US_B = \{(8, 2), (8, 3)\}$*. In order to illustrate the sets presented in Definition 12, consider the following sequences belonging to* $L(\mathcal{T}_a)$*:* $s_1 = chg^n d$*,* $s_2 = ca$*,* $s_3 = \varepsilon gdc$*,* $s_4 = \varepsilon gdc(gdc)^n gdd$*, and* $s_5 = s_4 e$*, where* $n \in \mathbb{N}$*. Sequence* s_1 *is generated when, due to an attack in the supervisory control channel, event* g *is enabled after the occurrence of sequence*

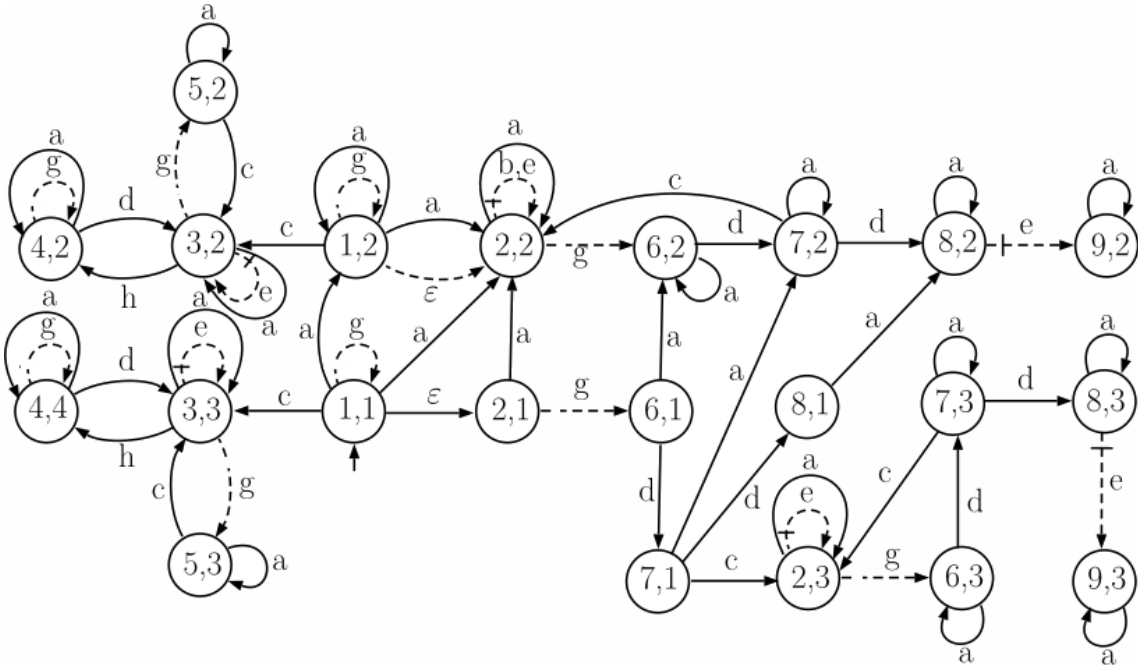

Figure 3.8: Attacked closed loop system model $\mathcal{T}_a$ of Example 11, computed considering $\Sigma_{ca} = \{b, e\}$, $\Sigma_o = \{a, c, d, h\}$.

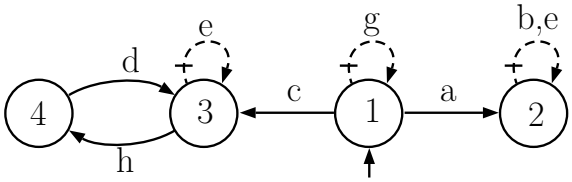

Figure 3.9: Closed loop system model T of Example 11

ch, and n occurrences of event g takes place before the occurrence of d. Such a sequence belongs to the undetectable language L_U*, since sequence* $w_1 = chd \in L(T)$ *and* $P_o(s_1) = P_o(w_1) = chd$*. Sequence* s_2 *is generated when the intruder creates a fake observation of event a after the occurrence of event c. Such a sequence does not belong to* L_U *since* $P_o(s_2) = ca \notin P_o(L(T)) = \overline{\{a\} \cup \{c\}\{hd\}^\star}$*. In addition, as it can be seen from Figure 3.8,* $\forall t_2 \in L(\mathcal{T}_a)/s_2$*,* $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s_2t_2) \cap US_R = \emptyset$*, and so,* s_2 *belongs to the no risk language* L_{NR}*. Sequence* s_3 *is neither in* L_U *nor in* L_{NR} *since* $P_o(s_3) = dc \notin P_o(L)$*, and* $t_3 = gdde \in L(\mathcal{T}_a)/s_3$ *and* $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s_3t_3) \cap US_R = \{(9,3)\} \neq \emptyset$*. However,* $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s_3) \cap (US_R \cup US_B) = \emptyset$*, and, consequently,* s_3 *belongs to the non-imminent risk language* L_{NIR}*. Finally, sequences* s_4 *and* s_5 *belong, respectively, to the imminent risk language* L_{IR} *and the damage language* L_D*, since* $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s_4) = \{(8,3)\} \subset US_B$ *and* $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s_5) = \{(9,3)\} \subset US_R$*, and* $P_o(s_4) = P_o(s_5) = (dc)^n dd \notin P_o(L(T))$*.* □

Examples 10 and 11, illustrate cases where a man-in-the-middle attack can reach the unsafe region, and therefore, damage an ANS, which shows the need for changing supervisor S or to implement a new layer of supervisors for ensuring the security of the system. There are several strategies that can be used to prevent damages caused by an attacker to the system.

Moreover, In CARVALHO *et al.* (2018), a security module, called Intrusion Detection Module, is designed to detect attacks online and, upon detection, to disable all non-vulnerable controllable events of the system. Notice that, the method proposed in CARVALHO *et al.* (2018) can be very conservative since, after detecting an attack, the system may be in a state from which it is not possible to reach an unsafe state, *i.e.*, the system may have generated a sequence $s \in L_{NR}$, and thus, no action is needed to prevent damages to the system.

By using the Intrusion Detection Module proposed in CARVALHO *et al.* (2018), an attack that inserts a sensor reading can stop the system, even when there is no risk of reaching an unsafe state. A less conservative approach is proposed in this chapter, where the definition of the safe language $L_s(\mathcal{T}_a) = \{s \in L(\mathcal{T}_a) : (\forall t \in L(\mathcal{T}_a)/s)[f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, st) \cap X^{\mathcal{T}_a}_{US} = \emptyset]\}$ is introduced. A security supervisor that disables all controllable events when the attacked closed-loop system generates an observation that does not belong to $P_o(L(T) \cup \overline{L_s(\mathcal{T}_a)})$ is presented.

3.2 Problem Formulation

In order to improve the supervisory control structure to prevent damages in the ANS caused by man-in-the-middle attacks, we propose in this work a defense strategy that aims to achieve the following objectives:

O1. While no man-in-the-middle attack occurs, the language of the closed-loop system cannot be altered, *i.e*, the language generated by the supervised system without network attacks must be equal to $L(T)$.

O2. The closed-loop system cannot reach unsafe states even when man-in-the-middle attacks take place.

Notice that the achievement of Objective **O1** guarantees that, if the system is not attacked, then the desired behavior is executed by the system. Thus, if Objective **O1** is not considered, it necessarily implies in relaxing the system specifications, which can, in practice, reduce the performance of the system in the absence of attacks.

A possible approach to ensure that an ANS does not reach unsafe states (Objective **O2**) is to replace the existing supervisor S with a new supervisor S' computed by assuming $\mathcal{G}_a$ as the plant, and Σ_o and Σ_{ca} as the sets of observable events and controllable events, respectively. However, since S' is computed by assuming that the events in Σ_{va} are uncontrollable, it never disables such events. Thus, this strategy may fail in achieving Objective **O1**, since S' may allow undesirable occurrences of sequences of events that do not belong to $L(T)$, as illustrated in the following example.

Example 12 *Consider the same ANS Υ of Examples 7 to 9, where G and H are depicted in Figures 3.2 and 3.4, respectively, $\Sigma_o = \{a, b, d, g\}$, $\Sigma_c = \{c, e\}$, $\Sigma_{vs} = \{g\}$, and $\Sigma_{va} = \{c\}$.*

It can be seen, from Figure 3.4, that supervisor S disables events c always, with a view to preventing the occurrence of any sequence in $L(G) \setminus L(T)$, where T is depicted in Figure 3.6. If we replace S with a new supervisor S' computed by assuming $\mathcal{G}_a$ as the plant, and $\Sigma_o = \{a, b, d, g\}$ and $\Sigma_{ca} = \Sigma_c \setminus \Sigma_{va} = \{e\}$ as the sets of observable and controllable events, respectively, then, since event c became uncontrollable after the cyber attack, S' never disables c. As a consequence, S' allows the execution of sequences of $L(G)$ that do not belong to $L(T)$, even in the case that no attack occurs, as, for example, sequences c and $cgda$. □

Since replacing the existing supervisor S with a new supervisor S' may not be effective to achieve Objective **O1**, we propose the implementation of the supervisory control structure depicted in Figure 3.10, in which a new supervisor, called security supervisor, denoted by Ψ, is connected to the existing supervisor S of the ANS $\Upsilon = (G, H, \Sigma_{vs}, \Sigma_{va})$. The security supervisor Ψ has the same event observation as supervisor S, and a controllable event can occur in the plant if it is neither disabled by supervisor S nor disabled by security supervisor Ψ. It is important to remark that since the intruder can attack supervisory control channels, then only non-vulnerable controllable events can be disabled by the security supervisor. Therefore, the security supervisor is defined as a mapping $\Psi : P_o(L(\mathcal{T}_a)) \to 2^{\Sigma_{ca}}$,

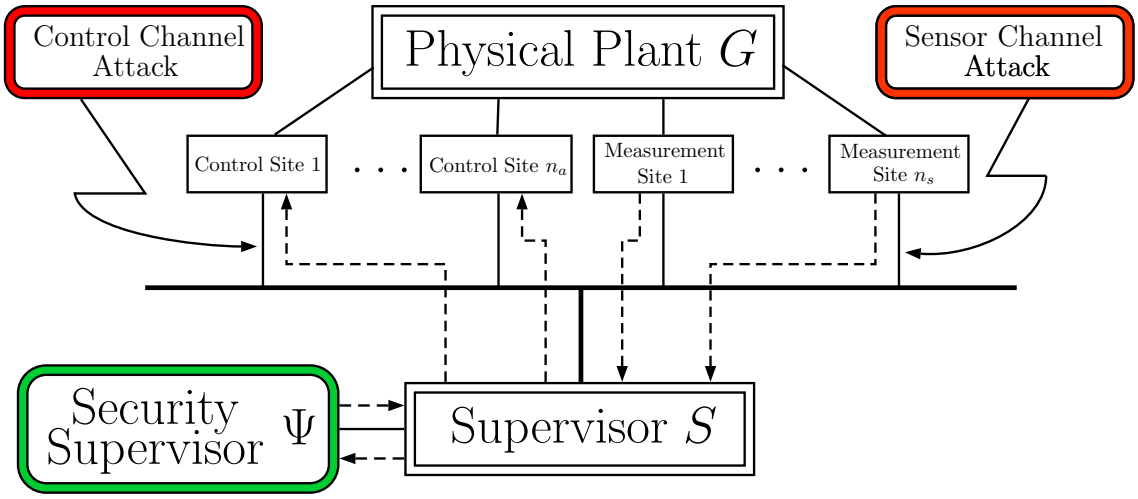

Figure 3.10: Security supervisor for an ANS.

where, for a given sequence $s \in L(\mathcal{T}_a)$, $\Psi(P_o(s))$ are the events that are disabled by the security supervisor after the occurrence of s. Thus, the language generated by the ANS modeled by $\mathcal{T}_a$ under the restrictions imposed by Ψ, denoted by $L(\Psi/\mathcal{T}_a)$, is recursively defined as: *(i)* $\varepsilon \in L(\Psi/\mathcal{T}_a)$, and *(ii)* $\forall s \in L(\mathcal{T}_a)$ and $\forall \sigma \in \Sigma$, $s\sigma \in L(\Psi/\mathcal{T}_a) \Leftrightarrow s \in L(\Psi/\mathcal{T}_a) \wedge s\sigma \in L(\mathcal{T}_a) \wedge \sigma \notin \Psi(P_o(s))$.

In Section 3.3, we present an necessary and sufficient condition for the existence of a security supervisor that is capable of ensuring Objectives **O1** and **O2**.

3.3 An intrusion detection based approach

3.3.1 NA-Safe Controllability

NA-Safe controllability definition

Let us consider an ANS $\Upsilon = (G, H, \Sigma_{vs}, \Sigma_{va})$, where $\mathcal{T}_a$ is the automaton model of the attacked system, then, $L(\mathcal{T}_a) = L_s(\mathcal{T}_a)\dot{\cup}L_{us}(\mathcal{T}_a)$ denote the language generated by $\mathcal{T}_a$, where $L_s(\mathcal{T}_a)$ denotes the safe language, and $L_{us}(\mathcal{T}_a)$ denotes the unsafe language of the system. $L_s(\mathcal{T}_a)$ is composed of traces $s \in L(\mathcal{T}_a)$ such that it is not possible to reach an unsafe state in $X_{US}^{\mathcal{T}_a}$ after the occurrence of s, *i. e.*, $L_s(\mathcal{T}_a) = \{s \in L(\mathcal{T}_a) : (\forall t \in L(\mathcal{T}_a)/s)[f_{\mathcal{T}_a}^e(X_{0,\mathcal{T}_a}, st) \cap X_{US}^{\mathcal{T}_a} = \emptyset]\}$. $L_{us}(\mathcal{T}_a)$ is composed of traces $s \in L(\mathcal{T}_a)$ such that it is possible to reach an unsafe state in $X_{US}^{\mathcal{T}_a}$ after the occurrence of s, *i.e.*, $L_{us}(\mathcal{T}_a) = \{s \in L(\mathcal{T}_a) : (\exists t \in L(\mathcal{T}_a)/s)[f_{\mathcal{T}_a}^e(X_{0,\mathcal{T}_a}, st) \cap X_{US}^{\mathcal{T}_a} \neq \emptyset]\}$. Notice that a trace in the unsafe language may be part of the non-attacked behavior of the system, *i.e.*, it may belong to the language generated by T, $L(T)$, or even be part of $\overline{L}_s(\mathcal{T}_a)$. Therefore, the security supervisor must act only after distinguishing traces of $L(\mathcal{T}_a)$ that will certainly reach an unsafe state, and do not belong to the language generated by the system before an attack $L(T)$. This leads to the following

definition of NA-Safe Controllability (LIMA *et al.*, 2017).

Definition 13 (NA-Safe Controllability) *$L(\mathcal{T}_a)$ is said to be NA-Safe Controllable with respect to T, $P_o : \Sigma^\star \to \Sigma_o^\star$, Σ_{ca} and the set of unsafe states $X_{US}^{\mathcal{T}_a}$ if*

$(\forall s \in L(\mathcal{T}_a))[f_{\mathcal{T}_a}^e(X_{0,\mathcal{T}_a}, s) \cap X_{US}^{\mathcal{T}_a} \neq \emptyset] \Rightarrow (s = s_1 s_2)[(\forall \omega \in L(T) \cup \overline{L}_s(\mathcal{T}_a))[P_o(s_1) \neq P_o(\omega)] \wedge (\Sigma_{ca} \in s_2)]$. □

According to Definition 13, $L(\mathcal{T}_a)$ is NA-Safe Controllable if all traces $s \in L(\mathcal{T}_a)$, that lead the system to an unsafe state in $X_{US}^{\mathcal{T}_a}$, can be divided as $s = s_1 s_2$, such that: *(i)* $s_1 \in L_{us}(\mathcal{T}_a)$ can be distinguished from any trace of $L(T)$ and $\overline{L}_s(\mathcal{T}_a)$; and *(ii)* $s_2 \in \Sigma^\star$ has an event from Σ_{ca}. These two conditions allow that unsafe traces that certainly lead the system to an unsafe state be detected, and the reach of unsafe states prevented by disabling the controllable events of Σ_{ca}.

Notice that, a supervisor could be designed to disable all controllable events of the plant after detecting the observation of traces that do not belong to $P_o(L(T))$. However, this approach would be more restrictive than the approach proposed in this section, since the system can execute safe traces in $\overline{L}_s(\mathcal{T}_a)$ after an attack that do not represent danger to the system.

3.3.2 NA-Safe controllability verification

In order to present an algorithm for the verification of NA-Safe controllability, we need to define two functions, namely, the rename function and the uncontrollable reach function. In addition, it is necessary to define the renaming function ρ as follows: $\rho(\sigma) = \sigma$, if $\sigma \in \Sigma_o$, and $\rho(\sigma) = \sigma_\rho$, if $\sigma \in \Sigma_{uo}$.

Let $\mathcal{G} = (X, \Sigma, f_{nd}, x_0, X_m)$ be a nondeterministic automaton. The uncontrollable reach is defined, for all state $x \in X$ and a set of uncontrollable events Σ_{uc}, as $UR(x, \Sigma_{uc}) = \{y \in X : (\exists t \in \Sigma_{uc}^\star)[f_{nd}^e(x, t) = y]\}$. This function is extended to a set of states $B \subseteq X$ as $UR(B, \Sigma_{uc}) = \bigcup_{x \in B} UR(x, \Sigma_{uc})$.

In order to verify the property of NA-Safe controllability, we need first to compute automata $\mathcal{T}_S$ and $\mathcal{T}_U$ whose generated languages are $L(T) \cup \overline{L}_s(\mathcal{T}_a)$ and $L_{us}(\mathcal{T}_a)$, respectively. These automata are constructed according to Algorithm 2.

The verification of NA-Safe controllability can be carried out by using the verifier automaton proposed in MOREIRA *et al.* (2011) computed according to Algorithm 3.

In Algorithm 3, we describe a method for verifying the NA-Safe controllability of $L(\mathcal{T}_a)$. The verification is based on the construction of the verifier automaton $\mathcal{V} = \mathcal{T}_S \| \mathcal{T}_{U,R}$, where the language generated by $\mathcal{T}_S$ is $L(T) \cup \overline{L}_s(\mathcal{T}_a)$, and the language generated by $\mathcal{T}_{U,R}$ is composed of all renamed traces of $L(\mathcal{T}_U)$, *i.e.*,

Algorithm 2: Construction of $\mathcal{T}_S$ and $\mathcal{T}_U$.

Inputs : $T = (X_T, \Sigma, f_T, x_{0,T})$, $\mathcal{T}_a = (X_{\mathcal{T}_a}, \Sigma \cup \{\varepsilon\}, f_{\mathcal{T}_a}, X_{0,\mathcal{T}_a})$, Σ_{ca}, $X_{US}^{\mathcal{T}_a}$.
Outputs: $\mathcal{T}_S = (X_{\mathcal{T}_S}, \Sigma \cup \{\varepsilon\}, f_{\mathcal{T}_S}, X_{0,\mathcal{T}_S})$ $\mathcal{T}_U = (X_{\mathcal{T}_U}, \Sigma \cup \{\varepsilon\}, f_{\mathcal{T}_U}, X_{0,\mathcal{T}_U})$

1 Define $X_{US}^{\mathcal{T}_a}$ as the set of marked states of $\mathcal{T}_a$.
2 Define $\mathcal{T}_U = CoAc(\mathcal{T}_a) = (X_U, \Sigma \cup \{\varepsilon\}, f_U, X_{0,U}, X_{US}^{\mathcal{T}_a})$, and unmark its states.
3 Define $X_{\mathcal{T}_a} \setminus X_U$ as the set of marked states of $\mathcal{T}_a$.
4 Define $\mathcal{T}_S' = CoAc(\mathcal{T}_a)$, and unmark its states.
5 Construct automaton $\mathcal{T}_S$ such that $L(\mathcal{T}_S) = L(\mathcal{T}_S') \cup L(T)$.

Algorithm 3: NA-Safe Controllability Verifier

Inputs : $T = (X_T, \Sigma_T, f_T, x_{0T})$, $\mathcal{T}_a = (X_{\mathcal{T}_a}, \Sigma \cup \{\varepsilon\}, f_{\mathcal{T}_a}, X_{0,\mathcal{T}_a})$, Σ_{ca}, $X_{US}^{\mathcal{T}_a} \subset X_{\mathcal{T}_a}$.
Output: NASafeCont $\in \{TRUE, FALSE\}$

1 Compute $\mathcal{T}_U$ and $\mathcal{T}_S$ by following Algorithm 2.
2 Construct automaton $\mathcal{T}_{U,\rho} = (X_U, \Sigma_\rho, f_\rho, x_{0U})$, where $\Sigma_\rho = \rho(\Sigma \cup \{\varepsilon\})$ and $f_\rho(x, \rho(\sigma)) = f(x, \sigma)$, $\forall \sigma \in \Sigma \cup \{\varepsilon\}$.
3 Compute $\mathcal{V} = (X_V, \Sigma_V, f_V, x_{0V}) = \mathcal{T}_S \| \mathcal{T}_{U,\rho}$.
4 *Compute the unsafe region US_R, and unsafe boundary US_B, as follows:*
5 Define the reverse of $\mathcal{T}_a$ as $\mathcal{T}_a^r = (X_{\mathcal{T}_a}, \Sigma \cup \{\varepsilon\}, f_{\mathcal{T}_a}^r, X_{0,\mathcal{T}_a})$ where $f_{\mathcal{T}_a}^r(x, \sigma) = y \Leftrightarrow f_{\mathcal{T}_a}(y, \sigma) = x$, $\forall x, y \in X_{\mathcal{T}_a}$, and $\forall \sigma \in \Sigma \cup \{\varepsilon\}$.
6 Define the unsafe region from $\mathcal{T}_a^r$ as $US_R = UR(X_{US}^{\mathcal{T}_a}, \Sigma_{uca})$, where $\Sigma_{uca} = \Sigma \setminus \Sigma_{ca}$.
7 Define the unsafe boundary from $\mathcal{T}_a^r$ as $US_B = \{x \in X_{\mathcal{T}_a} \setminus US_R : (\exists \sigma \in \Sigma_{ca}) \wedge (\exists y \in US_R)\ [f_{\mathcal{T}_a}^r(y, \sigma) = x]\}$.
8 **end**
9 If there exists a state $x_V = (x_S, x_U) \in X_V$ such that $x_U \in US_R \cup US_B$ then NASafeCont $= FALSE$, otherwise NASafeCont $= TRUE$.

$L(\mathcal{T}_{U,R}) = R(L(\mathcal{T}_U)) = R(L_{us}(\mathcal{T}_a))$[1]. As shown in MOREIRA *et al.* (2011), all traces of $\mathcal{V}$ are associated with a trace in $L(T) \cup \overline{L}_s(\mathcal{T}_a)$ and a trace in $L_{us}(\mathcal{T}_a)$ that have the same projection. Thus, if a trace of $\mathcal{V}$ reaches a state $x_V = (x_S, x_U)$, where $x_U \in US_R \cup US_B$, then there exist an unsafe trace in $L_{us}(\mathcal{T}_a)$, and a trace in $L(T) \cup \overline{L}_s(\mathcal{T}_a)$ that cannot be distinguished, and since state x_U is in the unsafe boundary or in the unsafe region, it is impossible to prevent reaching an unsafe state by disabling controllable events. In the sequel, we prove the correctness of Algorithm 3.

