COPPE
Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia
UFRJ

SECURITY STRATEGIES TO ENFORCE CURRENT-STATE OPACITY OF DISCRETE-EVENT SYSTEMS USING SEGMENTED NETWORKS

Lucas Nelson Ribeiro Reis

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
Março de 2026

SECURITY STRATEGIES TO ENFORCE CURRENT-STATE OPACITY OF DISCRETE-EVENT SYSTEMS USING SEGMENTED NETWORKS

Lucas Nelson Ribeiro Reis

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. Lilian Kawakami Carvalho
Prof. Raphael Julio Barcelos
Prof. Max Hering de Queiroz
Prof. Marcelo Teixeira
Prof. Antonio Eduardo Carrilho da Cunha

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2026

Nelson Ribeiro Reis, Lucas

Security strategies to enforce Current-State Opacity of Discrete-Event Systems using segmented networks/Lucas Nelson Ribeiro Reis. – Rio de Janeiro: UFRJ/COPPE, 2026.

XIV, 80 p.: il.; 29, 7cm.

Orientadores: Marcos Vicente de Brito Moreira
Lilian Kawakami Carvalho

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2026.

Referências Bibliográficas: p. 73 – 80.

1. Cyber-security. 2. Current-state opacity. 3. Cyber-Physical Systems. 4. Discrete-Event Systems. 5. Automata. I. Vicente de Brito Moreira, Marcos *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Agradecimentos

Agradeço primeiramente a Deus por toda força e saúde por me pertmitir chegar aqui.

Agradeço a Mariana Domingues e a meus filhos Emanuel Reis, Elisa Reis e Laura Reis por todo apoio, incentivo e compreensão pelos muitos momentos de ausência necessários para a conclusão desse trabalho e por todo incentivo para que fosse possível.

Agradeço aos meus pais Mara Rocha e Luiz Reis por todo esforço, dedicação e incentivo na minha educação para que pudesse alçar vôos como esse.

Agradeço aos meus avós por sempre me fazerem acreditar que seria capaz.

Agradeço à Marinha do Brasil por me permitir me dedicar a este trabalho tão enriquecedor.

Agradeço a todos que passaram pelo Laboratório de Controle e Automação. Cada discussão técnica que tivemos ajudou na conclusão desse trabalho.

Agradeço aos meus orientadores Marcos Moreira e Lilian Carvalho por toda orientação e ensinamentos passados ao longo do trabalho.

Agradeço também à COPPE/UFRJ, seu corpo docente e administração, e a todos aqueles que, de alguma forma, contribuíram para que eu chegasse até aqui.

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

ESTRATÉGIAS DE SEGURANÇA PARA GARANTIR A OPACIDADE DO ESTADO ATUAL EM SISTEMAS DE EVENTOS DISCRETOS USANDO REDES SEGMENTADAS

Lucas Nelson Ribeiro Reis

Março/2026

Orientadores: Marcos Vicente de Brito Moreira
Lilian Kawakami Carvalho

Programa: Engenharia Elétrica

Os Sistemas Ciberfísicos (SCF) podem ser vulneráveis a ataques passivo, e uma das estratégias propostas na literatura para garantir a segurança dos dados em SCF abstraídos como Sistemas de Eventos Discretos em Rede (SEDR) é o forçamento de opacidade. Neste trabalho, consideramos uma nova noção de utilidade, chamada utilidade de estado atual (UEA), na qual o receptor válido deve sempre ter certeza do estado atual. Para atender esta noção de utilidade, propomos duas abordagens baseadas na segmentação de rede, que é um metodo de segurança que divide a rede comunicação em sub-redes isoladas para evitar que o atacante tenha acesso a todos os dados comunicados na rede. Na primeira abordagem propomos uma segmentação de rede particionando o conjunto de eventos de forma que garanta a opacidade de estado atual (OEA) em cada uma das sub-redes. O número de sub-rede impacta diretamente no custo da implementacao, portanto propomos uma solução ótima para encontrar a partição com menor número de sub-redes. A solução ótima tem uma alta complexidade computacional, então, para implementar a segmentação de rede garantindo OEA mitigando o custo computacional, apresentamos uma solução sub-ótima para o problema. Na segunda abordagem, propomos uma estratégia para garantir a OEA e a UEA para o receptor, baseada em criptografia e controle de roteamento, denominado roteamento de eventos multicaminhos, no qual os eventos são redistribuídos e comunicados ao receptor por meio de diferentes sub-redes. Para solucinoar a segunda abordagem, apresentamos um método com condição necessária e suficiente para calcular um autômato que representa todas as funções de trocas de eventos que garantem OEA e UEA.

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

SECURITY STRATEGIES TO ENFORCE CURRENT-STATE OPACITY OF DISCRETE-EVENT SYSTEMS USING SEGMENTED NETWORKS

Lucas Nelson Ribeiro Reis

March/2026

Advisors: Marcos Vicente de Brito Moreira
Lilian Kawakami Carvalho

Department: Electrical Engineering

Cyber-Physical Systems (CPS) can be vulnerable to passive attacks, and one of the strategies proposed in the literature to ensure data security in CPS abstracted as Network Discrete Event Systems (NDES) is opacity enforcement. In this work, we consider a new notion of utility, called current-state utility (CSU), in which the intended receiver must always be certain of the current state of the system. To address this notion of utility, we propose two approaches based on network segmentation, which is a security method that divides the communication network into isolated subnets to prevent the attacker from accessing all the data communicated on the network. In the first approach, we propose a network segmentation by partitioning the event set in a way that guarantees current-state opacity (CSO) in each of the subnets. The number of subnets directly impacts the implementation cost; therefore, we propose an optimal solution to find the partition with the fewest subnets. The optimal solution has high computational complexity, so to implement network segmentation guaranteeing CSO while mitigating computational cost, we present a suboptimal solution to the problem. In the second approach, we propose a strategy to guarantee CSO and CSU for the receiver, based on cryptography and routing control, called multipath event routing, in which events are replaced and communicated to the receiver through different subnets. To solve the second approach, we present a method with necessary and sufficient conditions to compute an automaton that represents all event replacement functions that guarantee CSO and CSU.