Theorem 1 *The language generated by $\mathcal{T}_a$, $L(\mathcal{T}_a)$, is NA-Safe controllable with respect to T, $P_o : \Sigma^\star \to \Sigma_o^\star$, Σ_{ca} and $X_{US}^{\mathcal{T}_a}$ if, and only if, $NASafeCont = TRUE$.*

Proof: ($\Rightarrow$) Let us first consider that $NASafeCont = FALSE$. Then, there exists a state $x_V = (x_S, x_U) \in X_V$ such that either $x_U \in US_R$, or $x_U \in US_B$. Since $\mathcal{V} = \mathcal{T}_S \| \mathcal{T}_{U,R}$, then, in accordance with MOREIRA *et al.* (2011), $\exists \omega \in L(\mathcal{T}_S)$ and $\exists s_1 \in L(\mathcal{T}_U)$ such that $P_o(\omega) = P_o(s_1)$. If $x_U \in US_B$, then there exists an event $\sigma \in \Sigma_{ca}$ such that $f_{\mathcal{T}_a}^e(X_{0,\mathcal{T}_a}, s_1\sigma) \cap US_R \neq \emptyset$. According to the construction of US_R, there exists a trace $\bar{s}_2 \in \Sigma_{uca}^\star$ such that $f_{\mathcal{T}_a}^e(X_{0,\mathcal{T}_a}, s_1\sigma\bar{s}_2) \cap X_{US}^{\mathcal{T}_a} \neq \emptyset$. Thus, defining trace $s = s_1\sigma\bar{s}_2$, we conclude that s violates Definition 13 since there exists $\omega \in L(T) \cup \overline{L}_s(\mathcal{T}_a))$ such that $P_o(s_1) = P_o(\omega)$, and $\Sigma_{ca} \notin \bar{s}_2$. Therefore, language $L(\mathcal{T}_a)$ is not NA-Safe controllable. On the other hand, if $x_U \in US_R$, then, $\exists \omega \in L(\mathcal{T}_S)$ and $\exists s_1 \in L(\mathcal{T}_U)$ such that $P_o(\omega) = P_o(s_1)$ and $f_{\mathcal{T}_a}^e(X_{0,\mathcal{T}_a}, s_1) \cap US_R \neq \emptyset$. According to the definition of US_R, there exists a trace $s_2 \in \Sigma_{uca}^\star$, such that $f_{\mathcal{T}_a}^e(X_{0,\mathcal{T}_a}, s_1 s_2) \cap X_{US}^{\mathcal{T}_a} \neq \emptyset$, which violates Definition 13, and language $L(\mathcal{T}_a)$ is not NA-Safe controllable.

($\Leftarrow$) Let $NASafeCont = TRUE$. Then, for all $x_V = (x_S, x_U) \in X_V$, we have that $x_U \notin US_B$ and $x_U \notin US_R$. Since, as shown in MOREIRA *et al.* (2011), the verifier represents only the traces of $L(\mathcal{T}_S)$ and $L(\mathcal{T}_U)$ that have the same projection, we conclude that for all traces $s_1 \in L(\mathcal{T}_U)$ that reaches a state in US_B, $P_o(s_1) \notin P_o(L(\mathcal{T}_S))$. From the construction of US_R and US_B, each state $x \in X_{US}^{\mathcal{T}_a}$ of $\mathcal{T}_a$ is reached from a state in US_B through a trace $s_2 = \sigma\bar{s}_2$, where $\sigma \in \Sigma_{ca}$ and $\bar{s}_2 \in \Sigma_{uca}^\star$. Therefore, for any trace $s \in L(\mathcal{T}_a)$ such that $f_{\mathcal{T}_a}^e(X_{0,\mathcal{T}_a}, s) \cap X_{US}^{\mathcal{T}_a} \neq \emptyset$, there exist traces $s_1, s_2 \in \Sigma^\star$ such that $s = s_1 s_2$, $\Sigma_{ca} \in s_2$, and, $\forall \omega \in L(\mathcal{T}_S)$, $P_o(s_1) \neq P_o(\omega)$. Since, $L(\mathcal{T}_S) = L(T) \cup \overline{L}_s(\mathcal{T}_a)$, we conclude that language $L(\mathcal{T}_a)$ is NA-Safe controllable. ■

Example 13 *Let us consider again the plant automaton of Example 7, depicted in Figure 3.2, where $\Sigma_o = \{a, b, d, g\}$ and $\Sigma_c = \{c, e\}$, and assume that $\Sigma_{vs} = \{g\}$ and $\Sigma_{va} = \{c\}$. Then, automata T and $\mathcal{T}_a$, shown in Figures 3.6 and 3.7, respectively,*

[1]Here we consider the extension of the domain of function ρ to consider traces in $\Sigma^\star$ as $\rho(\varepsilon) = \varepsilon$, and $\rho(s\sigma) = \rho(s)\rho(\sigma)$, for all $s \in \Sigma^\star$, and $\sigma \in \Sigma$.

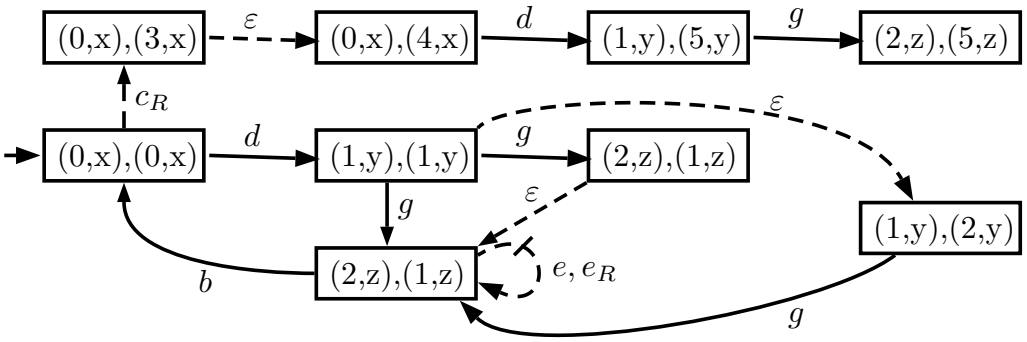

Figure 3.11: Verifier automaton $\mathcal{V}$.

are computed as presented in Example 9. In this case, $\Sigma_{ca} = \Sigma_c \setminus \Sigma_{va} = \{e\}$*, and* $X_{US}^{\mathcal{T}_a} = \{(7, z)\}$.

In the first step of Algorithm 3, automata $\mathcal{T}_S$ *and* $\mathcal{T}_U$ *are computed by applying Algorithm 2. In this case,* $\mathcal{T}_S$ *is equal to* T*, depicted in Figure 3.6, and* $\mathcal{T}_U$ *is equal to* $\mathcal{T}_a$*, depicted in Figure 3.7. Then, in Step 2 of Algorithm 3, automaton* $\mathcal{T}_{U,R}$ *is computed by renaming the events in* Σ_{uo} *of* $\mathcal{T}_U$*, and in Step 3, the verifier automaton* $\mathcal{V}$*, shown in Figure 3.11, is computed. In order to verify the NA-Safe controllability of* $L(\mathcal{T}_a)$*, it is important to find the unsafe region* US_R *and the unsafe boundary* US_B *of* $\mathcal{T}_a$*. By following the procedure presented in Step 4 of Algorithm 3, we obtain* $US_R = \{(7, z)\}$ *and,* $US_B = \{(6, z)\}$*. Since the second component of all states of* $\mathcal{V}$ *are not in the unsafe boundary* US_B*, or in the unsafe region* US_R*, then, we conclude that language* $L(\mathcal{T}_a)$ *is NA-Safe controllable with respect to* T*,* $P_o : \Sigma^\star \to \Sigma_o^\star$*,* Σ_{ca} *and* $X_{US}^{\mathcal{T}_a}$*.* □

3.3.3 Implementation of the Security Module

The security supervisor observes the same trace observed by the supervisor, and then, it verifies if this trace belongs to $P_o(L(T) \cup \overline{L}_s(\mathcal{T}_a))$. If the observed trace does not belong to $P_o(L(T) \cup \overline{L}_s(\mathcal{T}_a))$, then the security supervisor must send an information to the supervisor to disable all controllable events in Σ_{ca}, preventing the reach of unsafe states. Thus, in order to implement the security supervisor, we need first to compute automaton $\mathcal{T}_S$ whose generated language is $L(T) \cup \overline{L}_s(\mathcal{T}_a)$. This automaton is constructed as shown in Algorithm 2. After the computation of automaton $\mathcal{T}_S$, the security supervisor operation can be implemented as follows.

We prove in the sequel that the NA-Safe controllability of $L(\mathcal{T}_a)$ is a necessary and sufficient condition for the existence of the security supervisor.

Theorem 2 *There exists a security supervisor, obtained according to Algorithm 4, that is capable of preventing* G *from reaching an unsafe state in* X_{US} *if, and only if,* $L(\mathcal{T}_a)$ *is NA-Safe controllable with respect to* T*,* $P_o : \Sigma^\star \to \Sigma_o^\star$*,* Σ_{ca} *and* $X_{US}^{\mathcal{T}_a}$*.*

Algorithm 4: Security supervisor operation

1 Compute the initial state estimate of automaton $\mathcal{T}_S$, $x_{0,Obs}$, and define the current state estimate as $x_{c,Obs} = x_{0,Obs}$.
2 Define the set of events Γ_c in the current state estimate $x_{c,Obs}$ that belong to the active events as $\Gamma_c = \bigcup_{x \in x_{c,Obs}} \Gamma(x)$.
3 *Wait for the next event observation e:*
4 **if** $e \notin \Gamma_c$ **then**
5 disable all controllable events of Σ_{ca}.
6 **else**
7 update the current state estimate $x_{c,Obs}$ of automaton $\mathcal{T}_S$, and go back to Step 2.
8 **end**
9 **end**

Proof:

($\Rightarrow$) Assume that $L(\mathcal{T}_a)$ is not NA-Safe controllable. Then, according to Definition 13, there exits $s \in L(\mathcal{T}_a)$ such that $f^e_{\mathcal{T}_a}(X_{0,\mathcal{T}_a}, s) \cap X^{\mathcal{T}_a}_{US} \neq \emptyset$, and, for all s_1 and s_2 such that $s = s_1 s_2$, at least one of the two conditions is true: *(i)* there exists $w \in L(T) \cup \overline{L}_s(\mathcal{T}_a)$ such that $P_o(s_1) = P_o(w)$; *(ii)* $s_2 \in \Sigma^\star_{uca}$. If condition *(i)* is true, then it is impossible to obtain a security supervisor that can force the supervisor to disable the controllable events after observing $P_o(s_1)$, since $P_o(s_1)$ cannot be distinguished from the observation of a trace w generated by the closed-loop system without the occurrence of attacks ($w \in L(T)$), or from a trace w that belongs to the prefix closure of the safe language ($w \in \overline{L}_s(\mathcal{T}_a)$). If condition *(ii)* is true, even if the security supervisor forces the supervisor to disable all controllable events after observing $P_o(s_1)$, the system will reach an unsafe state, since s_2 is composed only of uncontrollable and/or vulnerable controllable events.

($\Leftarrow$) According to Steps 2 and 3 of Algorithm 4, the security supervisor verifies if the trace observed by the supervisor belongs to $P_o(L(\mathcal{T}_S))$ after each new event observation, and, when the observed trace does not belong to $P_o(L(\mathcal{T}_S))$, the security supervisor forces the supervisor to disable all controllable events. According to Definition 13, if $L(\mathcal{T}_a)$ is NA-Safe controllable, then all traces $s \in L(\mathcal{T}_a)$ such that $f^e_{\mathcal{T}_a}(X_{0,\mathcal{T}_a}, s) \cap X^{\mathcal{T}_a}_{US} \neq \emptyset$ can be partitioned as $s = s_1 s_2$ where, for all $w \in L(T) \cup \overline{L}_s(\mathcal{T}_a)$, $P_o(w) \neq P_o(s_1)$ and $\Sigma_{ca} \in s_2$. Therefore, the security supervisor will be capable of forcing the supervisor to disable all controllable events after the observation of $P_o(s_1)$, since $P_o(s_1) \notin P_o(L(T) \cup \overline{L}_s(\mathcal{T}_a)) = P_o(L(\mathcal{T}_S))$. As a consequence, the system will not reach an unsafe state in X_{US}, since s_2 has at least one controllable event that can be disabled by the supervisor, and cannot be enabled by the intruder. ■

According to Theorem 2, if $L(\mathcal{T}_a)$ is NA-Safe Controllable, then the security

supervisor implemented according to Algorithm 4 is capable of preventing damages caused by cyber-attacks in the system. The idea of implementing a device that prevents the system from reaching unsafe states is also presented in PAOLI and LAFORTUNE (2005) in a different context. In PAOLI and LAFORTUNE (2005), the authors address the problem of safe diagnosability, where a diagnoser is used to prevent the system from reaching unsafe states after the occurrence of a fault event. Differently from the security supervisor proposed in this work, the safe diagnoser proposed in PAOLI and LAFORTUNE (2005) is constructed based on diagnosers (SAMPATH *et al.*, 1995). Consequently, in the worst-case, it grows exponentially with the size of the plant. In the sequel, we present a simple example that illustrates the systematical computation and implementation of the security strategy proposed.

Example 14 *In order to illustrate the implementation of the security supervisor following the method described in Algorithm 4, let us consider the railway system shown in Figure 3.12, which consists of two tracks connected by a secondary line. In Figure 3.12, sensors are represented by arrows, and identify the passing of a train through a position in the track. The switches are represented by red lines in the tracks, and change the direction of trains. Depending on the position of switch i, Train 2 can move in the direction of Track 1, and depending on the position of switch g, it can continue moving in the direction of Track 1 or return to Track 2. Train 1 cannot move in the direction of Track 2.*

The desired behavior of the railway system is that Train 1 moves only in Track 1, and Train 2 moves only in Track 2. Thus, the supervisor objective is to avoid the crashing of the trains, i.e., the objective is to keep Train 2 in Track 2.

In this example, the set of observable events is given by $\Sigma_o = \{a, b, d, e\}$, that are associated with the observation of the position of the trains provided by sensors placed in the tracks. The set of controllable events is given by $\Sigma_c = \{i, g, c\}$, where i and g are associated with switches such that when i is enabled, Train 2 is deviated to the secondary line, and when g is enabled, Train 2 moves in the direction of Track 1. Event c represents the entrance of Train 2 in Track 1. When event c is disabled, Train 2 and Train 1, if it is close to the secondary line, are stopped to avoid their collision. Thus, disabling event c is a last resort to prevent Train 2 to enter in Track 1. Note that event c must be enabled during the non attacked operation of the system.

With a view to designing the security supervisor for the system of Figure 3.12, it is first necessary to obtain the automaton model for the plant depicted in Figure 3.13. Each state of G represents a position of Train 2 on the railway as shown in Table 3.1, where state 8 represents that Train 2 has reached Track 1 and, therefore, the set of unsafe states is given by $X_{US} = \{8\}$.

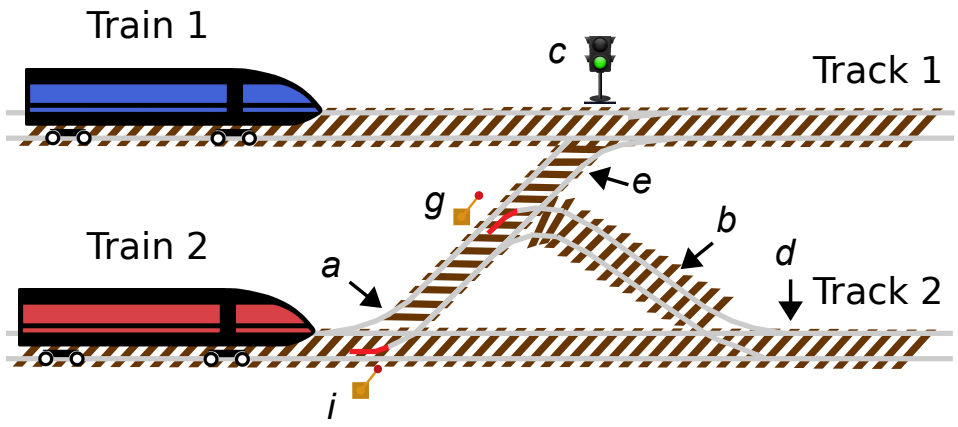

Figure 3.12: Railway example.

Table 3.1: Meaning of the states of G.

State	Meaning
1	Train 2 in Track 2 before position d
2	Train 2 in the secondary line before position a
3	Train 2 between position a and switch g
4	Train 2 in the secondary line after position b
5	Train 2 in Track 2 after position d
6	Train 2 between switch g and position e
7	Train 2 between position e and Track 1
8	Train 2 in Track 1

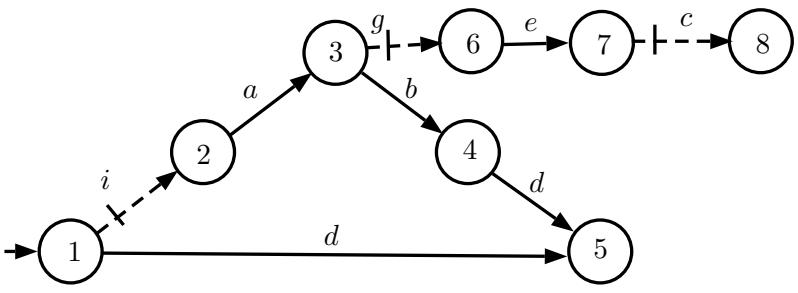

Figure 3.13: Plant model G for the railway example.

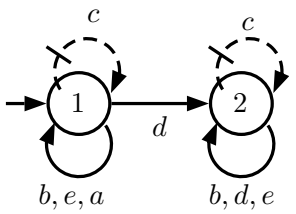

Figure 3.14: Supervisor H.

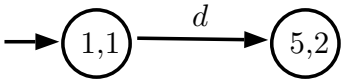

Figure 3.15: Closed-loop system $T = G \| H$ for the railway example.

A supervisor H that avoids that Train 2 reaches Track 1, is shown in Figure 3.14. Note that, event c is enabled in all states of H, since the disablement of c may cause the sudden stop of Train 1. The closed-loop system $T = G \parallel H$ is depicted in Figure 3.15. Note that, as expected, the unsafe state of the plant (state 8) is not reachable in the non attacked closed-loop system behavior.

Let us consider now that the set of vulnerable observable events is $\Sigma_{vs} = \{a\}$. Thus, according to Definition 8, the attacked plant model $\mathcal{G}_a$ is represented in Figure 3.16. Note that $\mathcal{G}_a$ is obtained from G by adding self-loops labeled with a to each state of G to represent the capability of the malicious agent of creating observations of the vulnerable event a. Moreover, transitions labeled with ε are also added in parallel to the transitions labeled with a of G to represent the capability of the intruder of erasing the observation of event a. The capability of changing the observation of an event has the same effect as deleting its observation, and then creating the observation of a new event. Thus, this type of modification is already represented by the self-loops and ε-transitions of $\mathcal{G}_a$.

Let $\Sigma_{va} = \{i, g\}$ be the set of vulnerable actuator events. Then, the supervisor under attack H_a is represented in Figure 3.17. For the construction of the attacked supervisor we use Definition 9, where the supervisory channel attack is modeled by adding self-loops labeled with the vulnerable actuator events in the states of H,

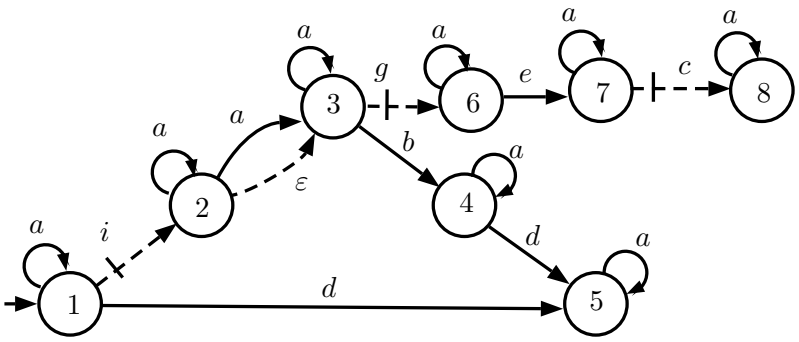

Figure 3.16: Plant model under attack $\mathcal{G}_a$.

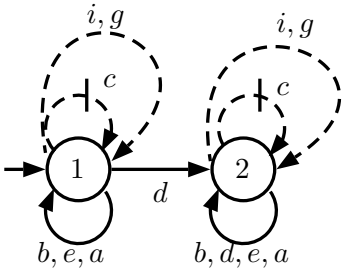

Figure 3.17: Supervisor model under attack H_a.

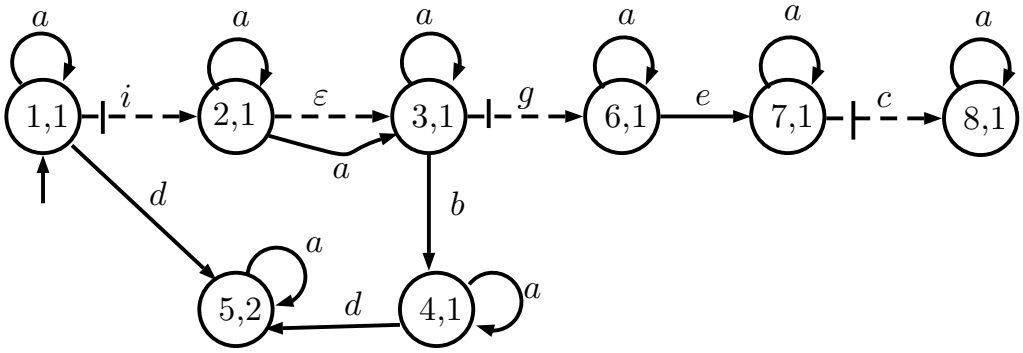

Figure 3.18: Closed-loop system model under attack $\mathcal{T}_a$ for the railway example.

to represent that these events become uncontrollable after the attack, and self-loops labeled with uncontrollable events such that the supervisor remains admissible.

The attacked closed-loop system $\mathcal{T}_a = \mathcal{G}_a \| H_a$ *is represented in Figure 3.18. Note that state* $(8,1)$ *of* $\mathcal{T}_a$ *represents that Train 2 has reached Track 1, i.e., the attack can make the plant reach the unsafe state* 8*. In this example, following the steps of Algorithm 3, it can be verified that* $L(\mathcal{T}_a)$ *is NA-Safe controllable. Thus, according to Theorem 2, it is possible to implement a security supervisor that prevents the plant from reaching unsafe state* 8*.*

In order to implement the security supervisor, it is first necessary to compute automaton $\mathcal{T}_S$ *following the steps of Algorithm 2. In Steps 1 and 2 of Algorithm 2, we compute automaton* $\mathcal{T}_U$*, depicted in Figure 3.19, by removing from* $\mathcal{T}_a$ *the states from which it is not possible to reach the unsafe state* $(8,1)$*, i.e., in this example, states* $(5,2)$ *and* $(4,1)$*. The computation of automaton* $\mathcal{T}_S'$*, depicted in Figure 3.20, is carried out by following Steps 3 and 4 of Algorithm 2. In this example, states* $(6,1)$*,* $(7,1)$*, and* $(8,1)$ *are removed from* $\mathcal{T}_a$ *to obtain automaton* $\mathcal{T}_S'$*. Now, following Step 5 of Algorithm 2,* $\mathcal{T}_S$ *is constructed. In this example,* $\mathcal{T}_S = \mathcal{T}_S'$*. Once automaton* $\mathcal{T}_S$ *has been obtained, the security supervisor is implemented by using Algorithm 4, which performs the online state estimate of* $\mathcal{T}_S$ *after the observation of events executed by the plant. If an observed event is not feasible in the current state estimate, then the attack is detected and all non vulnerable controllable events are disabled by the security supervisor.*

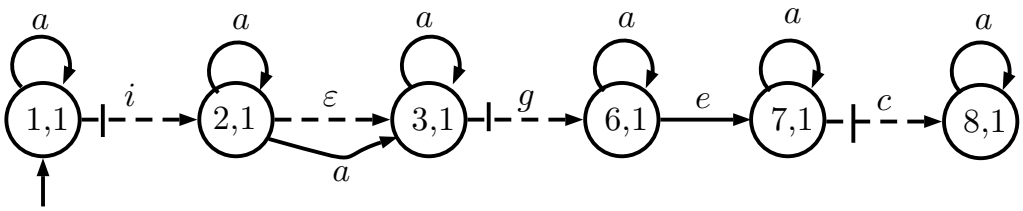

Figure 3.19: Unsafe automaton model $\mathcal{T}_U$.