Contents

List of Figures

List of Tables

List of Symbols

$Ac(G)$	Accessible part of G, p. 16
G	Automaton, p. 15
$L(G)$	Generated language of automaton G, p. 15
$Obs(G, \Sigma_o)$	Observer automaton of G considering Σ_o., p. 18
$P^{\Sigma}_{\Sigma_r}$	Projection operation defined as $P^{\Sigma}_{\Sigma_r} : \Sigma^* \rightarrow \Sigma_r^*$, p. 14
$Pr(L)$	Prefix-closure operation on language L,, p. 13
$UR(x)$	Unobservable reach of state x, p. 18
X	Set of states, p. 15
Σ	Set of Events, p. 12
ε	Empty trace, p. 12
f	Transition function, p. 15
$f_{ER} : \Sigma^* \rightarrow \Sigma_\rho^*$	Event replacement function, p. 43
x_0	Initial state, p. 15
$\mathcal{G}$	Nondeterministic Automaton, p. 16

List of Abbreviations

CDR	complete sets of distinct representatives, p. 20
CPS	Cyber-Physical Systems, p. 1
CSO-SO	Current-state opacity based on state outputs, p. 5
CSO	Current-state opacity, p. 3
CSU	Current-state utility, p. 8
DES	Discrete event systems, p. 1
DoS	Denial-of-Service, p. 2
IFO	Initial-and-final-state opacity, p. 3
ISO	Initial-state opacity, p. 3
LBO	Language-based opacity, p. 3
MSO	Maximal subsets for opacity, p. 8
NDES	Networked Discrete event systems, p. 1
NRP state	Non-RP state, p. 62
OSN-CSO	Optimal segmented network current-state opacity, p. 8
PRP state	Possible RP state, p. 62
PS	Partition subset, p. 8
RP state	Reversible-policy state, p. 61
SBO	State-based opacity, p. 3
SCADA	Supervisory control and data acquisition, p. 1
SN-CSO	Segmented network current-state opacity, p. 8

Chapter 1

Introduction

Discrete event systems (DES) are systems whose evolution is driven by the occurrence of events [1, 2], and whose set of states is discrete. The act of pushing a button by an operator, starting or ending a task, or the changing of a sensor state are examples of events. This kind of systems can be observed in several applications, such as robotic systems, operational systems, manufacturing systems, and data management.

It is important to remark that, in DES, events are defined as instantaneous occurrences, that can change the system state. This avoids the adoption of mathematical formalism based on differential or difference equations to represent these systems. Thus, it is necessary to adopt a mathematical formalism capable of dealing with the characteristics of a DES. In order to model DES, the most common formalism are Petri nets and automata [1–5]. In this work, we use automata to model DES. Several problems have been addressed in the literature considering systems abstracted as DES, such as, supervisory control [6–9], state estimation [10–12], fault diagnosis [13–20], and cyber-security [21–31].

A class of systems that, in some level of abstraction, can be modeled as a networked Discrete-Event System (NDES), are the Cyber-Physical Systems (CPS). CPS are the key element of Industry 4.0, where computational components are connected to the physical plant by using communication networks, which may be vulnerable to attacks [31–35]. In 2010, a cyberattack was discovered that targeted the supervisory control and data acquisition (SCADA) systems and is believed to be responsible for causing damage to the Iranian nuclear program. This attack was performed by the computer worm Stuxnet. More recently, in 2022, hackers attacked Ukrainian energy facilities, trying to shutdown Ukrainian electrical substations. In 2024, several pagers and walkie-talkies intended for use by Hezbollah exploded simultaneously in two separate events, in Siria and Lebanon.

Cyberattacks can be classified as active or passive. The former are those in which the attacker is capable of altering the system behavior, with the aim to vio-

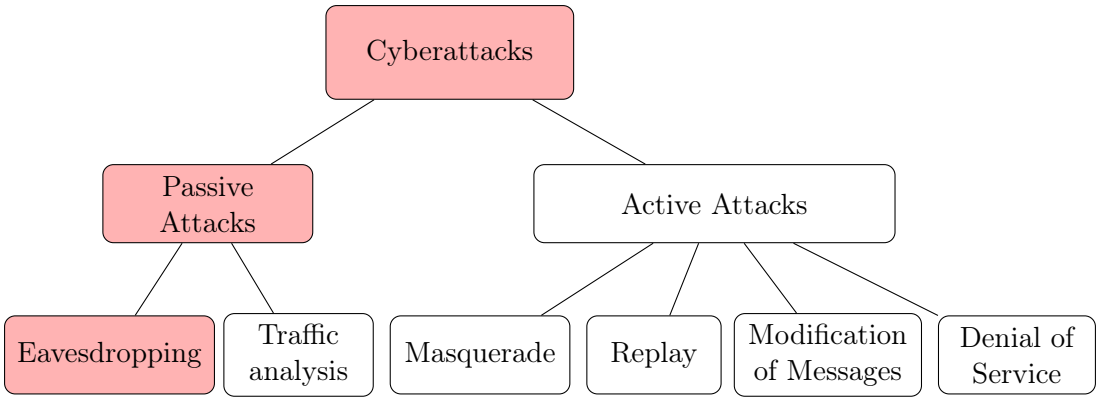


Figure 1.1: Summary of the types of cyberattacks

late data integrity and/or service availability [22–24], while the latter are those in which the attacker does not alter the behavior of the system but aims to violate data confidentiality [25–27]. Figure 1.1 summarizes the types of cyberattacks [36]. Eavesdropping attacks are those in which the attacker seeks to gather information by observing the transmitted message and, with that observation, learn the content of the transmitted message and discover the secret of the system. In traffic analysis, the attacker wants to obtain relevant information from the transmitted message but cannot comprehend the content of the message, thus, to obtain relevant information, the attacker uses the pattern of the transmitted information and gathers relevant information from that pattern. Replay attacks require first that the intruder captures data from a transmission for a later retransmission to produce an unauthorized effect. The denial of service is an attack that prevents or inhibits the normal use or management of communications facilities. Another form of denial of service is the disruption of an entire network by overloading or disabling the network. The data modification attack means that part of a legitimate message has been altered. In a masquerade attack an entity pretends to be a different entity, and usually includes other form of active attack.

In this work, we focus on eavesdropping attacks. Even though eavesdropping attacks do not alter system behavior, the information they compromise can be used to execute active attacks, such as a Denial-of-Service (DoS) attack. This is particularly relevant for industrial networked automation systems, where ensuring high availability is a primary requirement. In response, the adoption of cybersecurity standards is gradually being enforced by legal directives on field and device-level networks, such as the NIS2 European Union's framework for cybersecurity [37].

According to the NIS2 directive, organizations should adopt a wide range of basic cyber hygiene practices, evaluate their cybersecurity capabilities and, where appropriate, pursue the integration of cybersecurity enhancing technologies. Among the security measures an organization should adopt is the physical or logical segmen-

tation of the network, resulting in a set of isolated and independent sub-networks, or subnets. Segmenting networks limits lateral movements of attackers, protects critical assets, and reduces the overall attack surface of a given system.

Network segmentation of controller-to-sensor communication into isolated subnets creates the scenario where, if an attacker invades one of the subnets to observe the communication traffic of the system, they will not have access to the communication traffic and information on the remaining subnets. This architectural isolation not only mitigates the risk of common-cause failures by limiting an intruder's observational capabilities to a single compromised subnet, but also provides a structural foundation for formal security analysis.

In the context of NDES, different approaches are used to deal with passive attacks, such as opacity ([38, 39] and references therein). Different notions of opacity have been presented in the literature, that can by classified in two families, language-based opacity (LBO) [40–42] and state-based opacity (SBO) [43–48].

Regarding LBO, the system can be classified as strong opaque, weak opaque, or not opaque [42]. A system is said to be strong opaque, or just opaque, if for each secret sequence, there exists at least one non-secret sequence with the same observation. On the other hand, a system is said to be weak opaque if, for some secret sequence, there exists another non-secret sequence with the same observation. In addition, if a system is not strong opaque nor weak opaque, then the system is said to be not opaque.

The SBO is related to the capacity of the attacker to infer if the system is in a secret state [43, 44]. Regarding SBO, different notions of opacity have been defined: current-state opacity (CSO) [43–45]; initial-state opacity (ISO) [46]; initial-and-final-state opacity (IFO) [47]; K-step opacity [44]; and infinity-step opacity [48].

A system is said to be current-state opaque with respect to a set of secret states, if the intruder is never capable of inferring a secret state of the system based on the observation of events communicated through the attacked communication channels. Initial-state opacity is related to the initial state of the system. It is considered that the system has a set of initial states that are partitioned in secret initial states and non-secret initial states. Then, after the system executes a sequence of events, the system is said initial-state opaque if it is not possible to infer that the initial state was a secret state or a non-secret state considering the observation of the events. An extension of CSO and ISO is the initial-and-final-state opacity. Regarding IFO, the secret is defined over a pair of states, and the system is said to be initial-and-final-state opaque if the intruder is not certain if the initial state of the system is a secret state and the final state of the system is also a secret state. K-step opacity states that the secret is safe from the last K steps executed by the system until the current moment, *i.e.*, the intruder is not certain if the system was in a secret state

in the last k steps. When K is arbitrarily long, the system is said to be infinity-step opaque.

In SABOORI e HADJICOSTIS [46], it is shown that LBO can be transformed into ISO in order to apply supervisory control techniques. In CASSEZ *et al.* [49, 50], transformations from LBO to CSO are presented. Finally, in WU e LAFORTUNE [47], the relations between the notions of opacity are discussed and a map between LBO, CSO, ISO and IFO is presented.

In this work, since we intend to protect the secret behavior modeled by secret states, we consider the enforcement of current-state opacity.

Several methods to enforce current-sate opacity have been proposed in the literature, such as: *(i)* using supervisory control to restrict the system language [51–53]; or *(ii)* enforcing opacity by changing observations of events from the attackers' perspective [26, 54, 55]. In this work, we focus on enforcing opacity by changing the observations of the events.

A method to enforce opacity by using insertion functions is proposed in WU e LAFORTUNE [56]. In this approach, a monitoring interface, placed on the system output, receives each observation of the event executed by the plant and, if necessary, inserts additional fake events prior to the observed event to mislead the attacker. In this method, when successfully implemented, the attacker never infers the secret behavior since, the inserted events cannot be distinguished from the original ones by the attacker. In addition, after observing the insertion of events, the expected behavior for the attacker must be the same as the expected behavior of the system considering the plant event. It is assumed that the attacker knows the system model but is not aware of the use of an insertion function. In WU e LAFORTUNE [57], the problem of quantifying the insertion of events and synthesizing optimal insertion functions is addressed. In JI *et al.* [58], insertion functions are classified as privately and publicly known. In the former, the attacker does not know that a defense mechanism is being used [56, 57], while in the latter the attacker is aware that a defense mechanism is being used.

In KEROGLOU e LAFORTUNE [59], the embedded insertion function is introduced, where the exact state that the system reaches after the execution of this event is important to obtain the possible insertion of events. In JI *et al.* [54], it is defined an energy function in which the energy is related to the insertion of events. The energy can be consumed or increased depending on the system behavior and the problem of using insertion functions to enforce opacity in systems optimizing the energy function is addressed. In KEROGLOU *et al.* [60], instead of inserting and reporting the events just after its occurrence, the defense mechanism stores a number of events, modify the sequence by inserting events, and then communicate the modified sequence. In LI *et al.* [61], the extended insertion function is proposed,

where, instead of inserting events only prior to the observed events, the extended insertion function is capable of inserting fictitious events before and after the observed event. In LI *et al.* [62], the extended insertion function is revisited, and the events are inserted according to constraints that allow events from specified sets of events to be inserted before and after the occurrence of an observed event. In addition, a method to determine feasibility of an extended insertion strategy via an appropriate verifier construction is provided, and, if possible, a strategy to make the system current-state opaque under inserted language constraints is presented.

Edit functions are used in WU *et al.* [63] and JI e LAFORTUNE [64] to enforce opacity. Edit functions can insert, delete, and replace event observations in order to protect the secret from the attacker. As in the insertion function, the replaced events and the inserted events cannot be distinguished from the original ones by the attacker. Additionally, in JI e LAFORTUNE [64], the public-private enforcing is discussed by using edit functions. In JI *et al.* [65], the edit function is extended to allow nondeterministic editions, *i.e.*, the edit function outcome is randomly chosen from a pre-calculated set, and the intruder does not know the result a priori.

Different approaches of enforcing opacity by using edit functions continue to be studied. In WINTENBERG *et al.* [66], the enforcement of K-step opacity using edit functions is addressed. In DUAN *et al.* [67], an edit mechanism to enforce opacity is presented, where the observable events for the defense mechanism are different from the observable events for the attacker.