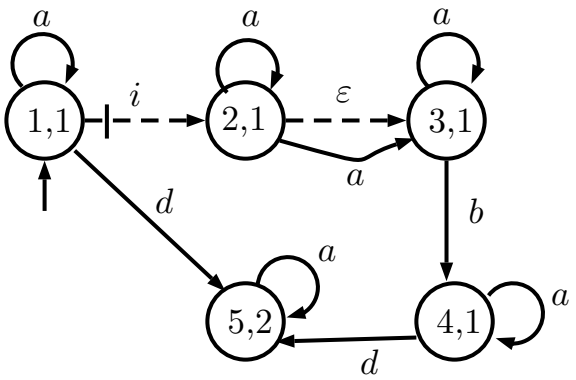

Figure 3.20: Safe automaton model $\mathcal{T}_S = \mathcal{T}'_S$.

The behavior of the security model can be analyzed from the observer automaton of $\mathcal{T}_S$ depicted in Figure 3.21. Consider that system G is in its initial state 1*, and the attack has not occurred yet. Thus, the current state estimate of $\mathcal{T}_S$ is $\{(1,1),(2,1),(3,1)\}$. Then, assume that the intruder enables event i, which makes Train 2 deviates to the secondary line, and erases the observation of event a. As a consequence, the security supervisor is not capable of detecting the intrusion since no change occurs in the state estimate. However, notice that, in state $\{(1,1),(2,1),(3,1)\}$ of the observer of $\mathcal{T}_S$, the feasible events are a, b, and d, and, consequently, if the intruder enables event g (which is unobservable) after trace ia has been executed by the plant, then event e is observed by the security supervisor after Train 2 has crossed switch g. Since e is not feasible in the current state estimate, event c is disabled by the security supervisor, and Train 2 is stopped without reaching Track 1.*

Note that if the attacker does not enable event g after enabling event i and deleting the observation of event a, then Train 2 goes back to Track 2, and event b is observed. In this case, the security supervisor does not force the disablement of the non vulnerable controllable events in Σ_{ca}, since the attack has not been efficient, i.e., the plant has reached, after the attack, a state from which it is not possible to reach its unsafe state anymore. This shows an advantage of the method proposed in this work with respect to the method presented in CARVALHO et al. *(2018). In CARVALHO* et al. *(2018), all controllable events are disabled when the attack is detected, even if the attack cannot cause damages to the system.*

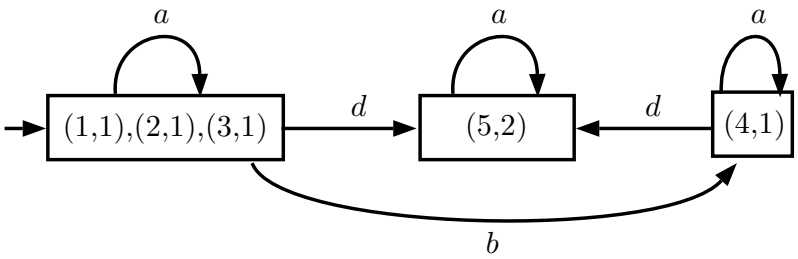

Figure 3.21: Observer of $\mathcal{T}_S$.

Although a simple defense strategy can be obtained by inspection of this small example, that is, to disable event c after the observation of event e, it is useful to illustrate how to systematically compute and implement the security supervisor for complex and large systems for which a security supervisor cannot be straightforwardly obtained. Another contribution of the work is the verification of which communication channels must have a higher level of hardware protection against cyber-attacks to prevent the plant from reaching unsafe states. In this example, if disabling event c is not desired, then it can be verified by using the method proposed in this work that the supervisory control communication channel that transmits event g must have a higher level of protection against attacks.

3.4 Maximally permissive approach

Although the approach proposed in LIMA *et al.* (2019) is more permissive than the one proposed in CARVALHO *et al.* (2018), it can still be conservative, as illustrated by the following example.

Example 15 *Consider the system presented in Example 11 of Section 3.3, and the attacked closed-loop system $\mathcal{T}_a$ depicted in Figure 3.8. Let us consider sequence $s_3 = gdc \in L(\mathcal{T}_a)$. As shown in Example 11, sequence s_3 belongs to L_{NIR}, and thus, there is no imminent risk of reaching an unsafe state in $X_{US}^{\mathcal{T}_a}$. However, it can be checked that $P_o(s_3) = dc$ is neither in $P_o(L(T))$ nor in $P_o(\overline{L_s(\mathcal{T}_a)}))$, and, consequently, the security supervisor proposed in LIMA* et al. *(2019) disables all non-vulnerable controllable events after the occurrence of s_3, and so, it restricts the system operation by preventing subsequent occurrences of event e. Notice that, in this example, the sequences $gdce^n$ for any $n \in \mathbb{N}$ lead the attacked closed-loop system to state $(2, 3)$, i.e., the plant reaches state 2, which is not an unsafe state of the plant. However, the security supervisor proposed in LIMA* et al. *(2019) unnecessarily prevent the occurrences of these sequences.* □

We then propose a new security supervisor that acts solely when it observes a sequence in $P_o(L_{IR})$ of Definition 12. In addition, in order to be more permissive than

the methods proposed in CARVALHO *et al.* (2018) and LIMA *et al.* (2019), instead of disabling all non-vulnerable controllable events, the security model proposed in this chapter disables only those controllable events that may lead the system to the unsafe region US_R.

We may now formulate the following security supervisory control problem.

Problem 1 (Maximally-Permissive Security Problem (MPSP)) *Given an ANS* $\Upsilon = (G, H, \Sigma_{vs}, \Sigma_{va})$*, whose non-attacked closed-loop and attacked closed-loop system models are represented by* T *and* $\mathcal{T}_a$*, respectively, and a set of unsafe states* $X_{US}^{\mathcal{T}_a}$*, synthesize a security supervisor* Ψ *that satisfies the following requirements:*
R1. $L(T) \subseteq L(\Psi/\mathcal{T}_a)$*;*
R2. *For all* $s \in L(\Psi/\mathcal{T}_a)$*,* $f_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s) \cap X_{US}^{\mathcal{T}_a} = \emptyset$*;*
R3. *There is no security supervisor* Ψ' *that satisfies Requirements* ***R1*** *and* ***R2*** *such that* $L(\Psi/\mathcal{T}_a) \subset L(\Psi'/\mathcal{T}_a)$*.* □

Requirement **R1** ensures that the language generated by the closed-loop system is equal to $L(T)$ when no attack occurs (Objective **O1**). This is so because security supervisor Ψ must allow the occurrence of every sequence in $L(T)$, and supervisor S ensures that the ANS generates $L(T)$ when no attack occurs. Requirement **R2** ensures that unsafe states are never reached (Objective **O2**). According to Requirement **R3**, the security supervisor Ψ must be maximally-permissive in order to allow that sequences generated after network attacks, that do not belong to the language generated by the closed-loop system model T, be executed, if these sequences do not drive the closed-loop system to unsafe states. Thus, if the attack is not efficient, in the sense that it is not capable of damaging the system, the security supervisor will not stop the system operation.

It is important to remark that Problem 1 can be formulated as a maximally permissive range control problem (MPRCP) LIN and WONHAM (1988); YIN and LAFORTUNE (2017). The MPRCP consists in computing a supervisor S for a plant G that satisfies three conditions: (i) the behavior of the system under supervisory control does not exceed the legal behavior defined by an upper bound language K, *i.e.*, $L(S/G) \subseteq K$; (ii) the required behavior, described by a lower bound language R, is contained in the behavior of the controlled system, *i.e.*, $R \subseteq L(S/G)$; and (iii) there is no other supervisor S' that satisfies (i) and (ii), and $L(S/G) \subset L(S'/G)$, *i.e.*, S is maximally permissive. Thus, Problem 1 can be reduced to MPRCP by assuming $\mathcal{T}_a$ as the plant and defining $R = L(T)$ and $K = L(\tilde{\mathcal{T}}_a)$, where $\tilde{\mathcal{T}}_a = (X_{\mathcal{T}_a} \setminus X_{US}^{\mathcal{T}_a}, \Sigma \cup \{\varepsilon\}, f_{\tilde{\mathcal{T}}_a}, x_{0,\mathcal{T}_a})$, with $f_{\tilde{\mathcal{T}}_a}(x, \sigma) = f_{\mathcal{T}_a}(x, \sigma) \setminus X_{US}^{\mathcal{T}_a}$, for all $x \in X_{\mathcal{T}_a} \setminus X_{US}^{\mathcal{T}_a}$ and $\sigma \in \Sigma \cup \{\varepsilon\}$ if $f_{\mathcal{T}_a}(x, \sigma) \setminus X_{US}^{\mathcal{T}_a} \neq \emptyset$, and undefined, otherwise. Notice that $\tilde{\mathcal{T}}_a$ models the attacked closed-loop system without the unsafe states. In YIN and LAFORTUNE (2017), a solution to MPRCP is computed using a structure

called All Inclusive Controller (AIC) YIN and LAFORTUNE (2016). The AIC is a game structure that includes all supervisors S such that $L(S/G) \subseteq K$ for a given specification K. The construction of the AIC has computational complexity $O(|X||\Sigma|2^{|X|+|\Sigma|})$, where $|X|$ and $|\Sigma|$ are the cardinalities of the sets of states and events of the plant, respectively. Thus, the solution to MPRCP proposed in YIN and LAFORTUNE (2017) is exponential with respect to the number of states and events of the system. In addition, the methods presented in the literature for the verification of the existence of a solution to the MPRCP have also exponential complexity with the number of system states MASOPUST (2018); YIN and LAFORTUNE (2017).

With a view to circumventing the complexity problem, in Section 3.4, we introduce a class of systems, called NA-Secure Systems, for which there exists a polynomial-time solution to MPSP (Problem 1). In addition, determining if a system belongs to the class of NA-Secure systems can also be carried out in polynomial-time.

In LIMA *et al.* (2019) as presented in Section 3.3, a method to thwart man-in-the-middle attacks in both sensor and actuator channels is presented, where all controllable events are disabled only if the system may reach an unsafe state. This defense strategy does not restrict the system behavior if the attack cannot cause damages to the system.

In LIMA *et al.* (2018), two notions of network attack security for DESs, called Detectable Network Attack Security and Undetectable Network Attack Security, associated with the capability of disabling some controllable events to prevent the system from reaching unsafe states, are presented. Based on these notions, a security supervisor to prevent the reach of unsafe states, without altering the non-attacked language of the system, is proposed.

In this section, we propose a new defense strategy that thwarts man-in-the-middle attacks in the sensor and/or control communication channels in supervisory control systems. We propose the design of a security supervisor that disables controllable events only if there is a real risk of damaging the system, without interrupting unnecessarily the system operation. Moreover, the security supervisor proposed in this part of this work operates together with the existing supervisor, instead of substituting it. By doing so, the combined supervisory control policy can be made more permissive, without altering the non-attacked closed-loop behavior. The main difference between the security supervisor proposed in Section 3.3 and the security supervisor proposed in this section, is that in Section 3.3 all controllable events are disabled when the attack is detected and may lead the system to unsafe states, whereas, in this section, the proposed security supervisor disables only those controllable events that lead the system to an imminent risk scenario. In addition, the proposed security supervisor, in some cases, can prevent the system from reaching unsafe states even without the attack detection, *i.e.*, the method proposed in this

work can also be used to protect the system against stealthy attacks GAO *et al.* (2019); GOES *et al.* (2017). This strategy is less restrictive than the one proposed in Section 3.3, in the sense that the system behavior may execute more sequences after an attack, without reaching unsafe states. Thus, the system may execute safe sequences while the threat is eliminated, or eventually the system returns to its non-attacked closed-loop behavior after the actuation of the security supervisor. We also introduce in this work a class of systems, called NA-Secure Systems, for which a security supervisor, that thwarts attacks in the network without altering the non-attacked closed-loop behavior, can be computed in polynomial time. A polynomial time algorithm to verify the NA-Security property is proposed.

It is important to remark that the security supervisor proposed in this section is an improvement of the security supervisor presented in LIMA *et al.* (2018), since in the work presented in this section we propose a security supervisor that does not interfere with the non-attacked closed-loop system, and is also maximally permissive. part of this work was published in LIMA *et al.* (2021).

In summary, the main contributions of the work presented in this section are: (*i*) the proposal of a new defense strategy against cyber-attacks with polynomial computational complexity; (*ii*) the definition of NA-Security; (*iii*) the proposal of a method for the verification of this property; and (*iv*) the discussion of the computational complexity of the methods proposed in this work.

3.4.1 NA-Secure Systems

In this section, we introduce a language property, called network attack security (NA-Security), that is used to define a class of ANS called NA-Secure Systems. Such a property is associated with the capability of a security supervisor of preventing the ANS from reaching unsafe states (Requirement **R2** of Problem 1), without altering the non-attacked closed-loop behavior (Requirement **R1** of Problem 1). We also present a polynomial-time algorithm for the verification of NA-Security.

NA-Security

In order to define NA-Security, firstly we introduce the concepts of unsafe region and unsafe boundary for an ANS, considering the associated attacked closed-loop system model $\mathcal{T}_a$, as follows.

Definition 14 (NA-Security) *Let T and $\mathcal{T}_a$ be the non-attacked closed-loop system and the attacked closed-loop system models of an ANS, respectively. Then, $L(\mathcal{T}_a)$ is said to be NA-Secure with respect to $L(T)$, $P_o : \Sigma^\star \to \Sigma_o^\star$, and the set of unsafe states $X_{US}^{\mathcal{T}_a}$ if*

$(\forall s \in L(\mathcal{T}_a))[f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s) \cap US_R \neq \emptyset] \Rightarrow C_{NA}$

where C_{NA} is given as:

$(\exists t_1\sigma \in \overline{\{s\}}, t_1 \in \Sigma^\star, \sigma \in \Sigma_{ca}, f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, t_1) \cap US_B \neq \emptyset, f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, t_1\sigma) \cap US_R \neq \emptyset)[(\forall \omega \in L(T), P_o(t_1) = P_o(\omega))(\omega\sigma \notin L(T))]$. □

According to Definition 14, $L(\mathcal{T}_a)$ is NA-Secure if all sequences $s \in L(\mathcal{T}_a)$ that leads the system to a state in the unsafe region US_R, have a prefix $t_1\sigma$ such that: (i) t_1 leads the system to a state in US_B; (ii) $t_1\sigma$ leads the system to a state in US_R; (iii) t_1 is distinguishable from any sequence ω in $L(T)$ such that $\omega\sigma \in L(T)$, and; (iv) σ is a non-vulnerable controllable event. In the following theorem we show that NA-Security is a sufficient condition for the existence of a security supervisor that solves Problem 1.

Theorem 3 *Let T and $\mathcal{T}_a$ be the non-attacked closed-loop system and the attacked closed-loop system models of an ANS, respectively. If $L(\mathcal{T}_a)$ is NA-Secure with respect to $L(T)$, P_o, and $X^{\mathcal{T}_a}_{US}$, then there exists a security supervisor Ψ that is a solution to MPSP (Problem 1).*

Proof: Suppose that $L(\mathcal{T}_a)$ is NA-Secure, and consider supervisor $\Psi_{min} : P_o(L(\mathcal{T}_a)) \rightarrow 2^{\Sigma_{ca}}$ defined as $\Psi_{min}(t) = \{\sigma \in \Sigma_{ca} : P_o^{-1}(t)\{\sigma\} \cap L(T) = \emptyset\}$. Notice that, Ψ_{min} disables any controllable event σ after observing $t \in P_o(L(\mathcal{T}_a))$ such that there is no sequence $\omega\sigma$ in $L(T)$ with $P_o(\omega) = t$. Thus, $L(T) \subseteq L(\Psi_{min}/\mathcal{T}_a)$ and, consequently, Ψ_{min} satisfies Requirement **R1** of Problem 1. Since $L(\mathcal{T}_a)$ is NA-Secure, then, according to Definition 14, every sequence s that leads the system to an unsafe state has a prefix $t_1\sigma \in \overline{\{s\}}$, where $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, t_1) \cap US_B \neq \emptyset$, $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, t_1\sigma) \cap US_R \neq \emptyset$, $\sigma \in \Sigma_{ca}$, and, for every sequence $\omega \in L(T) \cap P_o^{-1}(P_o(t_1))$, $\omega\sigma \notin L(T)$. Then, according to the definition of Ψ_{min}, after the observation of $P_o(t_1)$, Ψ_{min} disables σ preventing s from occurring, which implies that Ψ_{min} satisfies Requirement **R2** of Problem 1. Finally, Requirement **R3** of Problem 1 is trivially satisfied, since, if there is a security supervisor that satisfies **R1** and **R2**, then there is a maximally-permissive security supervisor, not necessarily equal to Ψ_{min}, that also satisfies **R1** and **R2**. ■

We can now define the class of NA-Secure systems.

Definition 15 *An attackable networked system $\Upsilon = (G, H, \Sigma_{vs}, \Sigma_{va})$ is said to be NA-Secure if its associated attacked closed-loop language $L(\mathcal{T}_a)$ is NA-Secure with respect to $L(T)$, P_o, and $X^{\mathcal{T}_a}_{US}$.* □

Theorem 3 shows that if Υ belongs to the class of NA-Secure systems or, equivalently, if its associated attacked closed-loop language $L(\mathcal{T}_a)$ is NA-Secure, then there exists a maximally permissive security supervisor for Υ. In the sequel, we present a method for the verification of NA-Security.

Verification of NA-Security

In accordance with Definition 14, a language $L(\mathcal{T}_a)$ is NA-secure if any sequence in $L(\mathcal{T}_a)$ that leads the system to a state in the unsafe region US_R satisfies condition C_{NA}. Since, for the verification of NA-Security, it is sufficient to know if the sequence generated by the system leads it to a state in the unsafe region US_R, then all states in US_R can be merged forming a new state us_r. Thus, we define automaton $\mathcal{T}'_a = Ac(X_{\mathcal{T}_a} \cup \{us_r\}, \Sigma \cup \{\varepsilon\}, f_{\mathcal{T}'_a}, x_{0,\mathcal{T}'_a})$, where: *(i)* the new state us_r represents the unsafe region; *(ii)* for all $x \in X_{\mathcal{T}_a}$ and $\sigma \in \Sigma \cup \{\varepsilon\}$, $f_{\mathcal{T}'_a}(x,\sigma) = (f_{\mathcal{T}_a}(x,\sigma) \setminus US_R) \cup \{us_r\}$, if $f_{\mathcal{T}_a}(x,\sigma) \cap US_R \neq \emptyset$, and $f_{\mathcal{T}'_a}(x,\sigma) = f_{\mathcal{T}_a}(x,\sigma)$, otherwise, and $f_{\mathcal{T}'_a}(us_r,\sigma)$ is undefined for all $\sigma \in \Sigma$, and; *(iii)* $x_{0,\mathcal{T}'_a} = x_{0,\mathcal{T}_a}$, if $x_{0,\mathcal{T}_a} \cap US_R = \emptyset$, and $x_{0,\mathcal{T}'_a} = \{us_r\}$, otherwise. Notice that, in the construction of $\mathcal{T}'_a$, some states of the unsafe boundary US_B can be removed by the accessible operation, and thus, the unsafe boundary of $\mathcal{T}'_a$ is defined as $US'_B = US_B \cap X_{\mathcal{T}'_a}$, where $X_{\mathcal{T}'_a}$ is the set of states of $\mathcal{T}'_a$.

In the sequel we present Algorithm 5 for the construction of the verifier automaton to be used in the verification of NA-Security (Algorithm 6), based on the verifier proposed in MOREIRA *et al.* (2011). In order to do so, we need first to complete the language generated by the non-attacked closed-loop automaton T, which leads to automaton $T' = (X_{T'}, \Sigma, f_{T'}, x_{0,T'})$, where $X_{T'} = X_T \cup \{x_d\}$, $x_{0,T'} = x_{0,T}$, $f_{T'}(x,\sigma) = f_T(x,\sigma)$, if $f_T(x,\sigma)$ is defined, and $f_{T'}(x,\sigma) = x_d$, otherwise, for all $x \in X_T$ and $\sigma \in \Sigma$, and $f_{T'}(x_d,\sigma) = x_d$, for all $\sigma \in \Sigma$. Notice that $L(T') = \Sigma^\star$, and that the sequences that reach x_d are those sequences that do not belong to the non-attacked behavior $L(T)$. In addition, it is necessary to define the renaming function ρ as follows: $\rho(\sigma) = \sigma$, if $\sigma \in \Sigma_o$, and $\rho(\sigma) = \sigma_\rho$, if $\sigma \in \Sigma_{uo}$.

Algorithm 5: Verifier Automaton

input : T', $\mathcal{T}'_a$.
output: $\mathcal{V} = (X_\mathcal{V}, \Sigma_\mathcal{V}, f_\mathcal{V}, x_{0,\mathcal{V}})$

1 Construct the renamed automaton $T'_\rho = (X_{T'}, \Sigma_\rho, f_\rho, x_{0,T'})$, where $\Sigma_\rho = \{\rho(\sigma) : \sigma \in \Sigma\}$ and $f_\rho(x,\rho(\sigma)) = f_{T'}(x,\sigma), \forall x \in X_{T'}$ and $\forall \sigma \in \Sigma$.
2 Compute $\mathcal{V} = T'_\rho \| \mathcal{T}'_a$.

Notice that the renaming of the unobservable events of T', generating automaton T'_ρ, transforms these unobservable events into private events of T'_ρ. Thus, only the observable events are synchronized in the computation of $\mathcal{V}$. As a consequence, only the sequences of T'_ρ and $\mathcal{T}'_a$ that have the same observation are mapped into $\mathcal{V}$ CARVALHO *et al.* (2017).

After the construction of verifier automaton $\mathcal{V}$ in accordance with Algorithm 5, we can verify if the corresponding ANS is NA-Secure by using Algorithm 6. In order to facilitate the readability of Algorithm 6, we denote the first (resp. second)

component of a state $x_v \in X_{\mathcal{V}}$ by x_v^1 (resp. x_v^2), *i.e.*, $x_v = (x_v^1, x_v^2)$, where $x_v^1 \in X_{T'}$ and $x_v^2 \in \mathcal{T}_a'$. In addition, we define sets $US_B^{\mathcal{V}} = \{x_v \in X_{\mathcal{V}} : x_v^2 \in US_B'\}$ and $US_R^{\mathcal{V}} = \{x_v \in X_{\mathcal{V}} : x_v^2 = us_r\}$ to denote the sets of states of $\mathcal{V}$ whose second component is in US_B', and is equal to us_r, respectively. We also define set $X_d^{\mathcal{V}} = \{x_v \in X_{\mathcal{V}} : x_v^1 = x_d\}$ formed of the states of $\mathcal{V}$ whose first component is equal to x_d.

Algorithm 6: NA-Security Verification

input : $\mathcal{V} = (X_{\mathcal{V}}, \Sigma_{\mathcal{V}}, f_{\mathcal{V}}, x_{0,\mathcal{V}})$ and Σ_{ca}.
output: $NA_{Sec} \in \{True, False\}$

```
1  if x^2_{0,V} ≠ us_r then
2  |  NA_Sec = True
3  |  US^V_B = {x_v ∈ X_V : x^2_v ∈ US'_B}
4  |  US^V_R = {x_v ∈ X_V : x^2_v = us_r}
5  |  X^V_d = {x_v ∈ X_V : x^1_v = x_d}
6  |  for x_v ∈ US^V_B do
7  |  |  for σ ∈ Σ_ca ∩ Γ_V(x_v) do
8  |  |  |  if f_V(x_v, σ) ∩ US^V_R ≠ ∅ and f_V(x_v, ρ(σ)) ∩ X^V_d = ∅ then
9  |  |  |  |  NA_Sec = False
10 |  |  |  |  Stop the algorithm.
11 |  |  |  end
12 |  |  end
13 |  end
14 else
15 |  NA_Sec = False
16 end
```

In line 1 of Algorithm 6, if the second component of the initial state of $\mathcal{V}$, $x_{0,\mathcal{V}}^2$ is equal to state us_r, then NA_{Sec} is set as $False$. On the other hand, in the case that $x_{0,\mathcal{V}}^2$ is different from state us_r, we firstly set $NA_{sec} = True$ (line 2), and construct sets $US_B^{\mathcal{V}}$ (line 3), $US_R^{\mathcal{V}}$ (line 4), and $X_d^{\mathcal{V}}$ (line 5). In the sequel, we verify if there is a state in $US_B^{\mathcal{V}}$ such that there exists a feasible non-vulnerable controllable event σ that leads to a state in $US_R^{\mathcal{V}}$, for which event $\rho(\sigma)$ does not lead to a state in $X_d^{\mathcal{V}}$. In the case that such a state in $US_B^{\mathcal{V}}$ exists, we redefine NA_{sec} as $False$ and stop the algorithm, otherwise, NA_{sec} remains $True$.

Theorem 4 *A language $L(\mathcal{T}_a)$ is NA-Secure with respect to $L(T)$, P_o, and $X_{US}^{\mathcal{T}_a}$ if, and only if, Algorithm 6 returns $NA_{Sec} = True$.*

Proof: Algorithm 6 returns $NA_{Sec} = True$ if, and only if, the following two conditions are met:

C1. $x_{0,\mathcal{T}_a'} \neq \{us_r\}$;

C2. There do not exist $x_v \in US_B^{\mathcal{V}}$ and $\sigma \in \Sigma_{ca} \cap \Gamma_{\mathcal{V}}(x_v)$ such that $f_{\mathcal{V}}(x_v, \sigma) \cap US_R^{\mathcal{V}} \neq \emptyset$ and $f_{\mathcal{V}}(x_v, \rho(\sigma)) \cap X_d^{\mathcal{V}} = \emptyset$;

Notice that if C1 is false, then $x^2_{0,\mathcal{V}} = us_r$ and, thus, $NA_{Sec} = False$ in line 14. In addition, if C2 is false, then $NA_{Sec} = False$ in line 9. Thus, we will show that $L(\mathcal{T}_a)$ is NA-Secure if, and only if, Conditions C1 and C2 hold true.

($\Rightarrow$) Suppose that Condition C1 is false, *i.e.*, $x_{0,\mathcal{T}'_a} = \{us_r\}$. Then, according to the construction of $\mathcal{T}'_a$, $x_{0,\mathcal{T}_a} \in US_R$, which implies that $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, \varepsilon) \cap US_R \neq \emptyset$. Thus, condition C_{NA} of Definition 14 is not satisfied for $s = \varepsilon \in L(\mathcal{T}_a)$.

Suppose now that Condition C2 is false. Then, there exist a state $x_v = (x^1_v, x^2_v) \in US^{\mathcal{V}}_B$ and an event $\sigma \in \Sigma_{ca} \cap \Gamma_{\mathcal{V}}(x_v)$ such that $f_{\mathcal{V}}(x_v, \sigma) \cap US^{\mathcal{V}}_R \neq \emptyset$ and $f_{\mathcal{V}}(x_v, \rho(\sigma)) \cap X^{\mathcal{V}}_d = \emptyset$. According to the construction of $\mathcal{V}$ (CARVALHO *et al.*, 2017, Lemma 1), there exist $s = t_1\sigma \in L(\mathcal{T}'_a)$ and $\omega\sigma \in L(T')$ such that $f^e_{\mathcal{T}'_a}(x_{0,\mathcal{T}'_a}, t_1) = x^2_v$, $f_{T'}(x_{0,T'}, \omega) = x^1_v$, and $P_o(\omega) = P_o(t_1)$. In addition, we can conclude that: *(i)* $x^1_v \neq x_d$ and $f_{T'}(x_{0,T'}, \omega\sigma) \neq x_d$ since $f_{\mathcal{V}}(x_v, \rho(\sigma)) \cap X^{\mathcal{V}}_d = \emptyset$; *(ii)* $f^e_{\mathcal{T}'_a}(x_{0,\mathcal{T}'_a}, t_1) \cap US'_B \neq \emptyset$ since $x_v \in US^{\mathcal{V}}_B$, and so, $x^2_v \in US'_B$, and; *(iii)* $us_r \in f^e_{\mathcal{T}'_a}(x_{0,\mathcal{T}'_a}, t_1\sigma)$ since $f_{\mathcal{V}}(x_v, \sigma) \cap US^{\mathcal{V}}_R \neq \emptyset$. Thus, according to the definitions of T' and $\mathcal{T}'_a$, we can rewrite the above conclusions in terms of T and $\mathcal{T}_a$, as follows: *(i)* $\omega, \omega\sigma \in L(T)$, *(ii)* $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, t_1) \cap US_B \neq \emptyset$, and *(iii)* $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, t_1\sigma) \cap US_R \neq \emptyset$. Thus, $t_1\sigma$ does not satisfy condition C_{NA} of Definition 14 for $s = t_1\sigma \in L(\mathcal{T}'_a) \subseteq L(\mathcal{T}_a)$.

Notice that, according to the construction of $\mathcal{T}'_a$, the unsafe region US_R is merged into state us_r of $\mathcal{T}'_a$, which has no feasible event. This implies that, $t_1\sigma$ is the unique prefix of s such that $us_r \in f^e_{\mathcal{T}'_a}(x_{0,\mathcal{T}'_a}, t_1\sigma)$, or equivalently, $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, t_1\sigma) \cap US_R \neq \emptyset$. Therefore, we can conclude that $L(\mathcal{T}_a)$ is not NA-Secure.

($\Leftarrow$) Consider now that both Conditions C1 and C2 hold true. Then, according to Condition C1 and the definition of $\mathcal{T}'_a$, $x_{0,\mathcal{T}_a} \cap US_R = \emptyset$, which implies, according to Definitions 10 and 11, that all sequences $s \in L(\mathcal{T}_a)$ such that $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s) \cap US_R \neq \emptyset$ have one prefix $t_1\sigma \in \overline{\{s\}}$ with $t_1 \in \Sigma^\star$ and $\sigma \in \Sigma_{ca}$ such that $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, t_1) \cap US_B \neq \emptyset$, and $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, t_1\sigma) \cap US_R \neq \emptyset$, and additionally, for all $t' \in \overline{\{t_1\}}$, $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, t') \cap US_R = \emptyset$, *i.e.*, $t_1\sigma$ is the prefix of s with smallest length that leads the system to a state in the unsafe region US_R.

Notice that $t_1\sigma \in L(\mathcal{T}'_a)$ since $us_r \in f^e_{\mathcal{T}'_a}(x_{0,\mathcal{T}'_a}, t_1\sigma)$. In addition, according to the construction of $\mathcal{V}$ (CARVALHO *et al.*, 2017, Lemma 1), for all $\omega \in L(T')$ such that $P_o(\omega) = P_o(t_1)$, there is a set of states $X^\omega_{t_1} := \{(x^1_v, x^2_v) \in X_{\mathcal{V}} : x^1_v = f_{T'}(x_{0,T'}, \omega) \wedge x^2_v = f^e_{\mathcal{T}'_a}(x_{0,\mathcal{T}'_a}, t_1)\}$. Moreover, there exists at least one state $x_v \in X^\omega_{t_1}$ such that $x_v \in US^{\mathcal{V}}_B$ and $f_{\mathcal{V}}(x_v, \sigma) \cap US^{\mathcal{V}}_R \neq \emptyset$. Since T' is a deterministic automaton, then a unique state is reached in $\mathcal{V}$ from state x_v after the execution of event $\rho(\sigma)$, *i.e.*, $f_{\mathcal{V}}(x_v, \rho(\sigma)) = \{(f_{T'}(x^1_v, \rho(\sigma)), x^2_v)\}$.

Thus, since Condition C2 is by hypothesis true, then $f_{\mathcal{V}}(x_v, \rho(\sigma)) = \{(f_{T'}(x^1_v, \rho(\sigma)), x^2_v)\} \cap X^{\mathcal{V}}_d \neq \emptyset$. In addition, since $f_{\mathcal{V}}(x_v, \rho(\sigma))$ is a singleton, then $f_{\mathcal{V}}(x_v, \rho(\sigma)) \subseteq X^{\mathcal{V}}_d$, which implies, according to the definition of T', that $\omega\sigma \notin L(T)$. Therefore, it can be concluded that, for all $\omega \in L(T)$ such that $P_o(\omega) = P_o(t_1)$,

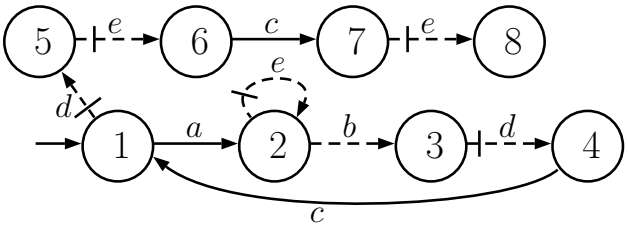

Figure 3.22: Automaton G of Example 16.

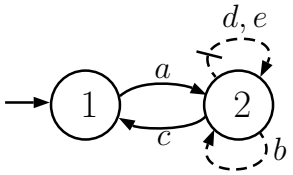

Figure 3.23: Supervisor realization H of Example 16.

$\omega\sigma \notin L(T)$. ■

Example 16 *Let us consider the ANS* $\Upsilon = (G, H, \Sigma_{vs}, \Sigma_{va})$*, where* G *and* H *are shown in Figures 3.22 and 3.23, respectively, where the set of events is defined as* $\Sigma = \{a, b, c, d, e\}$*, where the set of controllable events is* $\Sigma_c = \{d, e\}$ *and the set of observable events is* $\Sigma_o = \{a, c\}$*. Also consider the set of unsafe states to be a singleton* $X_{US} = \{8\}$*. Then, the closed-loop system model without network attacks is computed as* $T = G \parallel H$*, depicted in Figure 3.24. Let us consider that the set of vulnerable sensor and actuator channels attacks are such that,* $\Sigma_{vs} = \{c\}$ *and* $\Sigma_{va} = \{d\}$*. Then, we obtain the attacked closed-loop system model* $\mathcal{T}_a = \mathcal{G}_a \parallel H_a$ *presented in Figure 3.27. Notice that* $\Sigma_{ca} = \{e\}$ *and* $X_{US}^{\mathcal{T}_a} = \{(8,1),(8,2)\}$*.*

According to Algorithm 5, in order to obtain the verifier automaton $\mathcal{V}$ *we need first to construct automata* $\mathcal{T}_a'$ *and* T' *depicted in Figures 3.25 and 3.26, respectively. Then, we compute automaton* T_ρ'*, presented in Figure 3.28, by renaming its unobservable events. Finally, we construct verifier* $\mathcal{V}$*, whose part is depicted in Figure 3.29. The complete automaton* $\mathcal{V}$ *is not depicted in Figure 3.29 due to its size, since it has* 46 *states and* 240 *transitions.*

According to Algorithm 6, we first check if $x_{0,\mathcal{V}}^2 \neq us_r$*. Since* $x_{0,\mathcal{V}} = \{((1,1),(1,1))\}$*, we proceed to line 2, where* NA_{Sec} *is initially set as* $True$*. In the sequel, in lines 3 to 5, we define*

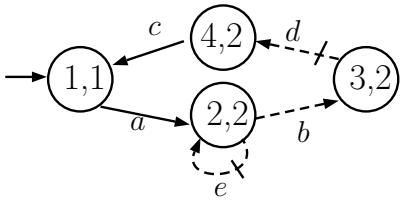

Figure 3.24: Close loop system model T of Example 16.

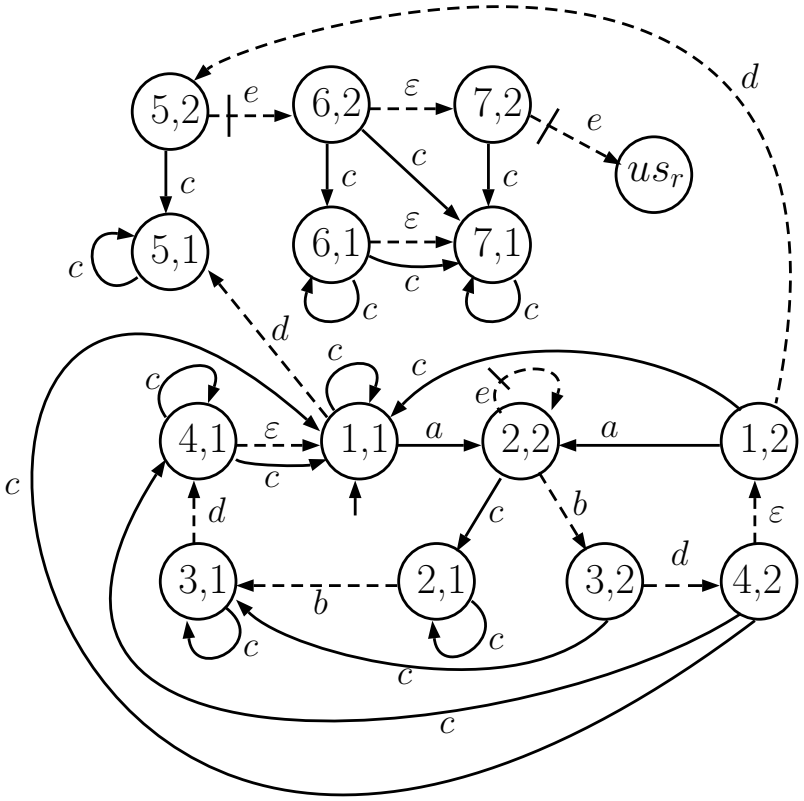

Figure 3.25: Modified attacked closed-loop system automaton $\mathcal{T}'_a$.

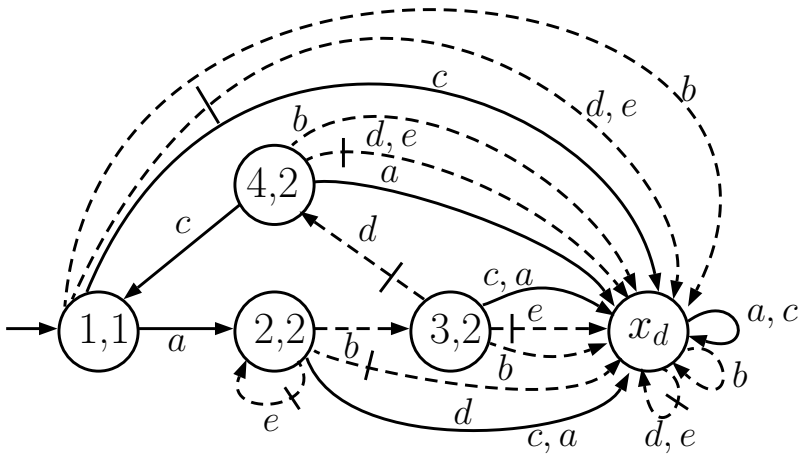

Figure 3.26: Modified closed-loop system automaton T'.

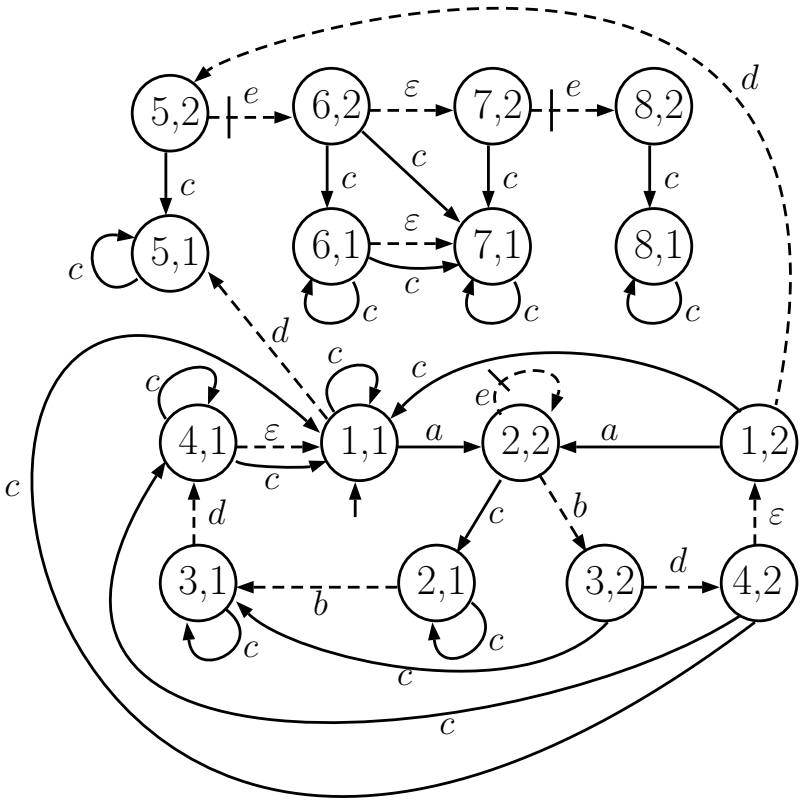

Figure 3.27: Attacked closed loop system model $\mathcal{T}_a$.

sets $US_B^{\mathcal{V}} = \{((2,2),(7,2)),((3,2),(7,2)),((4,2),(7,2)),(x_d,(7,2))\}$, $US_R^{\mathcal{V}} = \{((2,2),us_r),((3,2),us_r),((4,2),us_r),(x_d,us_r)\}$, *and* $X_d^{\mathcal{V}} = \{(x_d,(1,1)),(x_d,(1,2)),(x_d,(2,1)),(x_d,(2,2)),$ $(x_d,(3,1)),(x_d,(3,2)),(x_d,(4,1)),(x_d,(4,2)),(x_d,(5,1)),$ $(x_d,(5,2)),(x_d,(6,1)),(x_d,(6,2)),(x_d,(7,1)),(x_d,(7,2)),\quad (x_d,us_r)\}$. *Finally, we verify if there exists a transition from a state in* $US_B^{\mathcal{V}}$ *that satisfies the condition presented in line 8 of Algorithm 6. To this end, consider state* $x_v = ((2,2),(7,2)) \in US_B^{\mathcal{V}}$. *As illustrated in Figure 3.29, there is a transition labeled with event* e *from* x_v *to state* $((2,2),us_r) \in US_R^{\mathcal{V}}$, *and thus,* $f_{\mathcal{V}}(x_v,e) \cap US_R^{\mathcal{V}} \neq \emptyset$. *In addition, it can be seen, from Figure 3.29, that there is a self-loop in state* x_v *labeled with renamed event* e_ρ, *and so,* $f_{\mathcal{V}}(x_v,e_\rho) = \{((2,2),(7,2))\}$, *which implies that* $f_{\mathcal{V}}(x_v,e_\rho) \cap X_d^{\mathcal{V}} = \emptyset$. *Thus,* NA_{Sec} *is set as* $False$ *in line* 9, *and the algorithm stops. Therefore, according to Theorem 4,* $L(\mathcal{T}_a)$ *is not NA-Secure. Notice that, this result comes from the fact that state* x_v *is reached after the occurrence of sequence* $s_v = abdde \in L(\mathcal{V})$, *that corresponds to the synchronization of sequences* $t_1 = abdde \in L(\mathcal{T}_a')$ *and* $\omega = a \in L(T')$, *which can be obtained by applying projections* $P_\Sigma : \Sigma_{\mathcal{V}}^\star \to \Sigma^\star$ *and* $P_{\Sigma_\rho} : \Sigma_{\mathcal{V}}^\star \to \Sigma_\rho^\star$ *to* s_v, *respectively. Thus, by defining* $s = t_1 e$, *then* s *reaches a state in the unsafe region of* $\mathcal{T}_a$ *and violates condition* C_{NA} *of Definition 14, since* $P_o(t_1) = P_o(\omega)$ *and* $\omega e \in L(T)$. □

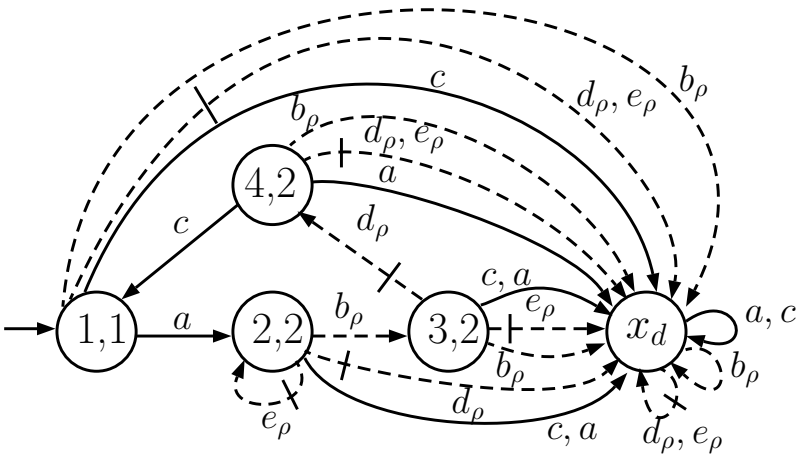

Figure 3.28: Renamed automaton T'_ρ obtained from T'.

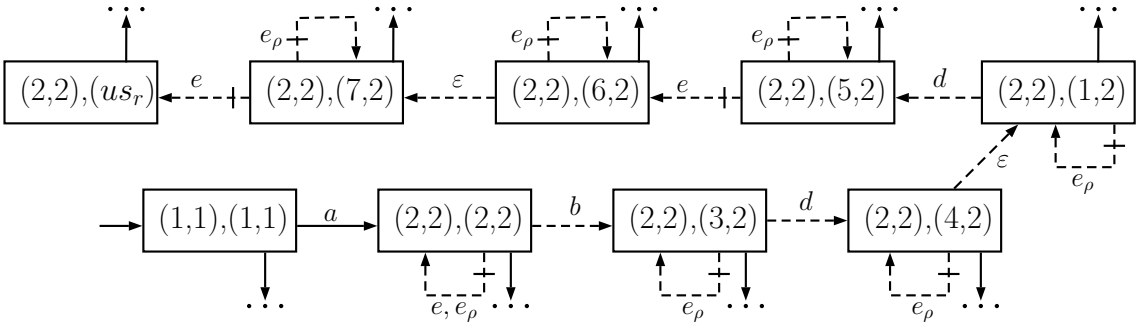

Figure 3.29: Part of Verifier automaton $\mathcal{V}$ of Example 16.

3.4.2 Online security supervisor

In this section, we propose an online synthesis of a security supervisor Ψ, that can be implemented in polynomial time, based on the attacked closed-loop system model $\mathcal{T}_a$, for NA-Secure systems. In order to do so, we first define function $\Gamma_{us} : X_{\mathcal{T}_a} \to 2^{\Sigma_{ca}}$, where, for all state $x \in X_{\mathcal{T}_a}$, $\Gamma_{us}(x) = \emptyset$, if $x \notin US_B$, and $\Gamma_{us}(x) = \{\sigma \in \Sigma_{ca} : f_{\mathcal{T}_a}(x, \sigma) \cap US_R \neq \emptyset\}$, otherwise. Notice that $\Gamma_{us}(x)$ returns all controllable events that are feasible in x and lead the system to a state in the unsafe region US_R. Thus, if a state x in the unsafe boundary US_B belongs to the current state estimate of $\mathcal{T}_a$, then the security supervisor must disable those controllable events in $\Gamma_{us}(x)$. This strategy is summarized in Algorithm 7.

In Algorithm 7, Function `controlDecision` is used for the online implementation of a security supervisor Ψ, which computes set $\Sigma_{dis} \in 2^{\Sigma_{ca}}$ formed of the events to be disabled in the plant, and the current state estimate of $\mathcal{T}_a$, denoted as $x_{c,Obs}$. In line 1, this function is executed when the system is initialized to compute the first event set Σ_{dis} to be disabled by Ψ in line 2. To this end, the inputs of `controlDecision` are defined as the attacked closed-loop system model $\mathcal{T}_a$, the empty sequence ε, and the initial state of $\mathcal{T}_a$, $x_{0,\mathcal{T}_a}$. After the observation of an event $\sigma \in \Sigma_o$ (line 3), in line 4, `controlDecision` is executed to compute the new control action to be issued by Ψ.

Thus, its inputs are set as $\mathcal{T}_a$, the last event observation σ, and the state estimate of $\mathcal{T}_a$ before the observation of σ. In line 5, Algorithm 7 returns to line 2.