In MAYER *et al.* [68], different from the other methods proposed in the literature, where events are transmitted in the network, the transmitted information is the status of system components, such as sensors, actuators, and local controller memory variables. In this case, the system is modeled as a finite state transducer, where each state of the model is associated with an output vector formed of signals associated with the system components. Since the attacker observes state outputs of a finite state transducer, current-state opacity based on state outputs (CSO-SO), is introduced. A method to verify this property is also proposed.

A disadvantage of using opacity enforcers is that, in some cases, useful information for the receiver can be obfuscated. Then, several works have been proposed with a view to taking into account the security of the transmitted information and, at the same time, its utility for the intended receiver [63, 69–74].

In WU *et al.* [63], the concepts of utility and privacy are presented. The notion of utility used in WU *et al.* [63] is associated with geolocation, where the reported position must be within a distance from the actual position to be considered useful. In order to guarantee privacy, the sequences of events generated by the system are obfuscated such that the secret positions are never reported by the system, and, consequently, never inferred by the intruder. In CARDOSO *et al.* [69], the privacy

and utility of information are also addressed, and an obfuscation policy that can report a secret state if the system is sufficiently far from the secret state, is proposed. Thus, there is a trade-off between the notions of privacy and utility as defined in WU *et al.* [63] and CARDOSO *et al.* [69].

In WINTENBERG *et al.* [70], it is proposed a strategy for obfuscation that includes an inference interface to allow the intended receiver to interpret the obfuscated information. The problem of finding an obfuscation policy that enforces privacy and the capability of the intended receiver to infer when a secret state is reached is formulated. In this approach, the intended receiver is not capable of recovering the original message transmitted by the sender, but only infer secret states.

In BARCELOS e BASILIO [26], it is proposed a strategy to enforce opacity based on shuffling the order of observations of the events, and also the deletion of the observation of the events to make the attacker always believe that the system is in a non-secret state. In addition, it is presented a protocol to mitigate the negative effect of opacity enforcement on the capability of a legitimate recipient to accurately estimate the current state of the system. In BARCELOS e BASILIO [71], the strategy presented in BARCELOS e BASILIO [26] to enforce opacity is updated to not just mitigate the negative effect of opacity enforcers, but to consider the utility of the information. The notion of utility defined in BARCELOS e BASILIO [71], called utility-ensured, is such that a system is said to be utility-ensured if the intended receiver is certain of the current state of the system when the system is in a secret state.

Recently, in LIN *et al.* [72] and DUAN *et al.* [73], the utility of the information is associated with fault diagnosis. In LIN *et al.* [72], an attack detection strategy is proposed where it is considered that the intended receiver is a fault diagnoser. Different from WU *et al.* [63], CARDOSO *et al.* [69], BARCELOS e BASILIO [71], which consider passive attacks, in LIN *et al.* [72], active sensor attacks are considered, and a method is proposed to verify if the diagnosability of the system language under active attacks is maintained. More recently, in DUAN *et al.* [73], the intended receiver is a fault diagnoser, and the intruder is capable of eavesdropping on all observable events of the system. In DUAN *et al.* [73], the information that must be kept secret from the intruder is the occurrence of specific events. Thus, an obfuscation mechanism is proposed to simultaneously conceal all secret event occurrences and release all fault occurrences.

In MAYER *et al.* [75], a new notion of utility, called state-based transparency, is presented. In this case, the useful information is whether a given property of interest, modeled by a set of states associated with the property of interest, is valid. A system is state-based transparent if the intended receiver is capable of inferring

that the system has reached a state where the property is valid before the system reaches a state in which the property does not hold anymore.

It is important to remark that the methods proposed in WU *et al.* [63], CARDOSO *et al.* [69], WINTENBERG *et al.* [70], BARCELOS e BASILIO [71], LIN *et al.* [72], DUAN *et al.* [73], MAYER *et al.* [75] do not guarantee that the intended receiver completely recovers the transmitted message, while keeping the secret information hidden from the attacker. An alternative to ensure that the intended receiver is able to recover the original message is proposed in LIU *et al.* [74], where it is assumed that the receiver is capable of distinguishing inserted events and replaced events from the original ones, while and the attacker is not able. The strategy adopted in LIU *et al.* [74] to allow the receiver to distinguish original sequences from obfuscated sequences is steganography, which is the technique of hiding data within a non-secret message. The main drawback with this strategy is that steganography is a weak security mechanism since once the strategy is discovered, the secret information can be easily obtained [36].

An alternative to ensure that the intended receiver is capable of recovering the original message is to use cryptography [25, 28, 29, 36, 76]. In cryptographic systems, a single key or two keys are used to cipher and decipher the message, in such a way that only the intended receiver, who possesses the key, can recover the original message. In this case, the attacker is aware of the use of a defense mechanism, and may try to perform cryptanalysis to discover the secret message or the key. In LIMA *et al.* [25], the property of confidentiality of DES with respect to an encryption function and a secret language is proposed, which is associated with the fact that only the sender and the intended receiver are able to understand the transmitted secret message. In LIMA *et al.* [28] a cryptographic scheme, called event-based cryptography, is presented. In this scheme, the events are defined as changes in the binary signals transmitted in the communication channels, and the cryptography modifies the events in the application layer of the network, keeping the transmitted data structure. In OLIVEIRA *et al.* [29], cryptography is used to protect the system from active attacks in actuators. To ensure the integrity of the system, the controllable events that are disabled by the supervisor are encrypted to avoid the opportunity for the attacker to trigger events that would lead the system to unsafe states.

In this work, we consider that the secret information is modeled by secret states of the plant, and the attacker tries to infer that a secret state has been reached by observing only part of the events. We also consider that the attacker knows the plant model represented by an automaton and the communication architecture used. The intended receiver considered in this work is any entity that receives data from the plant through a communication network, such as a SCADA system or a

diagnoser. We also consider that the useful information for the intended receiver is the knowledge of the system current state after the execution of any sequence of events of the plant. Thus, data confidentiality must be guaranteed, *i.e.*, only the sender and intended receiver must be able to understand the transmitted message. In this work we consider a new notion of utility, where the intended receiver is certain of the current state of the system based on the observation of the communicated events, called current-state utility (CSU). In order to guarantee data confidentiality, we propose two approaches based on network segmentation. First, we propose to apply a network segmentation with the aim of ensuring CSO for each of the subnets in isolation [77]. After segmenting the network, or given a segmented network, there may exist vulnerable subnets that, if the attacker invades these subnets, they could discover the system secret. In these cases, we propose a strategy to ensure CSO and CSU for the receiver based on cryptography and routing control, called multipath event routing, in which the events are replaced and communicated to the receiver through different subnets [27].