In Function `controlDecision`, we first compute state set $y_{c,Obs}$ (line 7), which is formed with the states of $\mathcal{T}_a$ reached after the occurrence of transitions from states in $x_{c,Obs}$ labeled with σ. In the sequel, we update the current state estimate $x_{c,Obs}$ by assuming that all events are enabled in the plant. Based on the updated $x_{c,Obs}$, we compute set Σ_{dis} by using function Γ_{us}. If $\Sigma_{dis} \neq \emptyset$, *i.e.*, some events will be disabled by Ψ, then we recalculate the current state estimate $x_{c,Obs}$ by assuming that only those events in $\Sigma \setminus \Sigma_{dis}$ will be enabled in the plant.

Algorithm 7: Online implementation of Ψ

1 $[\Sigma_{dis}, x_{c,Obs}] =$ `controlDecision`$(\mathcal{T}_a, \varepsilon, x_{0,\mathcal{T}_a})$.
2 Disable all events in Σ_{dis}.
3 Wait for the next event observation $\sigma \in \Sigma_o$.
4 $[\Sigma_{dis}, x_{c,Obs}] =$ `controlDecision`$(\mathcal{T}_a, \sigma, x_{c,Obs})$.
5 Go back to line 2.
6 **Function** `controlDecision`$(\mathcal{T}_a, \sigma, x_{c,Obs})$**:**
7 Set $y_{c,Obs} = \{y \in X_{\mathcal{T}_a} : (\exists x \in x_{c,Obs})[y \in f_{\mathcal{T}_a}(x, \sigma)\}$.
8 Set $x_{c,Obs} = R_{uo}(y_{c,Obs}, \mathcal{T}_a, \Sigma_{uo})$.
9 Construct set $\Sigma_{dis} = \{\sigma \in \Sigma_{ca} : (\exists x \in x_{c,obs})[\sigma \in \Gamma_{us}(x)]\}$.
10 **if** $\Sigma_{dis} \neq \emptyset$ **then**
11 Recalculate the current state estimate as $x_{c,Obs} = R_{uo}(y_{c,Obs}, \mathcal{T}_a, \Sigma_{uo} \setminus \Sigma_{dis})$.
12 **end**
13 **return** $[\Sigma_{dis}, x_{c,Obs}]$

Lemma 1 *Let Ψ be the security supervisor implemented in accordance with Algorithm 7. Let sequence $s\sigma \in L(\mathcal{T}_a)$, with $s \in L(\Psi/\mathcal{T}_a)$ and $\sigma \in \Sigma$. Then, $s\sigma \notin L(\Psi/\mathcal{T}_a)$ if, and only if, $\sigma \in \Sigma_{ca}$ and there exists $s' \in L(\Psi/\mathcal{T}_a)$, not necessarily different from s, such that $P_o(s') = P_o(s)$, $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s') \cap US_B \neq \emptyset$ and $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s'\sigma) \cap US_R \neq \emptyset$.*

Proof: ($\Rightarrow$) Suppose that $s\sigma \notin L(\Psi/\mathcal{T}_a)$. Then, $\sigma \in \Psi(P_o(s))$. Thus, we can infer, from line 2 of Algorithm 7, that, $\sigma \in \Sigma_{ca}$ and, after the observation of sequence $P_o(s)$, there exist $x \in x_{c,Obs}$ such that $\sigma \in \Gamma_{us}(x)$, which implies, according to the definition of Γ_{us}, that $x \in US_B$ and $f_{\mathcal{T}_a}(x, \sigma) \cap US_R \neq \emptyset$. Since, at each iteration of Algorithm 7, the current state estimate $x_{c,Obs}$ is refined in line 11 by taking into account the events disabled by Ψ, we can conclude that there exists $s' \in L(\Psi/\mathcal{T}_a)$, not necessarily different from s, such that $x \in f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s')$ and $P_o(s') = P_o(s)$.

($\Leftarrow$) Suppose that $\sigma \in \Sigma_{ca}$, and there exists $s' \in L(\Psi/\mathcal{T}_a)$, not necessarily different from s, such that $P_o(s') = P_o(s)$, $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s') \cap US_B \neq \emptyset$ and $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s'\sigma) \cap US_R \neq \emptyset$. Thus, since $P_o(s') = P_o(s)$, after the occurrence of s, the

Table 3.2: Computational complexity of Algorithms 5, 6, and 7.

Automaton	No. of states	No. of transitions
$\mathcal{G}_a$	$\lvert X\rvert$	$\lvert X\rvert(\lvert\Sigma\rvert + 2\lvert\Sigma_{vs}\rvert)$
H_a	$\lvert X_H\rvert$	$\lvert X_H\rvert \cdot \lvert\Sigma\rvert$
$\mathcal{T}_a$	$\lvert X\rvert \cdot \lvert X_H\rvert$	$(\lvert X\rvert \cdot \lvert X_H\rvert)^2 \cdot (\lvert\Sigma\rvert + 1)$
T	$\lvert X\rvert \cdot \lvert X_H\rvert$	$\lvert X\rvert \cdot \lvert X_H\rvert \cdot \lvert\Sigma\rvert$
$\mathcal{T}'_a$	$(\lvert X\rvert - \lvert X_{US}\rvert) \cdot \lvert X_H\rvert + 1$	$\big((\lvert X\rvert - \lvert X_{US}\rvert) \cdot \lvert X_H\rvert\big)^2 \cdot (\lvert\Sigma\rvert + 1)$
T'	$\lvert X\rvert \cdot \lvert X_H\rvert + 1$	$(\lvert X\rvert \cdot \lvert X_H\rvert + 1) \cdot \lvert\Sigma\rvert$
$\mathcal{V}$	$\lvert X_{\mathcal{T}'_a}\rvert \cdot \lvert X_{T'}\rvert$	$(\lvert X_{\mathcal{T}'_a}\rvert \cdot \lvert X_{T'}\rvert)^2 \cdot (\lvert\Sigma\rvert + 1)$
Overall complexities		
	Algorithm 5	$O((\lvert X_{\mathcal{T}'_a}\rvert \cdot \lvert X_{T'}\rvert)^2 \cdot \lvert\Sigma\rvert)$
	Algorithm 6	$O(\lvert X_{\mathcal{V}}\rvert^2 \cdot \lvert\Sigma\rvert)$
	Algorithm 7	$O(\lvert X_{\mathcal{T}_a}\rvert^2 \cdot \lvert\Sigma\rvert)$

current state estimate of $\mathcal{T}_a$ includes all states in $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s')$. Therefore, according to line 9 of Algorithm 7, we can conclude that $\sigma \in \Psi(P_o(s))$, which ultimately implies that $s\sigma \notin L(\Psi/\mathcal{T}_a)$. ■

Theorem 5 *Let $\Upsilon = (G, H, \Sigma_{vs}, \Sigma_{va})$ be an NA-Secure ANS. The security supervisor Ψ implemented online in accordance with Algorithm 7 is a solution to MPSP, i.e., Ψ satisfies Requirements **R1**, **R2** and **R3** of Problem 1.*

Proof: The proof will be partitioned into three parts: in parts *(a)* and *(b)* we assume that Ψ does not satisfy Requirements **R1** and **R2**, respectively, and we show that these assumptions lead to contradictions, and; in part *(c)* we show that, any security supervisor Ψ' that is more permissive than Ψ does not satisfy Requirement **R2**.

(a) Suppose that Ψ does not satisfy Requirement **R1**, *i.e.*, $L(T) \not\subset L(\Psi/\mathcal{T}_a)$. Since $L(T) \subseteq L(\mathcal{T}_a)$ and Ψ disables only events in Σ_{ca}, then, there exists $\omega\sigma \in L(T)$, with $\omega \in \Sigma^\star$ and $\sigma \in \Sigma_{ca}$, such that $\omega \in L(\Psi/\mathcal{T}_a)$ and $\omega\sigma \notin L(\Psi/\mathcal{T}_a)$. In addition, according to Lemma 1, there exists $t_1 \in L(\Psi/\mathcal{T}_a)$ such that $P_o(t_1) = P_o(\omega)$, $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, t_1) \cap US_B \neq \emptyset$ and $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, t_1\sigma) \cap US_R \neq \emptyset$. Thus, by defining sequence $s = t_1\sigma$, we can infer that $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s) \cap US_R \neq \emptyset$ and s does not satisfy Condition C_{NA} of Definition 14, which is a contradiction since Υ is NA-secure.

(b) Suppose now that Ψ does not satisfy Requirement **R2**. Then, there exists $s' \in L(\Psi/\mathcal{T}_a)$ such that $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s') \cap X^{\mathcal{T}_a}_{US} \neq \emptyset$. Thus, according to Definition 10, there exists $s \in L(\Psi/\mathcal{T}_a)$ such that $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s) \cap US_R \neq \emptyset$. Since $\mathcal{T}_a$ is NA-secure with respect to $L(T)$, P_o and $X^{\mathcal{T}_a}_{US}$, s has at least one event $\sigma \in \Sigma_{ca}$. Thus, without loss of generality, we can partition s as $s = t_1\sigma s''$ where $t_1, s'' \in \Sigma^\star$, $\sigma \in \Sigma_{ca}$, $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, t_1\sigma) \cap US_R \neq \emptyset$ and, $\forall t' \in \overline{\{t_1\}}$, $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, t') \cap US_R = \emptyset$, *i.e.*, t_1 is the largest prefix of s that does not lead $\mathcal{T}_a$ to a state in the unsafe region US_R. Moreover, it can be seen, according to Definition 11, that $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, t_1) \cap US_B \neq \emptyset$. Notice that,

after the occurrence of sequence t_1, the current state estimate $x_{c,Obs}$, computed in line 8 of Algorithm 7, contains all states in $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, t_1)$, and thus, there will exist $x \in x_{c,Obs}$ such that $\sigma \in \Gamma_{us}(x)$. Therefore, according to line 9 of Algorithm 7, Ψ disables σ after the occurrence of t_1, which is a contradiction since, from the initial hypothesis, $s = t_1\sigma s'' \in L(\Psi/\mathcal{T}_a)$.

(c) Suppose a security supervisor Ψ' such that $L(\Psi/\mathcal{T}_a) \subset L(\Psi'/\mathcal{T}_a)$. Then, there exists $t \in L(\Psi'/\mathcal{T}_a)\backslash L(\Psi/\mathcal{T}_a)$. Thus, t can be partitioned as $t = t'\sigma t''$, with $t', t'' \in \Sigma^\star$ and $\sigma \in \Sigma_{ca}$, such that $t' \in L(\Psi/\mathcal{T}_a)$ and $t'\sigma \notin L(\Psi/\mathcal{T}_a)$. According to Lemma 1, there exists $s' \in L(\Psi/\mathcal{T}_a)$ such that $P_o(s') = P_o(t')$, $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s') \cap US_B \neq \emptyset$, and $f^e_{\mathcal{T}_a}(x_{0,\mathcal{T}_a}, s'\sigma) \cap US_R \neq \emptyset$. In addition, we can infer that $s' \in L(\Psi'/\mathcal{T}_a)$ since, according to the initial hypothesis, $L(\Psi/\mathcal{T}_a) \subset L(\Psi'/\mathcal{T}_a)$, and, since $\sigma \notin \Psi'(P_o(s')) = \Psi'(P_o(t'))$, then $s'\sigma \in L(\Psi'/\mathcal{T}_a)$. Therefore, Ψ' does not satisfy Requirement **R2**, since $\mathcal{T}_a$ reaches states in US_R after sequence $s'\sigma$. ■

Example 17 *Let us consider the same ANS Υ presented in Example 16, and consider now that event b is observable, i.e., the set of observable events is $\Sigma_o = \{a, b, c\}$. It can be seen, by using Algorithm 6, that $L(\mathcal{T}_a)$ is NA-Secure with respect to $L(T)$, P_o, and $X^{\mathcal{T}_a}_{US}$. Thus, the security supervisor presented in Algorithm 7 can be used to avoid the system from reaching unsafe states. In order to illustrate the operation of Algorithm 7, let us consider the execution of a sequence by the plant. According to Algorithm 7, we first compute, by using Function* `controlDecision`*, the initial state estimate of $\mathcal{T}_a$ and the initial control action of the security supervisor. In order to do so, we compute, in lines 7 and 8 of Function* `controlDecision`*, $y_{c,Obs} = x_{0,\mathcal{T}_a} = \{(1,1)\}$, and the initial state estimate $x_{c,Obs} = R_{uo}(\{(1,1)\}, \mathcal{T}_a, \Sigma_{uo}) = \{(1,1),(5,1)\}$. Since $\Gamma_{us}(x) = \emptyset$ for all $x \in x_{c,Obs}$, we obtain $\Sigma_{dis} = \emptyset$ in line 9. Thus, no event is initially disabled by the security supervisor Ψ. Suppose now that event $a \in \Sigma_o$ is executed by the system. Thus, after the observation of a, the state estimate is updated, by using* `controlDecision` *in line 3, to $x_{c,Obs} = \{(2,2)\}$, and, since $\Gamma_{us}((2,2)) = \emptyset$, the control action of Ψ remains equal to $\Sigma_{dis} = \emptyset$. Suppose now that event b is executed in the sequel. Thus, we obtain $y_{c,Obs} = \{(3,2)\}$ and $x_{c,Obs} = \{(3,2),(4,2),(1,2),(5,2),(6,2),(7,2),us_r\}$. Since $\Gamma_{us}((7,2)) = \{e\}$, then we obtain $\Sigma_{dis} = \{e\}$, i.e., security supervisor Ψ disables event e after the observation of sequence ab, preventing the system from reaching a state in the unsafe region. Thus, for example, the attacker may try to generate sequence $s = abdcdece \in L(G)$ by hiding the observation of both occurrences of c in s. In this case, the supervisor H enables events d and e after the observation of ab. However, after the observation of $P_o(s) = ab$, the security supervisor disables event e and only the prefix of s, $s' = abdcd$, can be executed, and the unsafe state is not reached. Notice that the attack considered in this case would be stealthy, since sequence ab belongs to $P_o(L(T))$.* □

In the next section we show that the verification of NA-security (Algorithm 6), and the implementation of the online security supervisor (Algorithm 7) can be performed with polynomial complexity, differently from the maximally permissive range control approach, which has exponential complexity.

3.4.3 Computational complexity analysis

Table 3.2 shows the maximum number of states and transitions of all automata that must be computed in order to obtain the verifier automaton $\mathcal{V}$, and to perform the online implementation of security supervisor Ψ.

Since automaton T'_ρ, computed in line 1 of Algorithm 5, has the same numbers of states and transitions as T', and $\mathcal{V} = T'_\rho \parallel \mathcal{T}'_a$, the number of transitions of $\mathcal{V}$ is bounded by $(|X_{\mathcal{T}'_a}| \cdot |X_{T'}|)^2 \cdot (|\Sigma| + 1)$. Thus, the computational complexity of Algorithm 5, is $O((|X_{\mathcal{T}'_a}| \cdot |X_{T'}|)^2 \cdot |\Sigma|)$.

The computational complexity of Algorithm 6 is determined by the loop in lines 6 to 12, where, in the worst case, all states of $\mathcal{V}$ are visited and, for each state, $|\Sigma| \cdot |X_\mathcal{V}|$ transitions are checked. Therefore, Algorithm 6 is $O(|X_\mathcal{V}|^2 \cdot |\Sigma|)$.

Notice that each line of function `controlDecision` in Algorithm 7 can be computed with linear complexity in the number of transitions of $\mathcal{T}_a$. Thus, the computational complexity of Function `controlDecision` is $O(|X_{\mathcal{T}_a}|^2 \cdot |\Sigma|)$. Therefore, both the initialization (line 1) and the iterative steps (lines 2 to 5) of Algorithm 7 are $O(|X_{\mathcal{T}_a}|^2 \cdot |\Sigma|)$.

3.5 Conclusions

In this chapter, we propose a defense strategy to thwart man-in-the-middle attacks in Cyber-Physical Systems by using the framework of Discrete-Event Systems. In order to do so we proposed the implementation of a security supervisor that is capable of preventing damages caused by attacks in sensor and/or control communication channels of Cyber-Physical Systems. Models of the plant subject to sensor channel attacks, and of the supervisor subject to control channel attacks are obtained. We also present the definition of NA-Safe controllability, that leads to a necessary and sufficient condition for the existence of a security supervisor, and propose algorithms for the verification of NA-Safe controllability and the construction of the security supervisor. A practical example is used to illustrate how to systematically compute and implement the security supervisor. The work presented in Section 3.3 is published in LIMA *et al.* (2019). However, the security supervisor proposed is not maximally permissive, therefore, in Section 3.4 we aim to create a more permissive security supervisor.

In Section 3.4, we formalize the class of Attackable Networked Systems that are NA-Secure, which is associated with the existence of a solution to the Maximally-Permissive Security Problem (MPSP). If the system is NA-Secure, then the security supervisor proposed in this work disables the occurrence of events only on a possible imminent risk scenario, and it does not affect the non-attacked closed-loop system behavior. We also propose an algorithm to verify the NA-Security property. In contrast to the approaches for MPRCP, which have exponential complexity with respect to the number of states and events of the system, the approach proposed in this work is polynomial with respect to the number of states and linear with respect to the number of events of the system. The work presented in Section 3.4 has been published in LIMA *et al.* (2021).

An extension of this work would be to consider the case when the system is not NA-Secure. In this case, it may be necessary to restrict the closed-loop behavior to guarantee that the system does not reach unsafe states. By doing so, the closed-loop system may block. Thus, a future research topic is to design a security supervisor that prevents the system from reaching blocking states and unsafe states.

In the work presented in Chapter 3, we dealt with the problem of attacks that can alter data transmitted in a sensor channel or in a supervisory control channel, where we are able to protect the system from being damaged by this attack, with the help of a security supervisor. However, the ensuring of the operation security of a system is not the only concern that needs to be taken in account when protecting a system against attacks. In Chapter 4 we consider a different kind of attack, namely eavesdrop attack, where the objective of the attacker is not to damage the system, instead the attacker wants to gather information of the system.

Chapter 4

Security against passive communication network attacks

In this work, we considered active attacks in Chapter 3, we also need to consider passive attacks in the communication channels of CPS, as depicted in Figure 4.1, *i.e.*, we consider that the communication channel between the plant and the intended receiver can be invaded by an attacker that eavesdrops the transmitted messages in order to identify the occurrence of a secret behavior of the system. This secret behavior may represent, for instance, a sequence of operations that the intruder cannot know, or the reach of a specific state in which the system is more vulnerable to an active attack.

This work is an extension of the work presented in LIMA *et al.* (2020) where it is introduced the notion of confidentiality, that encompasses the notions of privacy and utility at the same time. In the confidentiality definition, only the sender and intended receiver should be able to understand the contents of the secret transmitted message KUROSE and ROSS (2011). Since eavesdroppers may intercept the

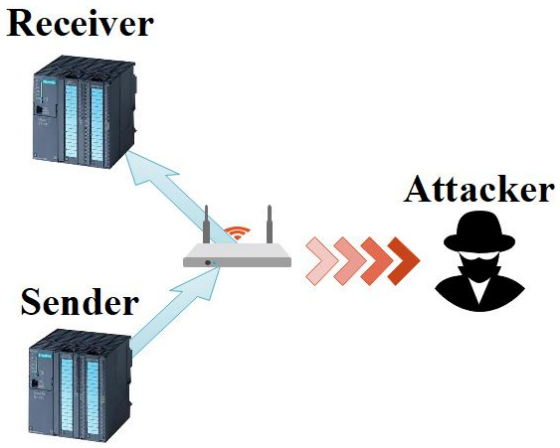

Figure 4.1: Cyber attack in industrial network.

message, the message must be somehow encrypted so that an intercepted message cannot be understood by an interceptor. Encryption is the process of transforming a message, plain text, into something illegible, cipher text, and by doing so, preventing the intruder from understanding the signals transmitted through the communication channel STALLINGS (2006). The work presented in this chapter was submitted to possible publication to Springer Journal "Discrete Event Dynamic Systems" (LIMA *et al.*, 2021 (submitted)).

In this work we introduce a new defense strategy based on cryptography that does not change the structure of the transmitted messages, *i.e.*, for every event that would be transmitted in the plain text a single event is transmitted in the cipher text. Then, the increase in the amount of transmitted data through the communication channels is avoided, which is important for some CPSs. In order to do so, the observation of events by the sensors of the plant are encrypted before transmission, and then decrypted in the receiver's site. Since the encryption is carried out in the event level, we call this type of cryptography as event-based cryptography. We also introduce the property of confidentiality of DES with respect to a secret language and an encryption function, and present a necessary and sufficient condition for this property. We propose a test to verify confidentiality of DES. In addition, we derive a method for designing an encryption function for a DES when the secret language is formed only of sequences with length bounded by a given number $n \in \mathbb{N}$. This secret language, may represent, for instance, that the secret behavior corresponds to the initialization procedure of a system, or the passage of a vehicle through a specific place where it is easier for the malicious agent to perform an attack.

This chapter is organized as follows. In Section 4.1, we propose a defense strategy against cyber attacks in the communication channels of a CPS. In Section 4.2, we formalize the property of confidentiality of DES, associated with the capability of protecting the transmitted events by using encryption functions. In Section 4.3, we introduce the transition-based encryption functions. In Section 4.4, we present a necessary and sufficient condition for confidentiality, and show a method for verifying this property of DESs. In Section 4.5 we derive a method for designing an encryption function that ensures confidentiality. Finally, in Section 4.6, the conclusions are drawn.

4.1 Defense Strategy

In the work presented in this chapter, we consider that the communication is carried out using a wired or wireless network, as shown in Figure 4.1, where the sender and the receiver can represent different entities in an industrial network, *e.g.*, the sender may be an industrial plant and the receiver a controller or supervisor, or

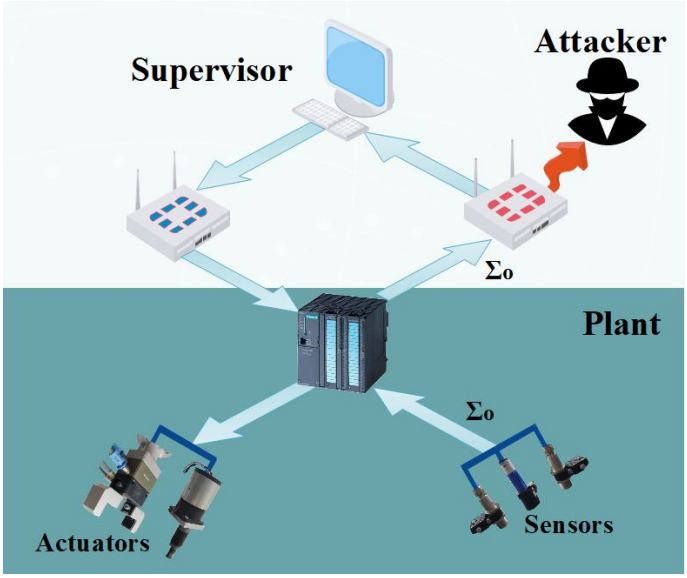

Figure 4.2: Cyber attack in the sensor channel of a supervisory control system.

the sender may be a controlled system and the receiver a diagnoser or observer. For example, in Figure 4.2, we consider the case where the sender is a plant with model P, and the receiver is a supervisor S with realization given by H. Then, the controlled system is modeled by $G = P \| H$. The channel that is used to send information, gathered by sensors, from the plant to the supervisor, is called sensor channel. Let us consider that this channel is vulnerable to attacks, as shown in Figure 4.2, and that the attacker can observe all events transmitted from the plant to the supervisor. Thus, assuming that the intruder knows the controlled plant model G, and that he/she observes only the events that are communicated through the sensor channel, then the observed model from the attacker's point of view is $G_o = Obs(G, \Sigma_o) = (X_o, \Sigma_o, f_o, x_{0,o})$, where Σ_o is the set formed of the events that are communicated through the sensor channel.

The objective of the attacker is to estimate that a sequence in the secret language, denoted as $L_S \subset L_o$, has occurred based on the observation of the events gathered by the sensors of the plant. We assume that L_S is a regular language, and make the following assumptions regarding the attacker capabilities: *(i)* the attacker can read all signals transmitted through the communication channel from the sender to the receiver, *i.e.*, the attacker and receiver observe the same events; *(ii)* the attacker knows the sender model G_o; and *(iii)* the current state of the system is unknown when the attacker starts to observe it.