The method to implement the first approach consists of partitioning the set of observable events such that the system is CSO with respect to the set of secret states and the observation of the events of each subset of the partition, called partition subset (PS), in isolation. Then, each isolated subnet is used to transmit the events belonging to each one of the PS, *i.e.*, the number of isolated subnets is equal to the number of PS. Thus, if the attacker invades only one of the subnets, they cannot discover the system secret. We call the problem of finding a partition of the observable event set, such that the system is CSO with respect to each one of the PS, the segmented network current-state opacity (SN-CSO) problem. Since the cost for implementing a segmented network increases with the number of isolated subnets, it is desirable to find the minimum number of PS such that the system is CSO with respect to each PS. The problem of finding the partition of the observable event set with the smallest possible number of PS such that the system is CSO with respect to each PS is called the Optimal SN-CSO (OSN-CSO) problem.

The OSN-CSO problem can be solved in two steps: (*i*) find all maximal subsets of observable events such that the system is CSO with respect to the set of secret states and the observation of events in these sets, called maximal subsets for opacity (MSO); (*ii*) obtain a partition of the set of observable events with the minimum number of PS from all the MSO. We show that Step (*i*) is, in the worst-case, exponential in the number of system events and exponential in the number of system states. Thus, we present sufficient conditions to mitigate the computational complexity of finding the MSO. Step (*ii*) leads to the classical set covering problem [78]. Since the set covering problem is NP-hard, we present a greedy algorithm that leads to a sub-optimal solution to the SN-CSO problem.

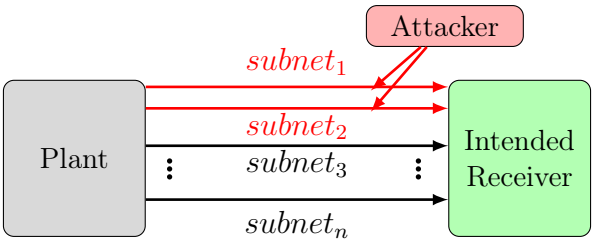


Figure 1.2: Default communication between the system and diagnoser, where subnets 1 and 2 are vulnerable to attacks.

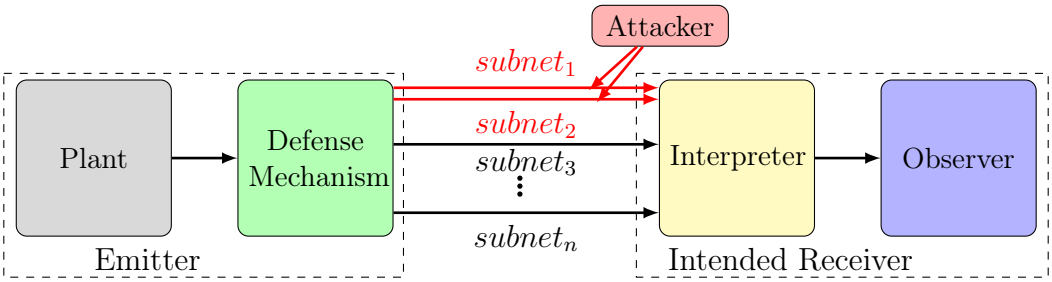


Figure 1.3: Proposed defense architecture using the multipath event routing considering subnets 1 and 2 vulnerables.

To implement the second approach, the strategy called multipath event routing, we propose to take advantage of the segmented network and we consider that part of the subnets is vulnerable to eavesdropping attaks. Figure 1.2 represents the communication architecture of a system with a segmented network in which the attacker invades all vulnerable subnets. In this case, if it is possible to infer the secret by observing the information transmitted through the attacked subnets, the secret is exposed. The multipath event routing determines through which subnet an event must be reported to the receiver after its occurrence, and which event will be used as replacement for transmission in this subnet. In addition, the intruder may not suspect that a security mechanism is being deployed. To do so, the events communicated through the attacked subnets must lie within the behavior expected by the intruder and the attacked subnets cannot be silenced for more than a given number of event transmissions by the routing mechanism. The architecture of the system with the defense mechanism is presented in Figure 1.3.

As in WU *et al.* [63], CARDOSO *et al.* [69], LIU *et al.* [74], to obtain obfuscation functions for the multipath event routing, we adopt a game-like structure represented by an automaton to model all obfuscation functions that ensure CSO, and then, we compute from this automaton an obfuscation policy that also enforces CSU. Different from WU *et al.* [63], CARDOSO *et al.* [69], LIU *et al.* [74], the game-like structure automaton is obtained considering the proposed segmented network communication

architecture, which allows us to compute, whenever possible, an obfuscation policy that enforces CSU without using other defense mechanisms, such as steganography. In addition, since we adopt the enforcement of opacity to protect the secret, the attacker must not raise suspicious that a defense mechanism is being used by the lack of communication of events. Thus, the trivial solution of enforcing opacity by erasing all transmissions on the attacked subnets is avoided. To do so, the defense mechanism considers that the attacked subnets cannot be silenced for more than a given number of event transmissions. Moreover, differently from WU *et al.* [63], LIU *et al.* [74], we allow the attacker to estimate that the system is in a secret state if the current state is not a secret one. By doing so, the number of obfuscation functions that can be applied is increased, which also increases the possibility of finding an obfuscation function that satisfies all desired conditions. To the author knowledge it is the first work that presents an obfuscation policy that ensures CSO with respect to the attacked subnets, avoiding silencing them, and ensuring CSU for the intended receiver.

In Figure 1.4, the security measures in the context of DES are summarized, presenting the different approaches in the literature. The main strategies in DES to protect the system against eavesdropping attacks are cryptograpy and obfuscation policies to enforce CSO. The obfuscation policies can be separated into two categories regarding the utility of the information: the methods that focus only in protecting the secret; and the methods that protect the secret while maintaining any notion of utility of the information. Regarding the methods that maintain the utility of the information, the methods can be separated into two categories: state based utility which are the methods where the utility is related to useful states; and event based utility which are the methods where the utility is related to useful events. Considering the state based utility, these methods are separated into four categories: the methods in which the utility is related to the proximity to the useful state; the methods in which the receiver is certain of the current state when the system is in an useful state; the methods in which the receiver is certain that the system is in any of the useful states; and the methods in which the receiver is certain of the current state, *i.e.*, ensuring CSU. This work, enforces CSO using obfuscation policies while also ensuring CSU. However, this work takes advantage of a security measure, network segmentation, to ensure CSU, avoiding an unsafe strategy like steganography.

A preliminary version of the multipath event routing strategy has been presented in REIS *et al.* [80] and REIS *et al.* [55]. In REIS *et al.* [80], it is proposed a method to verify the existence of an obfuscation policy to enforce CSO on a segmented network. In REIS *et al.* [55], it is proposed the computation of the opacity enforcer map, where all obfuscation policies that ensure CSO are represented. It is important

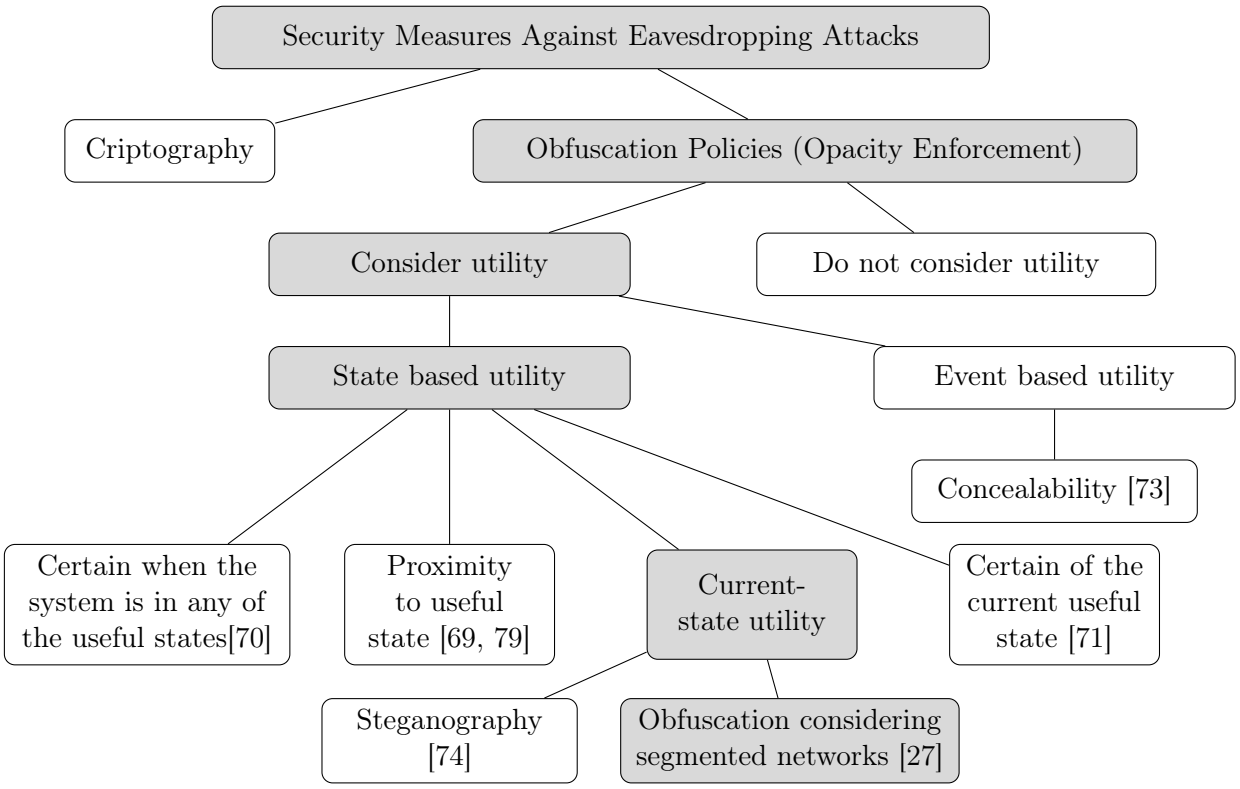


Figure 1.4: Summary of the security measures against eavesdropping attacks in the context of DES.

to remark that the opacity enforcer map proposed in REIS *et al.* [55] is different from the map presented in this work. The opacity enforcer map presented in this work has different strategies to report events through the subnets, and forces the use of attacked subnets to prevent them from being silenced for more than a given number of event transmissions. In addition, the obfuscation map and the computation of an obfuscation policy that also ensures CSU are not presented in REIS *et al.* [55].

This work is organized as follows. In Chapter 2, we present some theoretical background regarding DES. In Chapter 3, the network segmentation strategy is presented, and we formulate the OSN-CSO problem. Since the OSN-CSO problem has a high computational complexity, we present a suboptimal solution to the SN-CSO problem. In Chapter 4, we present the multipath event routing strategy, the objectives of the defense mechanism, and formulate the problem of finding an obfuscation function that enforce current state opacity. We also present in Chapter 4 an algorithm to represent how the intended receiver can understand the original message and present a necessary and sufficient condition for the obfuscation function using a multipath event routing strategy also ensures CSU. In Chapter 5, the conclusions are drawn and future works are discussed.

Chapter 2

Theoretical background

In Section 2.1 we introduce the notion of languages of a system and some operations with languages, and in Section 2.2 we present the model of a deterministic and non-deterministic automaton, the language of these automata and some basic operations using them. In Section 2.3, we present the notion of current-state opacity of DES. Finally, in Section 2.4, we present the Hall Marriage Theorem that provides a necessary and sufficient condition to determine if it is possible to choose one element from each subset of a collection of subsets without repetition. We also present an algorithm to compute a solution to this problem.

2.1 Languages

To introduce the concept of languages, it is first necessary to present some notations and definitions. The set formed of all possible events is the *"alphabet"* denoted as Σ. The concatenation of events forms sequences. The language of a system is the set of sequences that the system can execute. The length of a sequence, denoted as $\|s\|$, is the number of events that form it, considering multiple occurrences of the same event. The empty sequence ε is a sequence with zero length.

Definition 2.1 (Language [1]) *A language defined over an event set Σ is a set of finite length sequences formed of events in Σ.*

Example 2.1 *Let $\Sigma = \{a, b\}$. Then $L = \{\varepsilon, b, ab, ba, bab, baba\}$ is a language defined over Σ.* □

Since languages are sets, it is important to remark that all set operations can be applied to languages, such as union, intersection, difference, and complement. Some other operations can be applied to languages, such as the ones presented in the sequel.