In this work, we propose a defense strategy based on cryptography to guarantee that, given that a secret sequence $s \in L_S$ has been executed by the system, the attacker is not able to estimate the occurrence of any sequence in the secret language L_S. We denote by plain event any event not modified by an encryption function, and

by cipher event those modified by encryption functions. We also use this terminology for sequences and languages.

In Figure 4.3, we present the defense strategy that encrypts an observable event $\sigma \in \Sigma_o$, generating the cipher event σ_c before it is transmitted to the receiver, and the application of its inverse encryption at the receiver's site in order to recover σ from σ_c. The encryption function and decryption functions are responsible for ensuring the confidentiality of the transmitted data, and these functions can be implemented in a computer after the sender and in a computer before the receiver, or it can be implemented directly on the sender and receiver as a separated function independent from their intended operation.

Notice that the encryption function cannot mislead the receiver. Thus, the plain sequence s must be recoverable from the cipher sequence s_c, *i.e.,* the cryptography function must be invertible in order to the receiver be able to read, and eventually actuate correctly on the system. In addition, since the receiver may need to actuate just after the observation of an event executed by the system (e.g., if the receiver is a supervisor), it is important that one cipher event be transmitted to the receiver after every occurrence of an observable event in the plant. This procedure is guaranteed in this work by considering only stream encryption functions, *i.e.,* the encryption function considered in this work encrypts one observable event at a time.

It is important to remark that, differently from FRITZ *et al.* (2019), the defense strategy proposed in this work does not increase the amount of data transmitted in the communication channel, since, after encryption of an observed plain event, a cipher event is transmitted through the communication channel. Thus, by using an event-based cryptography, it is not necessary to change the structure of the transmitted data.

4.2 Confidentiality of DES

In this section, we formally present the definition of confidentiality of DES. In order to do so, it is first necessary to define function $Suf : \Sigma^* \to 2^{\Sigma^*}$, which returns the set of all suffixes of a sequence $s \in \Sigma^*$ given by:

$$Suf(s) = \{s_2 \in \Sigma^* : (\exists s_1 \in \Sigma^*)[s = s_1 s_2]\}.$$

The suffix operation can be extended to a language $L \subseteq \Sigma^*$ as $Suf(L) = \{s_2 \in \Sigma^* : (\exists s \in L)(\exists s_1 \in \Sigma^*)[s = s_1 s_2]\}$.

Since, by assumption, the attacker may start the observation of the sequence of events at any time, then the attacker observes a suffix of the cipher sequence s_c transmitted by the plant, $s' \in Suf(s_c)$. Based on the observation of s', and the

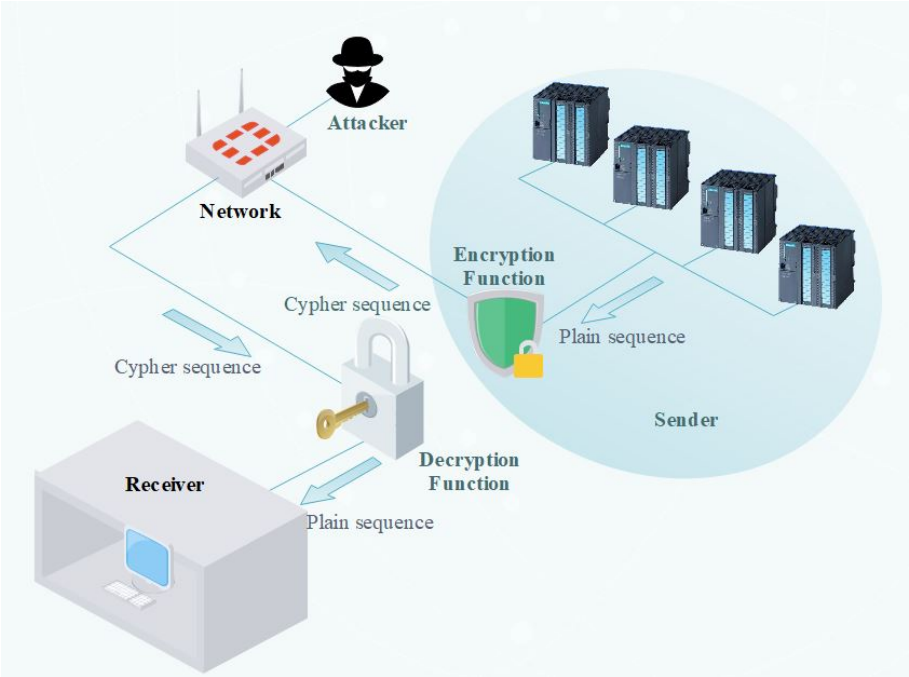

Figure 4.3: Defense Strategy.

knowledge of the system model G_o, the attacker estimates the sequences of L_o that may have occurred. If s' cannot be a suffix of any sequence in L_o, then the estimated language is equal to the empty set. On the other hand, the attacker estimates all sequences of L_o that have a suffix equal to s'. Let us denote the set of all estimated sequences from the observation of s' by $L_{e,s'} = \{s \in L_o : s' \in Suf(s)\}$. Notice that, if $L_{e,s'} \subseteq L_S$ and $L_{e,s'} \neq \emptyset$, the attacker estimates that a secret sequence has occurred. Therefore, after the occurrence of a secret sequence, it is important to mislead the attacker by either making $L_{e,s'} = \emptyset$ or $L_{e,s'} \cap (L_o \setminus L_S) \neq \emptyset$.

Example 18 *Let us consider that the language generated by the system is* $L_o = \overline{(ab + ca)c^*}$*, and that the secret language is* $L_S = \{ab, ca\}$*. Let us also consider that the system executes and transmits to the receiver sequence* $s = ca$*. Thus, depending on when the attacker starts to eavesdrop the system, any suffix of* $Suf(s) = \{\varepsilon, a, ca\}$ *can be observed. Consider that the observed sequence is* $s' = a$*. Then, since the attacker knows language* L_o*, the attacker estimates the occurrence of the sequences in* $L_{e,a} = \{a, ca\}$*. Since* $a \notin L_S$*, then the attacker is not certain about the occurrence of a secret sequence. However, if the observed sequence is* $s' = ca$*, then* $L_{e,ca} = \{ca\}$*, and the attacker is certain about the occurrence of a secret sequence.* □

The property of confidentiality depends on the definition of an encryption function $F_E : \Sigma_o^* \rightarrow \Sigma_o^*$ that transforms a plain sequence $s \in L_o \subseteq \Sigma_o^*$ into a cipher sequence $s_c \in \Sigma_o^*$. The encryption function must be invertible in order to ensure confidentiality, *i.e.*, there must exist a decryption function $F_E^{-1} : \Sigma_o^* \rightarrow \Sigma_o^*$ such that

$F_E^{-1}(F_E(s)) = s, \forall s \in L_o$. If the encryption function is applied to all sequences of L_o, then we obtain the cipher language $L_c = \{F_E(s) : s \in L_o\}$. The automaton that generates L_c is denoted as G_c.

In the sequel, we formally define the property of confidentiality of DES.

Definition 16 *A language L_o is said to be confidential with respect to the secret language $L_S \subset L_o$ and encryption function F_E if:*

$[\forall s \in L_S][\forall s' \in Suf(F_E(s))] \Rightarrow (s' \notin Suf(L_S)) \vee (s' \in Suf(L_o \setminus L_S))$. □

According to Definition 16, a system is said to be confidential if after the occurrence of a sequence $s \in L_S$: *(i)* the attacker does not estimate that a sequence in L_S has occurred, *i.e.*, $s' \notin Suf(L_S)$ for all $s' \in Suf(F_E(s))$; or *(ii)* there exists $s' \in Suf(F_E(s))$ such that $s' \in Suf(L_S)$, but there also exists a non secret sequence in the estimated language $L_{e,s'}$, *i.e.*, $s' \in Suf(L_o \setminus L_S)$. Therefore, if either the attacker cannot estimate a secret sequence by observing s', or if the attacker is uncertain about the occurrence of a secret sequence, the system is said to be confidential with respect to the secret language L_S and encryption function F_E. In other words, the confidentiality of the system is violated if, and only if, the system has executed a secret sequence and the attacker is certain that a secret sequence of L_S has occurred.

Example 19 *Let G_o be the automaton of the system depicted in Figure 4.4(a), and consider the secret language $L_S = \{ab, ca\}$. Notice that, in this example, after executing ab or ca, the system will not execute any sequence in the secret language, and therefore, the sequences generated after ab or ca do not lead to the violation of confidentiality. Let us consider an encryption function F_E which changes the observation of sequences in G_o such that sequence ab is encrypted as ba, and ca is encrypted as ab. Automaton G_c, that models the cipher language L_c, is depicted in Figure 4.4(b). In order to verify confidentiality we need to consider both secret sequences, (i) sequence ab, and (ii) sequence ca. Let us first consider that the system executes sequence ab and that the attacker has started to observe the system when the plant was in the initial state 1. Thus, the attacker observes the transmitted sequence ba. After observing ba the attacker is not able to estimate the current state of the system since there is no sequence in L_o whose suffix is ba, i.e., $L_{e,ba} = \emptyset$. Let us now suppose that the attacker has started to observe the system only after b has been transmitted, and therefore, the attacker observes only event a. In this case, $L_{e,a} = \{a, ca\}$, and the attacker does not know if the plant was in the initial state 1, and after the occurrence of event a reached state 2, or if the plant was in state 3, and is now in state 4. Thus the attacker is not certain about the occurrence of the secret sequence ca. In this case, the occurrence of sequence ab is kept confidential from the attacker by using encryption function F_E.*

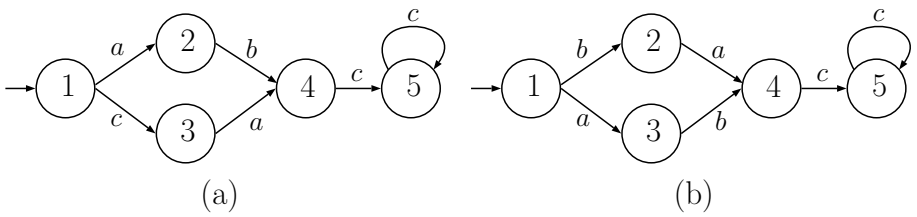

Figure 4.4: Example 19: Automata G_o and G_c.

Let us consider now that plant has generated sequence ca, observed by the attacker as ab. After observing ab the attacker would estimate $L_{e,ab} = \{ab\}$, and is certain that a secret sequence has been generated by G_o. Although ab is not the sequence generated by G_o, it is a secret sequence that leads to the same state of the plant as sequence ca. Thus, according to Definition 16, L_o is not confidential with respect to the secret language L_S and encryption function F_E. □

4.3 Transition-based encryption functions

An encryption function F_E is classified as transition-based if for all $s, s' \in L_o$ such that $f_o(x_{0,o}, s) = f_o(x_{0,o}, s')$, then $F_E(s\sigma) = F_E(s)\sigma_c$ and $F_E(s'\sigma) = F_E(s')\sigma_c$, where $\sigma_c \in \Sigma_o$, *i.e.*, the events labeling the transitions of G_o are encrypted in the same way independently of the sequence of events that has been executed before. As a consequence, the domain of transition-based encryption functions can be redefined generating the encryption function $F_T : X_o \times \Sigma_o \to \Sigma_o$. In this case, the inverse encryption function $F_T^{-1} : X_o \times \Sigma_o \to \Sigma_o$ is such that $F_T^{-1}(x, \sigma_c) = \sigma$, where $F_T(x, \sigma) = \sigma_c$. The domain and codomain of F_T can be extended to $X_o \times \Sigma_o^*$ and Σ_o^*, respectively, as $F_T(x_{0,o}, s\sigma) = F_T(F_T(x_{0,o}, s), \sigma)$, for all $s\sigma \in L_o$, and $F_T(x, \varepsilon) = \varepsilon$, for all $x \in X_o$.

Using the transition-based encryption function F_T, an automaton that generates the cipher language L_c can be obtained from G_o as $G_c = (X_o, \Sigma_o, f_c, x_{0,o})$, where $f_c(x, \sigma_c) = f_o(x, \sigma)$, with $\sigma_c = F_T(x, \sigma)$, for all $x \in X_o$ and $\sigma \in \Gamma_{G_o}(x)$.

The following result provides a necessary and sufficient condition for a transition-based encryption function be invertible.

Theorem 6 *Let G_c be the cipher automaton obtained from $G_o = (X_o, \Sigma_o, f_o, x_{0,o})$, using the transition-based encryption function F_T, as $G_c = (X_o, \Sigma_o, f_c, x_{0,o})$, where $f_c(x, \sigma_c) = f_o(x, \sigma)$, with $\sigma_c = F_T(x, \sigma)$, for all $x \in X_o$ and $\sigma \in \Gamma_{G_o}(x)$. Then, the transition-based encryption function F_T is invertible if, and only if, G_c does not have more than one transition labeled with the same event departing from any of its states.*

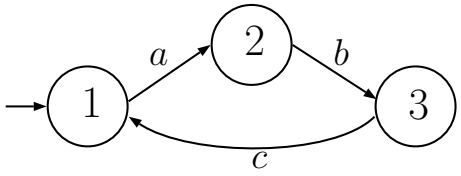

Figure 4.5: Example 20: Plant model G_o.

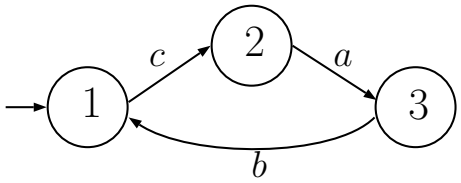

Figure 4.6: Example 20: Automaton G_c obtained using the transition-based encryption function F_T applied to automaton G_o of Figure 4.5.

Proof: ($\Rightarrow$) The proof is by contraposition. Let us consider that G_c has at two transitions departing from the same state x of G_c labeled with the same event σ_c. Since G_o is deterministic, and G_c and G_o have the same transition structure, then there are two different events $\sigma_1, \sigma_2 \in \Sigma_o$ such that $F_T(x, \sigma_1) = F_T(x, \sigma_2) = \sigma_c$, which implies that the inverse $F_T^{-1}(x, \sigma_c)$ cannot be defined.

($\Leftarrow$) Consider that each transition departing from the same state x in G_c has a different label $\sigma_c \in \Sigma_o$. Thus, since G_o is deterministic, then it is possible to obtain a one-to-one correspondence between the events labeling the transitions of G_o departing from state x, and the events labeling the corresponding transitions of G_c departing from x, which implies that F_T is invertible. ■

Example 20 *Let $G_o = (X_o, \Sigma_o, f_o, x_{0,o})$, depicted in Figure 4.5, be the system automaton, and consider that the secret language is formed of all sequences that reach state 3, i.e., $L_S = (abc)^*ab$. A transition-based encryption function F_T can be defined such that $F_T(1, a) = c$, $F_T(2, b) = a$, and $F_T(3, c) = b$. Let us now consider that the attacker observes the cipher sequence ca, transmitted by the system. Since the attacker knows the system model G_o, the attacker estimates that the system has reached state 2. However, in fact, the system has executed sequence ab, reaching the secret state 3. The model of the encrypted system G_c obtained from G_o and F_T is depicted in Figure 4.6.* □

In the following section, we present an algorithm to verify the confidentiality of DES for transition-based encryption functions.

4.4 Confidentiality verification

In order to obtain a method for the verification of confidentiality of DES, a possible approach is to associate the secret sequences of L_S with the states reached in the

system model G_o after the execution of these sequences. Let $X_S = \{x \in X_o : (\exists s \in L_S)[x = f_o(x_{0,o}, s)]\}$ denote the set of secret states of G_o, and assume that there does not exist a sequence $s' \in L_o \setminus L_S$ such that $x = f_o(x_{0,o}, s')$ and $x \in X_S$. It is important to remark that if this assumption is not true, then, since it is assumed that L_S is a regular language, it is always possible to obtain a modified automaton G'_o, by splitting the set of states as necessary, for which the assumption is valid CASSANDRAS and LAFORTUNE (2008); WU and LAFORTUNE (2013). Thus, the set of states of G_o can be partitioned as $X_o = X_S \dot{\cup} X_{NS}$, where $X_{NS} = \{x \in X_o : (\exists s \in L_o \setminus L_S)[x = f_o(x_{0,o}, s)]\}$ denotes the set of non-secret states of G_o.

In this work, we obtain a verifier for the property of confidentiality of DES inspired by the verifier for codiagnosability proposed in MOREIRA *et al.* (2011). Since we assume that the current state of the system is unknown when the attacker starts to eavesdrop the communication channels between sender and receiver, we need to present the current-state estimator of an automaton G, denoted as $\mathcal{E}(G) = (X_\mathcal{E}, \Sigma, f_\mathcal{E}, x_{0,\mathcal{E}})$ SABOORI and HADJICOSTIS (2011a). The current-state estimator of $G = (X, \Sigma, f, x_0)$ is computed in two steps: (i) obtain the nondeterministic automaton $\mathcal{G} = (X, \Sigma, f, X)$ from G, by defining the initial state of $\mathcal{G}$ as the set X; and (ii) compute $\mathcal{E}(G) = Obs(\mathcal{G}, \Sigma)$.

The set of states of the current-state estimator of the plant G_o, $\mathcal{E}(G_o) = (X_{o\mathcal{E}}, \Sigma_o, f_{o\mathcal{E}}, x_{0,o\mathcal{E}})$, can be partitioned as $X_{o\mathcal{E}} = X_{o\mathcal{E},S} \dot{\cup} X_{o\mathcal{E},NS} \dot{\cup} X_{o\mathcal{E},U}$, where $X_{o\mathcal{E},S}$ is the set formed only of secret states of X_S, $X_{o\mathcal{E},NS}$ is formed only of non-secret states of X_{NS}, and $X_{o\mathcal{E},U}$ is formed of states of X_S and X_{NS}. The language generated by $\mathcal{E}(G_o)$ is formed of all suffixes of the sequences of L_o, *i.e.*, $L(\mathcal{E}(G_o)) = Suf(L_o)$.

It is also possible to define the current-state estimator of the cipher automaton G_c as $\mathcal{E}(G_c) = (X_{c\mathcal{E}}, \Sigma_o, f_{c\mathcal{E}}, x_{0,c\mathcal{E}})$. The set of states of $\mathcal{E}(G_c)$ can also be partitioned as $X_{c\mathcal{E}} = X_{c\mathcal{E},S} \dot{\cup} X_{c\mathcal{E},NS} \dot{\cup} X_{c\mathcal{E},U}$, where $X_{c\mathcal{E},S}$ is formed only of secret states of X_S, $X_{c\mathcal{E},NS}$ is formed only of non-secret states of X_{NS}, and $X_{c\mathcal{E},U}$ is formed of states of X_S and X_{NS}. It is not difficult to see that $x_{0,c\mathcal{E}} = x_{0,o\mathcal{E}}$. The language generated by $\mathcal{E}(G_c)$ is formed of all suffixes of the cipher sequences of L_c, *i.e.*, $L(\mathcal{E}(G_c)) = Suf(L_c)$.

Algorithm 8: Confidentiality verifier

Inputs : $G_o = (X_o, \Sigma_o, f_o, x_{0,o})$ and $G_c = (X_o, \Sigma_o, f_c, x_{0,o})$.
Output: Verifier $V = (X_V, \Sigma_o, f_V, x_{0,V})$.
1 Compute $\mathcal{E}(G_o)$.
2 Compute $\mathcal{E}(G_c)$.
3 $V = \mathcal{E}(G_o) \parallel \mathcal{E}(G_c)$.

In Algorithm 8, the verifier automaton V is computed from the current-state estimators of the closed-loop system G_o and the cipher system G_c. The state estimate of the cipher system G_c represents the actual system state estimate after the execution

of a sequence $s \in Suf(L_o)$, observed as the cipher sequence $s_c \in Suf(F_T(x_{0,o}, s))$. The state estimate of G_o, on the other hand, represents what the attacker estimates from the observed cipher sequence s_c. Thus, the parallel composition between the current-state estimators compares, after each new observed event, the system states that the attacker estimates from G_o with the actual system state estimate. In the sequel we present a necessary and sufficient condition for the confidentiality of a DES based on the verifier automaton computed in Algorithm 8.

Theorem 7 *Let V be computed according to Algorithm 8. Then, L_o is confidential with respect to L_S and F_T if, and only if, for all $x_V = (x_{o\mathcal{E}}, x_{c\mathcal{E}}) \in X_V$ such that $x_{o\mathcal{E}} \in X_{o\mathcal{E},S}$, we have that $x_{c\mathcal{E}} \in X_{c\mathcal{E},NS}$.*

Proof: ($\Rightarrow$) The proof is by contraposition. Let us suppose that there is a state $(x_{o\mathcal{E}}, x_{c\mathcal{E}}) \in X_V$ such that $x_{o\mathcal{E}} \in X_{o\mathcal{E},S}$ and $x_{c\mathcal{E}} \notin X_{c\mathcal{E},NS}$. Then, the observed sequence is a suffix $s' \in Suf(L_S)$ such that $s' \notin Suf(L_o \setminus L_S)$, *i.e.*, the attacker estimates that a secret sequence has been executed by the system. If, in this case, $x_{c\mathcal{E}}$ belongs to $X_{c\mathcal{E},S}$ or $X_{c\mathcal{E},U}$, then there exists a sequence $s \in L_S$ such that $s' \in Suf(F_E(s))$, which violates Definition 16 of confidentiality of the DES.

($\Leftarrow$) If for all $x_V = (x_{o\mathcal{E}}, x_{c\mathcal{E}}) \in X_V$ such that $x_{o\mathcal{E}} \in X_{o\mathcal{E},S}$, we have that $x_{c\mathcal{E}} \in X_{c\mathcal{E},NS}$, then, there does not exist $s \in L_S$ such that $s' \in Suf(F_E(s))$, *i.e.*, the plant has not executed a secret sequence, and the attacker would estimate wrongly that a secret behavior has occurred. ■

According to Theorem 7, the confidentiality of a DES can be verified by searching in the verifier V a state $x_V = (x_{o\mathcal{E}}, x_{c\mathcal{E}})$ such that $x_{o\mathcal{E}}$ is formed only of secret states in X_S, which means that the attacker is certain that a secret sequence has been executed, and $x_{c\mathcal{E}}$ has at least one secret state of X_S, which means that it is possible that the system has indeed generated a secret sequence. If there exists a state satisfying these conditions, then the language of the system L_o is not confidential with respect to L_S and F_E. Otherwise, L_o is confidential.

Example 21 *Let us consider again automaton G_o and the cipher automaton G_c of Example 20, presented in Figures 4.5 and 4.6, respectively. The secret language is given by $L_S = (abc)^*ab$, and, consequently, $X_S = \{3\}$ and $X_{NS} = \{1, 2\}$. Following Steps 1 and 2 of Algorithm 8, we compute the current-state estimators $\mathcal{E}(G_o)$ and $\mathcal{E}(G_c)$, depicted in Figures 4.7(a) and 4.7(b), respectively. In this case, we have that $X_{o\mathcal{E},S} = X_{c\mathcal{E},S} = \{\{3\}\}$, $X_{o\mathcal{E},NS} = X_{c\mathcal{E},NS} = \{\{1\}, \{2\}\}$, and $X_{o\mathcal{E},U} = X_{c\mathcal{E},U} = \{\{1, 2, 3\}\}$.*

Verifier $V = \mathcal{E}(G_o) \parallel \mathcal{E}(G_c)$, is presented in Figure 4.8. Since the unique state $x_V = (x_{o\mathcal{E}}, x_{c\mathcal{E}})$, where $x_{o\mathcal{E}} = \{3\} \in X_{o\mathcal{E},S}$ is state $(\{3\}, \{1\})$, and $\{1\} \in X_{c\mathcal{E},NS}$, then L_o is confidential with respect to the encryption function F_E and L_S. □

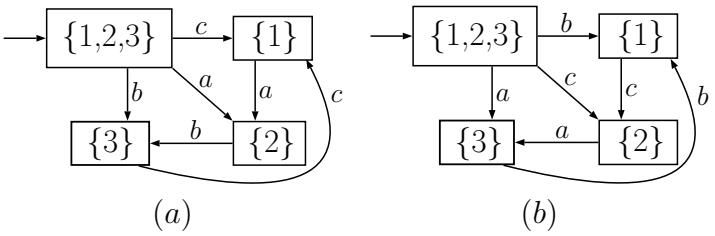

Figure 4.7: Example 21: $\mathcal{E}(G_o)$ (a) and $\mathcal{E}(G_c)$ (b).