2.1.1 Language Operations

Concatenation is an important operation related to the construction of sequences from a set of events Σ and languages. For example, the sequence *baba* is formed by the concatenation of the sequence *ba* with the sequence *ba*. The sequence *ba* itself is a concatenation of the event b with event a. It is important to remark that the empty sequence ε is the identity element of concatenation operation, meaning that $\varepsilon e = e\varepsilon = \varepsilon e\varepsilon = e$.

Definition 2.2 (Concatenation [1]) *Let $L_1, L_2 \subseteq \Sigma^*$, then the concatenation L_1L_2 is given by:*

$$L_1L_2 = \{s = s_1s_2 : (s_1 \in L_1) \text{ and } (s_2 \in L_2)\}.$$

According to Definition 2.2, a sequence s is in L_1L_2 if it is formed by the concatenation of sequences $s_1 \in L_1$ and $s_2 \in L_2$.

Let us denote by Σ^* the Kleene-closure of the set of events Σ, which consists of all finite length sequences that can be formed using elements of Σ, including the empty sequence ε. Thus, a language L defined over Σ is a subset of Σ^*. The Kleene-closure operation can also be applied to languages as presented in Definition 2.3 [1].

Definition 2.3 (Kleene-clousure[1]) *Let $L \subseteq \Sigma^*$. Then the Kleene-closure operation L^* is given by:*

$$L^* = \{\varepsilon\} \cup L \cup LL \cup LLL \cup \ldots$$

Consider now the sequence $s = tuv$, where $t, u, v \in \Sigma^*$. Then, t is the prefix of s, u is the subsequence of s, and v is the suffix of s. Considering that $t, u, v \in \Sigma^*$, then the sequences ε and s are also prefixes, subsequences and sufixes of s. The Prefix-closure of a language L is defined as follows [1].

Definition 2.4 (Prefix-closure [1]) *Let $L \subseteq \Sigma^*$. Then the prefix-closure operation $Pr(L)$ is given by:*

$$Pr(L) = \{s \in \Sigma^* : (\exists t \in \Sigma^*)[st \in L]\}.$$

The prefix-closure of a language L is the set of all prefixes of all sequences of L, consequently, $L \subseteq Pr(L)$. A language L is said to be prefix-closed if $L = Pr(L)$, *i.e.*, if all prefixes of all sequences of language L are also elements of L.

Other important operation applied to sequences and languages is the natural projection, defined as follows [1]:

Definition 2.5 (Projection [1]) *Consider Σ_r and Σ, such that $\Sigma_r \subset \Sigma$. The natural projection $P^{\Sigma}_{\Sigma_r} : \Sigma^* \to \Sigma_r^*$ is defined, recursively, as follows:*

$$P^{\Sigma}_{\Sigma_r}(\varepsilon) = \varepsilon,$$

$$P^{\Sigma}_{\Sigma_r}(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \in \Sigma_r, \\ \varepsilon, & \text{if } \sigma \in \Sigma \backslash \Sigma_r, \end{cases}$$

$$P^{\Sigma}_{\Sigma_r}(s\sigma) = P^{\Sigma}_{\Sigma_r}(s)P^{\Sigma}_{\Sigma_r}(\sigma) \text{ for all } s \in \Sigma^* \text{ and } \sigma \in \Sigma,$$

where $\backslash$ denotes set difference.

The projection operation $P^{\Sigma}_{\Sigma_r}(s)$ erases all events $\sigma \in \Sigma \backslash \Sigma_r$ from the sequences $s \in \Sigma^*$. This operation can be extended to languages by applying the operation to all sequences of the language.

Another important operation applied to sequences and languages is the inverse projection, defined as follows [1]:

Definition 2.6 (Inverse projection [1]) *The inverse projection $P^{\Sigma^{-1}}_{\Sigma_r} : \Sigma_r^* \to 2^{\Sigma^*}$ is defined as:*

$$P^{\Sigma^{-1}}_{\Sigma_r}(t) = \{s \in \Sigma^* : P^{\Sigma}_{\Sigma_r}(s) = t\}. \tag{2.1}$$

For a given sequence t, formed of events from Σ_r, $P^{\Sigma^{-1}}_{\Sigma_r}(t)$ produces a set formed of all the sequences s that can be constructed from Σ whose projection $P^{\Sigma}_{\Sigma_r}(s)$ is equal to t.

Similarly to the projection operation, the inverse projection can be extended to languages by applying Equation 2.1 to all sequences t that belong to the language.

The language of a DES is used to model the system behavior by representing all sequences that the system is capable of executing. Nonetheless, the representation of the system behavior using only their languages is not simple to work with. Considering this, we can use some formalism to describe DES as it becomes easier to analyze and manipulate it. The formalism adopted in this work is the automata.

2.2 Automata

An automaton is a device that is capable of representing a language according to well-defined rules, and is formally defined in the sequel [1, 2].

Definition 2.7 (Automaton [1]) *A deterministic automaton, denoted by G, is a four-tuple*

$$G = (X, \Sigma, f, x_0),$$

where X is the set of states, Σ is the set of events, $f : X \times \Sigma \to X$ is the partial transition function, and x_0 is the initial state.

The function of active events of G is denoted by $\Gamma_G : X \to 2^{\Sigma}$, where $\Gamma_G(x) = \{\sigma \in \Sigma : f(x, \sigma)!\}$, where $f(x, \sigma)!$ denotes that $f(x, \sigma)$ is defined. The domain of the transition function can be extended to $X \times \Sigma^*$.

Graphically, an automaton can be represented by a state transition diagram, which can reproduce all characteristics defined in G. The state transition diagram is formed of vertices and directed edges, represented by circles and arrows, respectively. The states of the system are represented by the vertices and the transition between states are represented by the edges. The events of Σ associated with the transitions appear as labels of the arrows. The initial state is represented by an arc with no origin state. Example 2.2 shows an automaton and its state transition diagram.

Example 2.2 *Consider an automaton $G = (X, \Sigma, f, x_0)$ with state set $X = \{0, 1, 2, 3\}$, event set $\Sigma = \{b, e, h\}$, transition function defined as $f(0, b) = 1, f(1, h) = 2, f(2, e) = 3, f(3, h) = 1$, and The initial state x_0 is 0. The active event function given by $\Gamma_G(0) = \{b\}$, $\Gamma_G(1) = \{h\}$, $\Gamma_G(2) = \{e\}$, $\Gamma_G(3) = \{h\}$. The state transition diagram representing automaton G is depicted in Figure 2.1.* □

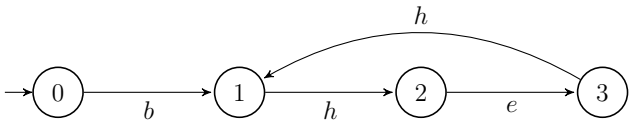


Figure 2.1: State transition diagram of Example 2.2.