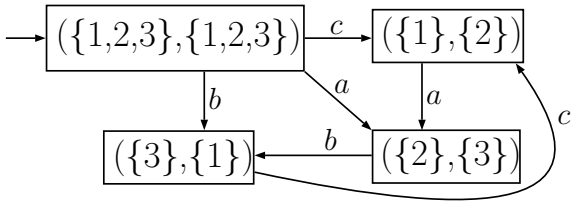

Figure 4.8: Example 21: Verifier $V = \mathcal{E}(G_o) \parallel \mathcal{E}(G_c)$.

In Section 4.5, we present a method for designing a transition-based encryption function F_T that ensures confidentiality, when it is possible, for a given system G_o.

4.5 Transition-based encryption function design

A first approach for designing a transition-based encryption function F_T is to choose, among all possibilities, an encryption function such that L_o is confidential with respect to F_T and L_S. This would require to test all possibilities of encryption functions by changing the event labels of the transitions of G_c, and verifying if L_o is confidential with respect to F_T and L_S. The problem with this strategy is that the number of possible transition-based encryption functions may be very large. In the worst-case, G_o is complete, *i.e.*, all events in Σ_o are feasible for all states of G_o. Since the cipher automaton G_c must have at most one transition labeled with the same event departing from each one of its states to ensure that F_T is invertible, then for each state $x \in X_o$ of G_c, it is only possible to permute the events that labels the transitions that depart from x. Thus, for each state x there are $|\Sigma_o|!$ possible encryptions, considering only this state. If we consider now all states of G_c, then there exist $(|\Sigma_o|!)^{|X_o|}$ possible transition-based encryption functions for G_o, which makes the direct search for one encryption function that ensures confidentiality infeasible when G_o has a large number of events and states. Notice that, the brute force method has complexity $O((|\Sigma_o|!)^{|X_o|} \times 2^{2|X_o|} \times |\Sigma_o|)$, since the computation of verifier V has complexity $O(2^{2|X_o|} \times |\Sigma_o|)$.

In this section, we present a method for computing a transition-based encryption function F_T, if it exists, when the length of the secret sequences of L_S are bounded

by a given number $n \in \mathbb{N}$. It is important to remark that the secret language L_S may represent, in this case, that the secret states belong to the initialization procedure of a system, or the reach of a secret state that represents a given place in the trajectory of a vehicle, or even any secret language in an acyclic system. The proposed method implements a backtracking strategy which may avoid testing some encryption functions, reducing the computational cost of obtaining F_T. In order to do so, let us first define a state-mapping automaton, denoted by G_m, created from G_o as, $G_m = (X_o, \Sigma_m, f_m, x_{0,o})$, where $\Sigma_m = \{\sigma_x : \sigma \in \Gamma_{G_o}(x)\}$, and $f_m(x, \sigma_x) = f_o(x, \sigma)$, for all $x \in X_o$ and $\sigma \in \Gamma_{G_o}(x)$. Notice that G_m is equal to G_o except that the events of G_m are the events of G_o renamed to map the states from which the transitions depart. By doing so, since G_o is deterministic, then each transition of G_m, labeled with σ_x is uniquely identified by the event σ and associated state x. The secret states of G_m are defined as the same secret states of G_o.

In Algorithm 9, we define a set of rules that the encryption of events associated with each transition of G_m must satisfy to ensure system confidentiality, generating an encrypted automaton G_c. In order to do so, it is important to define the *height* of a tree T, and the *depth* of a node y of T, as the number of edges between the root node and its furthest leaf, and the number of edges between the root node and y, respectively COMER (2009).

In Algorithm 9, in Steps 1 to 9, we compute the set of sequences that the attacker declares as secret sequences, L_{DS}, which is formed of sequences that may compromise the system confidentiality, *i.e.*, if the attacker observes any sequence in L_{DS}, then he/she is certain that the system is at a secret state. In addition, according to Definition 16, any suffix s' of any ciphered secret sequence may not belong to $Suf(L_S)$ or it must have at least one non-secret sequence with the same suffix s'. Notice that, if $t \in L_{DS}$ leads the estimator of the system $\mathcal{E}(G_o)$ from its initial state $x_{0,o\mathcal{E}}$ to a completely secret state $x_{cs} \in X_{o\mathcal{E},S}$, then, both $t \in Suf(L_S)$ and $t \notin Suf(L_o \setminus L_S)$. Thus, in order for a system to be confidential for all $s \in L_S$, any sequence $s' \in Suf(F_T(x_{0,o}, s))$ must not belong to L_{DS}. In Step 10 of Algorithm 9 we compute the state-mapping automaton G_m, which is an auxiliary automaton in order to construct the rules for ensuring confidentiality. In Step 11 the reversed automaton of G_m is calculated. In Step 12 we initialize the list of rules R. In Steps 13 to 23, we construct trees T_{x_s} associated with each secret state $x_s \in X_S$, whose nodes are labeled with pairs composed of a state x and a path such that it is possible to reach the secret state x_s from x following this path. After identifying all sequences s of L_{DS}, in Steps 24 to 26 a rule is added to R that states that every ciphered sequence s_c of length n_s, generated from any suffix of G_m that reaches a secret state x_s, must be different from the sequences $s \in L_{DS}$ of the same length n_s, *i.e.*, $[s_c \neq s]$. Finally, in Steps 29 to 31, additional rules are added to ensure

Algorithm 9: Confidentiality verification rules

Inputs : L_S, $G_o = (X_o, \Sigma_o, f_o, x_{0,o})$.
Output: List of rules R.

Compute $\mathcal{E}(G_o) = (X_{o\mathcal{E}}, \Sigma_o, f_{o\mathcal{E}}, x_{0,o\mathcal{E}})$.
Define $L_{DS} = \emptyset$
for *each sequence* s *in* L_S **do**
  for *each* $s' \in Suf(s)$ **do**
    if $f_{o\mathcal{E}}(x_{0,o\mathcal{E}}, s') = x_s$ *where* $x_s \in X_{o\mathcal{E},S}$ **then**
      $L_{DS} \leftarrow L_{DS} \cup \{s'\}$
    end
  end
end
Compute the state-mapping automaton G_m
Compute the reverse of G_m, $G_m^r = (X_o, \Sigma_m, f_m^r, x_{0,o})$.
Initialize the list of rules R
Create a tree T_{x_s} for each secret state $x_s \in X_S$, formed only of the root associated with the pair (x_s, p_{x_s}), where $p_{x_s} = (x_s, \varepsilon, x_s)$.
for *each sequence* s *in* L_{DS} **do**
  Set $n_s = \|s\|$
  for *each* $x_s \in X_S$ **do**
    while $n_s >$ *height*(T_{x_s}) **do**
      for *each node* y *of* T_{x_s}, *whose associated pair is* (x, p_y), *such that* *depth*(y) = *height*(T_{x_s}) **do**
        for *each transition* $x' = f_m^r(x, \sigma_{x'})$ *of* G_m^r **do**
          Create a new node y' as a child of y labeled with $(x', p_{y'})$ where $p_{y'} = (x', \sigma, x) \cdot p_y$.
        end
      end
    end
    for *each node* y *of* T_{x_s} *such that* *depth*$(y) = n_s$, *whose associated pair is* (x, p_y) *with* $p_y = (y_1, \sigma_1, \ldots, \sigma_{k-1}, y_k)$, **do**
      Add to R the rule $[s_c \neq s]$, where $s_c = F_T(y_1, \sigma_1)F_T(y_2, \sigma_2)\ldots F_T(y_{k-1}, \sigma_{k-1})$
    end
  end
end
for *each state* $x \in X_o$ **do**
  Add to R the rule $[F_T(x, \sigma_1) \neq F_T(x, \sigma_2)]$ for all $\sigma_1, \sigma_2 \in \Gamma_{G_o}(x)$ where $\sigma_1 \neq \sigma_2$.
end

that the encryption function F_T is deterministic, *i.e.*, if the rules created in Steps 29 to 31 are not satisfied, then F_T is not invertible, and therefore, it is not a valid transition-based encryption function.

Theorem 8 *L_o is confidential with respect to L_S and a transition-based encryption function F_T if, and only if, all rules of R generated by using Algorithm 9 are satisfied.*

Proof: Notice that if the rules of Steps 29 to 31 are not satisfied, then F_T is not invertible, which means that it is not an encryption function by definition. Thus, we assume from now on that the rules created in Steps 29 to 31 are all satisfied.

($\Rightarrow$) The proof is by contraposition. Let us consider that a rule of R, $[s_c \neq s]$, is not satisfied. Then, there exist a sequence $s \in L_{DS}$ and a ciphered sequence s_c, obtained by using the transition-based encryption function F_T, such that $s = s_c$. Since $s \in L_{DS}$, then, if the attacker starts eavesdropping the communication channel and observes $s_c = s$, he/she is certain that the system has reached a secret state of X_S. Since, according to Steps 24 to 26 of Algorithm 9, the ciphered sequence s_c is a suffix of a sequence of G_m that actually reaches a secret state, then L_o is not confidential with respect to L_S and F_T.

($\Leftarrow$) The proof is by contraposition. Now let us consider that the system is not confidential, *i.e.*, there exists $s \in L_S$ and $s' \in Suf(F_T(x_{0,o}, s))$, such that $s' \in Suf(L_S)$ and $s' \notin Suf(L_o \setminus L_S)$. Since $s' \in Suf(L_S)$ and $s' \notin Suf(L_o \setminus L_S)$, then, in Steps 3 to 9 of Algorithm 9, s' is added to L_{DS}. Since $s' \in L_{DS}$, then there exists a sequence $s'' \in Suf(s)$ with the same length as s', such that there exists a node y in a tree T_{x_s}, labeled with the pair (x, p_y), whose sequence of events associated to p_y is s''. Since $s' \in Suf(F_T(x_{0,o}, s))$ and $\|s'\| = \|s''\|$, then $s' = F_T(x, s'')$. This implies, in Step 25, in the creation of the rule $[s_c \neq s']$ associated with node y, where $s_c = F_T(x, s'') = s'$, which cannot be satisfied. ■

Example 22 *Consider the plant G_o of Example 19, depicted in Figure 4.4(a), reproduced again in Figure 4.9, where state 4 is the unique secret state. Notice that, if we consider the encryption function that generates G_c depicted in Figure 4.4(b), the system is not confidential, as shown in Example 19. In this regard we can use G_o and the secret language $L_S = \{ab, ca\}$ to create a set of rules so that we can calculate an encryption function F_T such that L_o is confidential with respect to F_T and L_S. Firstly we compute $\mathcal{E}(G_o)$, which is depicted in Figure 4.10. From this estimator and the secret language L_S we compute $L_{DS} = \{b, ab, ca\}$. The next step is to compute the state-mapping automaton G_m, depicted in Figure 4.11, based on G_o. Notice that the transitions of G_m are labeled with events $\sigma_x \in \Sigma_m$, where $x \in X_o$. From automaton G_m, the reversed automaton G_m^r, depicted in Figure 4.12, is computed. Then, the tree T_4 is obtained. The root node of T_4 (depth 0), is associated with the unique secret*

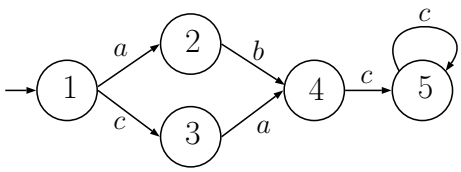

Figure 4.9: Example 22: Automaton G_o.

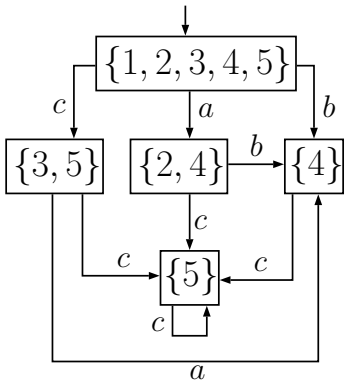

Figure 4.10: Example 22: $\mathcal{E}(G_o)$.

state 4 *and path* $(4, \varepsilon, 4)$. *Then, considering transition* $f_m^r(4, b_2) = 2$, *the node with depth 1 with label* $(2, (2, b, 4, \varepsilon, 4))$ *is created. Continuing the steps of Algorithm 9, the complete tree* T_4, *presented in Figure 4.13, is computed. From tree* T_4, *the rules that must be satisfied to ensure confidentiality are obtained. In order to do so, each sequence* $s \in L_{DS}$ *generates one rule associated with each node of* T_4 *with depth equal to the length of* s. *For example, for* $b \in L_{DS}$, *the rules* $[F_T(2, b) \neq b]$ *and* $[F_T(3, a) \neq b]$ *are created. Thus, following the Steps 24 to 26, the list of rules is computed as* $R = [[F_T(2, b) \neq b]$, $[F_T(3, a) \neq b]$, $[F_T(1, a)F_T(2, b) \neq ab]$, $[F_T(1, c)F_T(3, a) \neq ab]$, $[F_T(1, a)F_T(2, b) \neq ca]$, $[F_T(1, c)F_T(3, a) \neq ca]$, $[F_T(1, a) \neq F_T(1, c)]]$, *where the rule* $[F_T(1, a) \neq F_T(1, c)]$ *is added in Steps* 29 *to* 31 *of Algorithm 9 to ensure that the encryption function be invertible.*

Notice that, the encryption function F_T *such that* $F_T(1, a) = b$, $F_T(1, c) = a$, $F_T(2, b) = a$, *and* $F_T(3, a) = c$ *satisfies all rules of* R. *Thus, defining* $F_T(4, c) = c$ *and* $F_T(5, c) = c$ *to complete the encryption function, then, according to Theorem 8,* L_o *is confidential with respect to* F_T *and* L_S. □

Notice that Algorithm 9 provides all rules that the encryption function F_T must satisfy to ensure confidentiality. However, depending on the system complexity, it

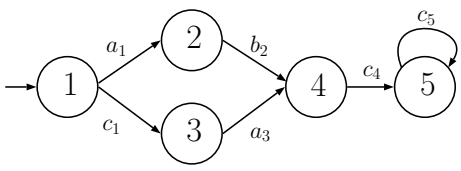

Figure 4.11: Example 22: G_m.

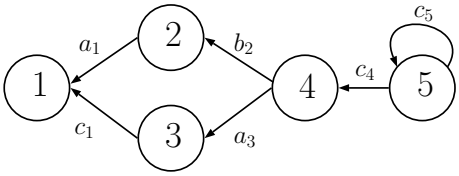

Figure 4.12: Example 22: G_m^r.

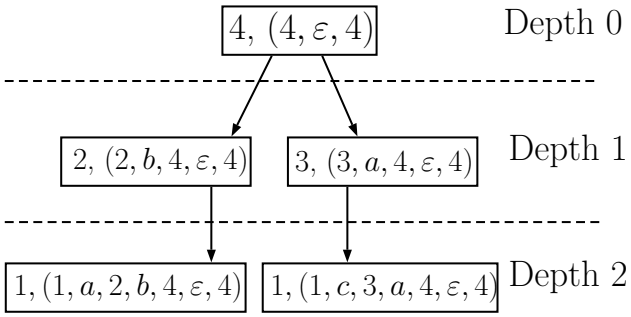

Figure 4.13: Example 22: T_4.

may be difficult to find a solution that satisfy all rules, since there are, in the worst case, $(|\Sigma_o|!)^{|X_o|}$ transition-based encryption function candidates. In these cases, we can use the recursive Algorithm 10, which uses a backtracking strategy (BIERE *et al.*, 2009; KNUTH, 1975; KUMAR, 1992) to find a solution F_T, if it exists. The main advantage of the backtracking is that it ensures correctness by enumerating all possibilities, and it also ensures efficiency by never visiting a node more than once (SKIENA, 2020). Algorithm 10 may reduce the computational cost of finding a solution F_T, since it can avoid, in some cases, testing several encryption function candidates.

In Step 1 of Algorithm 10, function F_T' is created associating to each pair $(x, \sigma) \in X_o \times \Sigma_o$, such that $f_o(x, \sigma)$ is defined, event $\sigma_x \in \Sigma_m$, *i.e.*, event σ of transition (x, σ) of G_o is encrypted with the label of the corresponding transition of G_m, σ_x. Then, in Step 2, the initialization of function C is carried out flagging all transitions of G_o with number 0. Function C is used in the algorithm to flag the transitions of G_o that have been explored with 1, and the others that have not been explored with 0. In Step 3, the recursive Function **FTSearch** is called, where the outputs of this function are the encryption function candidate F_T and the boolean variable D. If in the output of function **FTSearch** in Step 3 of Algorithm 10, D is true, then F_T is an encryption function that satisfies all rules. On the other hand, if D is false, then there does not exist a transition-based encryption function F_T such that L_o is confidential with respect to F_T and L_S.

In Function **FTSearch**, a recursive search for an encryption function F_T that ensures confidentiality is carried out. In Steps 1 and 2 of Function **FTSearch**, a transition (x, σ) that has not been explored is chosen, and then, in Step 3, this

Algorithm 10: Computation of the transition-based encryption function F_T

Inputs : Σ_o, Σ_m, R, f_o.
Output: Transition-based encryption function F_T.

Create function $F'_T : X_o \times \Sigma_o \to \Sigma_o \cup \Sigma_m$, where $F'_T(x,\sigma) = \sigma_x$, $\forall (x,\sigma) \in X_o \times \Sigma_o$ such that $f_o(x,\sigma)$ is defined, and $F'_T(x,\sigma)$ is undefined, otherwise
Create function $C : X_o \times \Sigma_o \to \{0,1\}$, where $C(x,\sigma) = 0$, $\forall (x,\sigma) \in X_o \times \Sigma_o$ such that $f_o(x,\sigma)$ is defined, and $C(x,\sigma)$ is undefined, otherwise
$[F_T, D] =$ **FTSearch**(F'_T, Σ_o, R, C)
if $D = False$ **then**
    F_T does not exist
end

Function $[F''_T, D] =$ **FTSearch** (F'_T, Σ_o, R, C)*:*
    if $\exists (x,\sigma) \in X_o \times \Sigma_o$*:* $C(x,\sigma) = 0$ **then**
        Choose a transition $(x,\sigma) \in X_o \times \Sigma_o$ such that $C(x,\sigma) = 0$
        Define $C(x,\sigma) = 1$
        $D = False$
        for $\sigma' \in \Sigma_o$ **do**
            if D=*False* **then**
                Define $D = True$
                Define $F'_T(x,\sigma) = \sigma'$
                $F''_T \leftarrow F'_T$
                for *each rule* $r = [s_c \neq s]$ *of* R **do**
                    Compute s_c using F'_T instead of F_T
                    if $[s_c = s]$ **then**
                        $D = False$
                    end
                end
                if $D = True$ **then**
                    $[F''_T, D]$=**FTSearch**(F'_T, Σ_o, R, C)
                end
            end
        end
    else
        $F''_T = F'_T$
        $D = True$
    end
end

transition is flagged as already explored, *i.e.*, $C(x,\sigma) = 1$. In Step 5, an event $\sigma' \in \Sigma_o$ is chosen to encrypt event σ such that $F'_T(x,\sigma) = \sigma'$. Then, in Steps 10 to 15, it is verified if this choice violates any rule in R considering F'_T as the encryption function. It is important to remark that some transitions (x,σ) are encrypted as σ_x, which implies that all rules such that σ_x belongs to s_c are clearly different from s, and consequently, are true. If some rule is false, then another event $\sigma' \in \Sigma_o$ is chosen for encrypting σ. On the other hand, if all rules are satisfied, then, in Step 17, Function **FTSearch** is called again in order to encrypt another transition that has not been explored yet. It is important to remark that the algorithm performs as a depth-first search. Thus, at each step of the algorithm, we verify if the current choice for the encryption function until this step is valid. If any rule is violated, then all other encryption function candidates that have the same encryption for the transitions already explored, are also not valid, and are not checked in Algorithm 10. In the exhaustive search method, we would check all possible candidates since we do not have this information. This reduces the number of verifications that are carried out in comparison with the exhaustive search algorithm. If a solution to the problem is found, *i.e.*, the algorithm finds a valid encryption function after going through all transitions of G_o, then, in Steps 22 and 23, Function **FTSearch** returns the encryption function and the boolean variable D true.

Example 23 *Let us consider the plant G_o of Example 19, depicted in Figure 4.9, and $L_S = \{ab, ca\}$. As seen in Example 21 the list of rules R is given as*

$$\begin{aligned} R \;=\; & [[F_T(2,b) \neq b], [F_T(3,a) \neq b], [F_T(1,a)F_T(2,b) \neq ab], [F_T(1,c)F_T(3,a) \neq ab], \\ & [F_T(1,a)F_T(2,b) \;\neq\; ca], [F_T(1,c)F_T(3,a) \;\neq\; ca], [F_T(1,a) \neq F_T(1,c)]] \,. \end{aligned}$$

*Then, it is possible to use Algorithm 10 to find a transition-based encryption function F_T such that L_o is confidential with respect to F_T and L_S. In order to do so, In Step 1 of Algorithm 10 we initialize function F'_T as $F'_T(1,a) = a_1, F'_T(1,c) = c_1, F'_T(2,b) = b_2, F'_T(3,a) = a_3, F'_T(4,c) = c_4, F'_T(5,c) = c_5$, where $\Sigma_m = \{a_1, c_1, b_2, a_3, c_4, c_5\}$. In Step 2, we initialize function C as $C(1,a) = C(1,c) = C(2,b) = C(3,a) = C(4,c) = C(5,c) = 0$, and $C(x,\sigma)$ is undefined for all other pairs $(x,\sigma) \in X_o \times \Sigma_o$. Then, in Step 3 function **FTSearch** is called. Considering now Function **FTSearch**, in Steps 1 and 2 we choose a transition (x,σ) such that $C(x,\sigma) = 0$. Let us consider that $(x,\sigma) = (1,a)$ has been chosen. Then, in Steps 3 and 4 of Function **FTSearch** we define $C(1,a) = 1$ and D as $False$. In Step 5 of Function **FTSearch** an event $\sigma' \in \Sigma_o$ is chosen. Let us consider that $\sigma' = a$. Then, in Step 8, $F'_T(1,a) = a$, and, in Step 9, we define F''_T as a copy of F'_T. In Steps 10 to 15, each rule of R is tested considering the encryption provided by F'_T, i.e., $R = [[b_2 \neq b], [a_3 \neq b], [ab_2 \neq ab], [c_1a_3 \;\neq\; ab], [ab_2 \;\neq\; ca], [c_1a_3 \;\neq\; ca], [a \neq c_1]]$. It is important to*

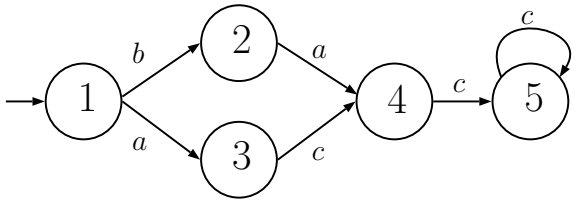

Figure 4.14: Example 23: Cipher automaton G_c corresponding to the encryption function F_T obtained using Algorithm 10.

remark that events c_1, b_2, a_3 are not defined in Σ_o, therefore, for a rule $[s_c \neq s]$, if c_1, b_2, or a_3 appears in s_c, the rule is automatically satisfied since $\Sigma_o \cap \Sigma_m = \emptyset$. In this case all rules are satisfied, and function **FTSearch** *is called again. Let us consider that the transition (x, σ) chosen now is $(1, c)$. Then, in Step 5 we can choose $\sigma' = a$, and define $F'_T(1, c) = a$. Thus, $F''_T = F'_T$ is such that $F''_T(1, a) = a, F''_T(1, c) = a, F''_T(2, b) = b_2, F''_T(3, a) = a_3, F''_T(4, c) = c_4, F''_T(5, c) = c_5$. In Steps 10 to 15 each rule is computed with the new encryption function candidate F'_T which is updated to $R = [[b_2 \neq b], [a_3 \neq b], [ab_2 \neq ab], [aa_3 \neq ab], [ab_2 \neq ca], [aa_3 \neq ca], [a \neq a]]$. In this case, the rule $[a \neq a]$ is $False$, which shows that $(1, c)$ cannot be encrypted as a, which implies that in Step 13 D is set as $False$. Since D is false, then the function returns to Step 5 to choose another $\sigma' \in \Sigma_o$. Let us consider that now $\sigma' = b$. Then, $F'_T(1, c) = b$, and by consequence, the new list of rules is given as $R = [[b_2 \neq b], [a_3 \neq b], [ab_2 \neq ab], [ba_3 \neq ab], [ab_2 \neq ca], [ba_3 \neq ca], [a \neq b]]$. In this case all rules are satisfied, and therefore, in Step 17 we call a new instance of* ***FTSearch****. Let us consider that transition $(2, b)$ is chosen, and event $\sigma' = a$. Hence, $F'_T(2, b) = a$ and the updated list of rules is $R = [[a \neq b], [a_3 \neq b], [aa \neq ab], [ba_3 \neq ab], [aa \neq ca], [ba_3 \neq ca], [a \neq b]]$. Since all rules are satisfied, then a new instance of* ***FTSearch*** *is called. Now, let us consider that transition $(3, a)$ is chosen, and $\sigma' = a$. Then the list of rules is updated to $R = [[a \neq b], [a \neq b], [aa \neq ab], [ba \neq ab], [aa \neq ca], [ba \neq ca], [a \neq b]]$. Since no rules are violated, we call again* ***FTSearch****. Since the only transitions remaining to be explored are $(4, c)$ and $(5, c)$, and all rules have already been satisfied, then $F'_T(4, c)$ and $F'_T(5, c)$ can be equal to any event $\sigma \in \Sigma_o$. Let us consider that $F'_T(4, c) = F'_T(5, c) = a$.*

In Algorithm 10, function ***FTSearch*** *returns the values $D = True$ and $F_T(1, a) = a$, $F_T(1, c) = b$, $F_T(2, b) = a$, $F_T(3, a) = a$, $F_T(4, c) = a$, and $F_T(5, c) = a$, which is a transition-based encryption function that satisfies all rules R, and therefore, is a solution to the confidentiality problem.*

It is important to remark that this choice of F_T ensures the confidentiality of the system language, as expected according to Theorem 8, but also leads the attacker to estimate the empty set depending on the observed sequence of events. In order to see this fact, let us construct the cipher automaton G_c using the encryption function F_T,

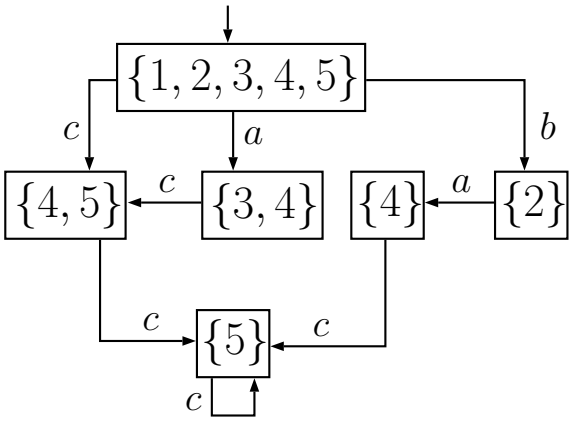

Figure 4.15: Example 23: Current state estimator of automaton G_c, $\mathcal{E}(G_c)$.

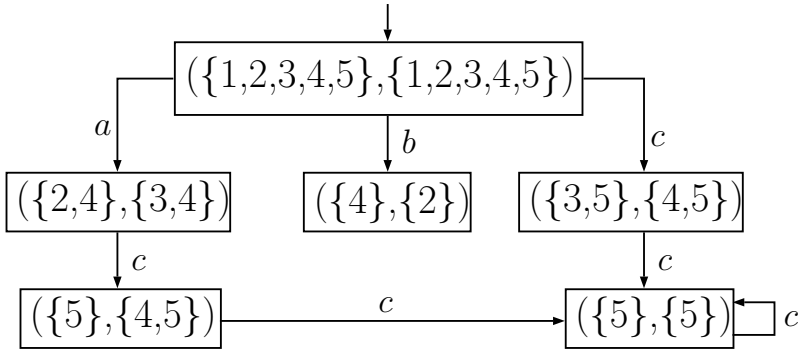

Figure 4.16: Example 23: Verifier automaton $V = \mathcal{E}(G_o) \| \mathcal{E}(G_c)$.

depicted in Figure 4.14. Then, as presented in Section 4.4, it is possible to test the confidentiality by constructing the current-state estimator of G_o, $\mathcal{E}(G_o)$, depicted in Figure 4.10, and the current-state estimator of G_c, $\mathcal{E}(G_c)$, depicted in Figure 4.15. Then the verifier can be constructed as $V = \mathcal{E}(G_o) \| \mathcal{E}(G_c)$, which is shown in Figure 4.16. Notice that, there is only one state (x_o, x_c) in V such that $x_o = \{4\}$, i.e., the attacker believes that a secret sequence has occurred when the observed sequence is b. However, after transmitting b the state actually reached by the system is state 2. In addition, after the transmission of b, it is possible for the system to transmit event a, as shown in Figure 4.14. In this case, the attacker estimates the empty set. This fact is clear from Figure 4.16, since sequence ba is not feasible in the verifier.

It is also important to remark that, in this example, there are 486 encryption function candidates. However, using Algorithm 10, only seven verifications of the rules were needed to obtain an encryption function F_T that ensures the confidentiality of L_o, which shows that Algorithm 10 may significantly reduce the computational cost of computing F_T. □

4.6 Conclusions

In the work presented in this chapter, we propose a defense strategy, based on an event-based cryptography, to prevent attackers from obtaining secret information from the communication channel between sender and receiver of a CPS. We define transition-based encryption functions, which changes the transmitted events in order to prevent the attacker from correctly estimating that a secret sequence has been executed by the system. This encryption function must be invertible in order to the receiver be capable of recovering the sequence generated by the sender. We also introduce the notion of confidentiality of DES, associated with the capability of the encrypted system to hide a secret from the attacker. A method to verify this property is proposed, and we present a method for designing an encryption function to ensure the confidentiality of a cyber-physical system with respect to a secret language formed only of sequences with length bounded by a given number. Part of the work presented in Chapter 4 is presented in LIMA *et al.* (2020), and an extended version of this work is currently under submission LIMA *et al.* (2021 (submitted).

Chapter 5

Conclusions

Considering the protection against active attacks, *i.e.*, attacks that alter data in the communication channel in a Cyber-Physical System, we considered in this work two different approaches, (i) the NA-Safe Controllability case, where, after the attack detection, we are able to prevent damages caused by the attack without altering the non-attacked behavior of the system. This work was published in LIMA *et al.* (2019). and (ii) the NA-Security approach, where we considered the possibility that damages caused by active attacks can be prevented while ensuring maximally permissibility. We showed that for a class of systems, denoted as NA-Secure Systems, the maximally permissive security supervisor can be computed in polynomial time. This approach is addressed in LIMA *et al.* (2021).

Regarding passive attacks, *i.e.*, attacks where the intruder only gather information without altering data from the attacked channel, the property of confidentiality is proposed. As defined in Chapter 4, the confidentiality property is not trivially verified. A direct verification, for a given system G, secret language L_S, and encryption function F_E, would be to analyze all secret sequences and their suffixes, and compare with all encrypted sequences, since a single sequence can make the whole system not confidential. Therefore, we propose in Chapter 4 a method to verify this property. This approach is addressed in LIMA *et al.* (2020) whose expanded version is submitted LIMA *et al.* (2021 (submitted)

Additionally, the definition of confidentiality proposed in Chapter 4 based on encryption functions, therefore, it is important to represent encryption functions on Discrete Event Systems (DES). Thus, we proposed a method to generate an encryption functions for discrete event systems which ensures confidentiality of the system.

Summarily, the main contributions of this work are listed in the sequel:

- We analyzed when it is possible to prevent damages caused by active attack even without altering the non-attacked behavior, *i.e.*, when a system is NA-

Safe Controllable, and then, created a security supervisor capable of preventing this damages (LIMA *et al.*, 2019).

- We optimized the security supervisor for ensuring maximally permissibility, *i.e.*, Computed security supervisors for the NA-Security systems. It is important to remark that, this approach can be computed in polynomial time LIMA *et al.* (2021).

- We proposed the formal definition property of Confidentiality for Discrete Event Systems, related to only the sender and the intender receiver being able to understand the message in a channel (LIMA *et al.*, 2021 (submitted, 2020).

- We proposed conditions to verify the property of Confidentiality of Discrete Event Systems, for a given plant, secret language, and encryption function (LIMA *et al.*, 2021 (submitted, 2020).

- We presented a method to generate transition based event-based encryption functions (LIMA *et al.*, 2021 (submitted).

Regarding future works we intend to further explore the passive attack issue, where the attacker gather information from the plant, we intend to explore known cryptographic schemes and adapt them to the framework of Discrete Event Systems to improve the security of the transmission of data in the lower levels of the industry. In this regard, we are also current studying a codification for I/O communication systems, where any variation in the I/O vector is codded as an event, so that more encryption schemes can be adapted to the framework of Discrete Event Systems.

Bibliography

ASHIBANI, Y., MAHMOUD, Q. H., 2017, "Cyber physical systems security: Analysis, challenges and solutions", *Computers & Security*, v. 68, pp. 81–97.

BARCELOS, R. J., BASILIO, J. C., 2018, "Enforcing current-state opacity through shuffle in event observations", *IFAC-PapersOnLine*, v. 51, n. 7, pp. 100–105.

BARCELOS, R. J., BASILIO, J. C., 2021, "Enforcing current-state opacity through shuffle and deletions of event observations", *Automatica*, v. 133, pp. 109836.

BENCSÁTH, B., PÉK, G., BUTTYÁN, L., FELEGYHAZI, M., 2012, "The cousins of stuxnet: Duqu, flame, and gauss", *Future Internet*, v. 4, n. 4, pp. 971–1003.

BIERE, A., HEULE, M., VAN MAAREN, H., 2009, *Handbook of satisfiability*, v. 185. IOS press.

CAO, L., JIANG, X., ZHAO, Y., WANG, S., YOU, D., XU, X., 2020, "A survey of network attacks on cyber-physical systems", *IEEE Access*, v. 8, pp. 44219–44227.

CARVALHO, L. K., MOREIRA, M. V., BASILIO, J. C., 2017, "Diagnosability of intermittent sensor faults in discrete event systems", *Automatica*, v. 79, pp. 315–325.

CARVALHO, L. K., WU, Y.-C., KWONG, R., LAFORTUNE, S., 2018, "Detection and mitigation of classes of attacks in supervisory control systems", *Automatica*, v. 97, pp. 121 – 133.

CASSANDRAS, C. G., 2016, "Smart cities as cyber-physical social systems", *Engineering*, v. 2, n. 2, pp. 156–158.

CASSANDRAS, C. G., LAFORTUNE, S., 2008, *Introduction to discrete event systems*. Secaucus, NJ, Springer.

COMER, D., 2009, *Computer networks and internets*. Upper Saddle River, NJ, USA, Pearson/Prentice Hall.

COUTO, C., COUTO, G. C. K., DA CUNHA, A. E. C., 2020, "Análise da segurança de redes em sistemas de automação e controle industriais: estudo de caso com a planta MecatrIME", *Anais da Sociedade Brasileira de Automática*, v. 2, n. 1.

DA XU, L., HE, W., LI, S., 2014, "Internet of things in industries: A survey", *IEEE Transactions on industrial informatics*, v. 10, n. 4, pp. 2233–2243.

DYER, J., DYER, M., XU, J., 2017, "Practical Homomorphic Encryption Over the Integers for Secure Computation in the Cloud". In: O'Neill, M. (Ed.), *Cryptography and Coding*, pp. 44–76.

FARWELL, J. P., ROHOZINSKI, R., 2011, "Stuxnet and the future of cyber war", *Survival*, v. 53, n. 1, pp. 23–40.

FRITZ, R., ZHANG, P., 2018, "Modeling and detection of cyber attacks on discrete event systems", *IFAC-PapersOnLine*, v. 51, n. 7, pp. 285–290.

FRITZ, R., FAUSER, M., ZHANG, P., 2019, "Controller encryption for discrete event systems". In: *2019 American Control Conference (ACC)*, pp. 5633–5638, PHILADELPHIA, CA, USA.

GAO, C., SEATZU, C., LI, Z., GIUA, A., 2019, "Multiple attacks detection on discrete event systems". In: *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pp. 2352–2357.

GILCHRIST, A., 2016, *Industry 4.0: the industrial internet of things*. 1st ed. Berkeley, Apress.

GOES, R. M., KANG, E., KWONG, R. H., LAFORTUNE, S., 2017, "Stealthy Deception Attacks for Cyber-Physical Systems". In: *Proceedings of the 56th IEEE Conference on Decision and Control*, pp. 4224–4230, Melbourne, Australia.

GOES, R. M., KANG, E., KWONG, R. H., LAFORTUNE, S., 2020, "Synthesis of sensor deception attacks at the supervisory layer of Cyber-Physical Systems", *Automatica*, v. 121, pp. 109172.

GOU, Q., YAN, L., LIU, Y., LI, Y., 2013, "Construction and strategies in IoT security system". In: *IEEE international conference on green computing and communications and IEEE internet of things and IEEE cyber, physical and social computing*, pp. 1129–1132.

GUBBI, J., BUYYA, R., MARUSIC, S., PALANISWAMI, M., 2013, "Internet of Things (IoT): A vision, architectural elements, and future directions", *Future generation computer systems*, v. 29, n. 7, pp. 1645–1660.

GUO, Y., JIANG, X., GUO, C., WANG, S., KAROUI, O., 2020, "Overview of Opacity in Discrete Event Systems", *IEEE Access*, v. 8, pp. 48731–48741.

HE, H., MAPLE, C., WATSON, T., TIWARI, A., MEHNEN, J., JIN, Y., GABRYS, B., 2016, "The security challenges in the IoT enabled cyber-physical systems and opportunities for evolutionary computing & other computational intelligence". In: *2016 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1015–1021.

HOPCROFT, J. E., MOTWANI, R., ULLMAN, J. D., 2006, *Introduction to automata theory, languages, and computation*. Boston, Addison Wesley.

HUANG, X., DONG, J., 2018, "Reliable control policy of cyber-physical systems against a class of frequency-constrained sensor and actuator attacks", *IEEE transactions on cybernetics*, v. 48, n. 12, pp. 3432–3439.

HUMAYED, A., LIN, J., LI, F., LUO, B., 2017, "Cyber-physical systems security?A survey", *IEEE Internet of Things Journal*, v. 4, n. 6, pp. 1802–1831.

IGURE, V. M., LAUGHTER, S. A., WILLIAMS, R. D., 2006, "Security issues in SCADA networks", *computers & security*, v. 25, n. 7, pp. 498–506.

JADOON, A. K., LI, J., WANG, L., 2021, "Physical layer authentication for automotive cyber physical systems based on modified HB protocol", *Frontiers of Computer Science*, v. 15, n. 3, pp. 1–8.

JI, Y., 2019, *From Security Enforcement to Supervisory Control in Discrete Event Systems: Qualitative and Quantitative Analyses*. Tese de Doutorado.

JIRKOVSKY, V., OBITKO, M., MARIK, V., 2017, "Understanding Data Heterogeneity in the Context of Cyber-Physical Systems Integration", *IEEE Transactions on Industrial Informatics*, v. 13, n. 2, pp. 660–667.

KNUTH, D. E., 1975, "Estimating the efficiency of backtrack programs", *Mathematics of computation*, v. 29, n. 129, pp. 122–136.

KUMAR, V., 1992, "Algorithms for constraint-satisfaction problems: A survey", *AI magazine*, v. 13, n. 1, pp. 32–44.

KUROSE, J. F., ROSS, K. W., 2011, *Computer networking: a top-down approach*. Addison Wesley.

LAWSON, M. V., 2003, *Finite automata*. Florida, CRC Press.

LEE, E. A., 2008, "Cyber physical systems: Design challenges". In: *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pp. 363–369.

LEHMER, D., OTHERS, 1932, "On Euler's totient function", *Bulletin of the American Mathematical Society*, v. 38, n. 10, pp. 745–751.

LIMA, P. M., 2017, *Security against network attacks in supervisory control systems*. Tese de Mestrado, Federal University of Rio de Janeiro.

LIMA, P. M., ALVES, M. V. S., CARVALHO, L. K., MOREIRA, M. V., 2021, "Security of Cyber-Physical Systems: Design of a Security Supervisor to Thwart Attacks", *IEEE Transactions on Automation Science and Engineering*. doi: 10.1109/TASE.2021.3076697.

LIMA, P. M., CARVALHO, L. K., MOREIRA, M. V., 2021 (submitted), "Ensuring confidentiality of cyber-physical systems using event-based cryptography", *Discrete Event Dynamic Systems*.

LIMA, P. M., ALVES, M. V., CARVALHO, L. K., MOREIRA, M. V., 2017, "Security Against Network Attacks in Supervisory Control Systems", *IFAC-PapersOnLine*, v. 50, n. 1 (jul), pp. 12333–12338.

LIMA, P. M., CARVALHO, L. K., MOREIRA, M. V., 2018, "Detectable and Undetectable Network Attack Security of Cyber-physical Systems", *IFAC-PapersOnLine*, v. 51, n. 7, pp. 179–185.

LIMA, P. M., ALVES, M. V. S., CARVALHO, L. K., MOREIRA, M. V., 2019, "Security Against Communication Network Attacks of Cyber-Physical Systems", *Journal of Control, Automation and Electrical Systems*, v. 30, n. 1 (feb), pp. 125–135.

LIMA, P. M., CARVALHO, L. K., MOREIRA, M. V., 2020, "Confidentiality of Cyber-Physical Systems Using Event-Based Cryptography". In: *21st IFAC World Congress 2020*, pp. 1761–1766, Berlin, Germany.

LIN, F., WONHAM, W., 1988, "On observability of discrete-event systems", *Information Sciences*, v. 44, n. 3, pp. 173 – 198.

LIN, F., 2011, "Opacity of discrete event systems and its applications", *Automatica*, v. 47, n. 3, pp. 496–503.

MASOPUST, T., 2018, "Complexity of Infimal Observable Superlanguages", *IEEE Transactions on Automatic Control*, v. 63, n. 1, pp. 249–254.

MAZARÉ, L., 2004, "Using unification for opacity properties", *Proceedings of the 4th IFIP WG1*, v. 7, pp. 165–176.

MO, Y., SINOPOLI, B., 2010, "False data injection attacks in control systems". In: *Preprints of the 1st workshop on Secure Control Systems*, pp. 1–6, Stockholm, Sweden.

MO, Y., SINOPOLI, B., 2012, "Integrity Attacks on Cyber-Physical Systems". In: *Proceedings of the 1st International Conference on High Confidence Networked Systems*, pp. 47–54.

MOHAMMAD, S. M., LAKSHMISRI, S., 2018, "Security automation in Information technology", *International Journal of Creative Research Thoughts (IJCRT)*, v. 6.

MOREIRA, M. V., JESUS, T. C., BASILIO, J. C., 2011, "Polynomial time verification of decentralized diagnosability of discrete event systems", *IEEE Transactions on Automatic Control*, v. 56, n. 7, pp. 1679–1684.

PAOLI, A., LAFORTUNE, S., 2005, "Safe diagnosability for fault-tolerant supervision of discrete-event systems", *Automatica*, v. 41, n. 8, pp. 1335–1347.

PFLEEGER, C. P., PFLEEGER, S. L., 2002, *Security in computing*. Prentice Hall Professional Technical Reference.

SABOORI, A., HADJICOSTIS, C. N., 2007, "Notions of security and opacity in discrete event systems". In: *2007 46th IEEE Conference on Decision and Control*, pp. 5056–5061.

SABOORI, A., HADJICOSTIS, C. N., 2008, "Verification of initial-state opacity in security applications of DES". In: *2008 9th International Workshop on Discrete Event Systems*, pp. 328–333.

SABOORI, A., HADJICOSTIS, C. N., 2011a, "Verification of infinite-step opacity and complexity considerations", *IEEE Transactions on Automatic Control*, v. 57, n. 5, pp. 1265–1269.

SABOORI, A., HADJICOSTIS, C. N., 2011b, "Verification of K-step opacity and analysis of its complexity", *IEEE Transactions on Automation Science and Engineering*, v. 8, n. 3, pp. 549–559.

SABOORI, A., HADJICOSTIS, C. N., 2013, "Current-state opacity formulations in probabilistic finite automata", *IEEE Transactions on automatic control*, v. 59, n. 1, pp. 120–133.

SAMPATH, M., SENGUPTA, R., LAFORTUNE, S., SINNAMOHIDEEN, K., TENEKETZIS, D., 1995, "Diagnosability of discrete-event systems", *IEEE Transactions on automatic control*, v. 40, n. 9, pp. 1555–1575.

SHIREY, R., 2000, "RFC 2828: Internet security glossary", *The Internet Society*, v. 13.

SINGH, S., SINGH, N., 2015, "Internet of Things (IoT): Security challenges, business opportunities & reference architecture for E-commerce". In: *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, pp. 1577–1581.

SKIENA, S. S., 2020, *The algorithm design manual*. Springer International Publishing.

STALLINGS, W., 2006, *Cryptography and network security, 4/E*. Pearson Education India.

SU, R., 2018, "Supervisor synthesis to thwart cyber attack with bounded sensor reading alterations", *Automatica*, v. 94, pp. 35–44.

THORSLEY, D., TENEKETZIS, D., 2006, "Intrusion detection in controlled discrete event systems". In: *Proceedings of the 45th IEEE Conference on Decision and Control*, pp. 6047–6054, San Diego, CA, USA.

TONG, Y., LI, Z., SEATZU, C., GIUA, A., 2018, "Current-state opacity enforcement in discrete event systems under incomparable observations", *Discrete Event Dynamic Systems: Theory and Applications*, v. 28, n. 2, pp. 161–182.

WANG, S., WAN, J., ZHANG, D., LI, D., ZHANG, C., 2016, "Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination", *Computer Networks*, v. 101, pp. 158–168.

WU, B., CHEN, J., WU, J., CARDEI, M., 2007, "A survey of attacks and countermeasures in mobile ad hoc networks". In: *Wireless network security*, pp. 103–135.

WU, Y.-C., LAFORTUNE, S., 2013, "Comparative analysis of related notions of opacity in centralized and coordinated architectures", *Discrete Event Dynamic Systems*, v. 23, n. 3, pp. 307–339.

WU, Y.-C., LAFORTUNE, S., 2014, "Synthesis of insertion functions for enforcement of opacity security properties", *Automatica*, v. 50, n. 5, pp. 1336–1348.

WU, Y.-C., RAMAN, V., RAWLINGS, B. C., LAFORTUNE, S., SESHIA, S. A., 2018, "Synthesis of obfuscation policies to ensure privacy and utility", *Journal of Automated Reasoning*, v. 60, n. 1, pp. 107–131.

X-800, 1991, "Security Architecture for Open Systems Interconnection for CCITT Applications." .

YIN, X., LAFORTUNE, S., 2016, "Synthesis of Maximally Permissive Supervisors for Partially-Observed Discrete-Event Systems", *IEEE Transactions on Automatic Control*, v. 61, n. 5, pp. 1239–1254.

YIN, X., LAFORTUNE, S., 2017, "Synthesis of maximally-permissive supervisors for the range control problem", *IEEE Transactions on Automatic Control*, v. 62, n. 8, pp. 3914–3929.

YIN, X., LI, Z., WANG, W., LI, S., 2019, "Infinite-step opacity and K-step opacity of stochastic discrete-event systems", *Automatica*, v. 99, pp. 266–274.

ZHANG, Q., LI, Z., SEATZU, C., GIUA, A., 2018, "Stealthy Attacks for Partially-Observed Discrete Event Systems". In: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, v. 1, pp. 1161–1164.