In the following we present the definition of generated language [1].

Definition 2.8 (Generated language [1]) *The generated language of an automaton $G = (X, \Sigma, f, x_0)$ is defined as:*

$$L(G) = \{s \in \Sigma^* : f(x_0, s) \text{ is defined}\}$$

Additionally, for any automaton such that $X \neq \emptyset$, $\varepsilon \in L(G)$.

The language $L(G)$ is composed of all sequences that can be generated by following the transitions of the state transition diagram starting at the initial state. Consequently, since a sequence is only feasible if all its prefixes are also feasible, the generated language $L(G)$ is always prefix-closed. Furthermore, if f is a total function over its domain, then $L(G) = \Sigma^*$.

Definition 2.9 (Nondeterministic Automaton [1]) *A nondeterministic automaton, denoted by $\mathcal{G}$, is a four-tuple*

$$\mathcal{G} = (X_{\mathcal{G}}, \Sigma, f_{\mathcal{G}}, X_{\mathcal{G},0}),$$

$X_{\mathcal{G}}$ is the set of states, Σ is the finite set of events, $f_{\mathcal{G}} : X_{\mathcal{G}} \times \Sigma \to 2^{X_{\mathcal{G}}}$ is the nondeterministic transition function, and $X_{\mathcal{G},0} \subseteq X_{\mathcal{G}}$ is the set of initial states.

The domain of the transition function $f_{\mathcal{G}}$ can be extended to $X_{\mathcal{G}} \times \Sigma^*$, recursively as: $f_{\mathcal{G}}(x, \varepsilon) = \{x\}$, and $f_{\mathcal{G}}(x, s\sigma) = \bigcup_{y \in f_{\mathcal{G}}(x,s)} f_{\mathcal{G}}(y, \sigma)$, for all $s \in \Sigma^*$ and $\sigma \in \Sigma$. The language generated by $\mathcal{G}$ is defined as $L(\mathcal{G}) = \{s \in \Sigma^* : (\exists x_0 \in X_0)[f_{\mathcal{G}}(x_0, s)!]\}$.

2.2.1 Operations on automata

There are several operations that can be applied to automata, which can be separated into two groups: unary and composition operations.

Unary operations

Unary operations are applied to a single automaton, altering its state transition diagram, but keeping its event set the same. In the following, we present the accessible part of an automaton.

Definition 2.10 (Accessible part [1]) *Consider automaton $G = (X, \Sigma, f, x_0)$. The accessible part of G, denoted as $Ac(G)$, is defined as:*

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

where $X_{ac} = \{x \in X : (\exists s \in \Sigma^*)[f(x_0, s) = x]\}$ and $f_{ac} : X_{ac} \times \Sigma \to X_{ac}$. The transition function f_{ac} differs from the transition function f due to the restricted domain of the accessible states X_{ac}.

In the accessible part of an automaton G, the states which are not reachable from the initial state x_0 and its associated transitions are erased from G. Note that this operation does not modify the generated language of G.

Composition operations

Composition operations applied to DES modeled by automata are those that allow us to combine automata, resulting in a new automaton.

In general, complex systems are formed of several components, which work together to accomplish their tasks. It is possible that those components have particular events. The usual way to obtain the global model of the system from the model

of its components is by making the parallel composition of the component models. The parallel composition allows each component to maintain its private behavior and synchronize only the common events of the components. In the following we present the formal definition of the parallel composition.

Definition 2.11 (Parallel composition [1]) *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 automata. The parallel composition of* G_1 *and* G_2 *results in automaton:*

$$G_1 \| G_2 = Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1\|2}, (x_{0,1}, x_{0,2})),$$

where

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

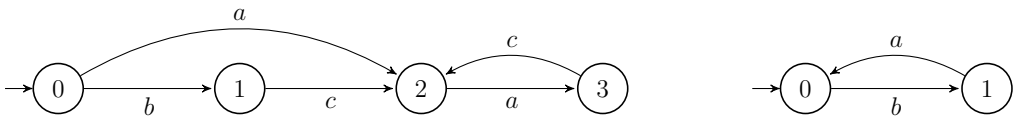
It is important to remark that in the parallel composition an event $\sigma \in \Sigma_1 \cap \Sigma_2$ can only occur in the composed automaton $G_1 \| G_2$ if it is simultaneously enabled in both states of G_1 and G_2. The private events, on the other hand, can be executed whenever possible in its automaton, *i.e.,* events in $\Sigma_1 \backslash \Sigma_2$ can occur in $G_1 \| G_2$ when possible in G_1 and events in $\Sigma_2 \backslash \Sigma_1$ can occur in $G_1 \| G_2$ when possible in G_2.

The generated language of the parallel composition $G_1 \| G_2$ is obtained by using the natural projections $P_{\Sigma_i}^{\Sigma_1 \cup \Sigma_2} = (\Sigma_1 \cup \Sigma_2)^* \rightarrow \Sigma_i^*$, for $i = 1, 2$. The generated language of the parallel composition is $L(G_1 \| G_2) = P_{\Sigma_1}^{\Sigma_1 \cup \Sigma_2^{-1}}(L(G_1)) \cap P_{\Sigma_2}^{\Sigma_1 \cup \Sigma_2^{-1}}(L(G_2))$. In the sequel we present an example of parallel composition.

Example 2.3 *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 automata, with event sets* $\Sigma_1 = \{a, b, c\}$ *and* $\Sigma_2 = \{a, b\}$*, whose state transition diagrams are presented in Figures 2.2a and 2.2b, respectively. The automaton* G_{par} *is obtained by making the parallel composition between* G_1 *and* G_2*,* $G_{par} = G_1 \parallel G_2$*, which is presented in Figure 2.3.* □

2.2.2 Automata with partially observed events

In real word systems, the observation of the occurrence of all events is a very difficult task. The number of sensors and the position where they must be placed in order to provide the information for the occurrence of events usually make it impossible



(a) G_1. (b) G_2.

Figure 2.2: Automata G_1 and G_2 of Example 2.3.

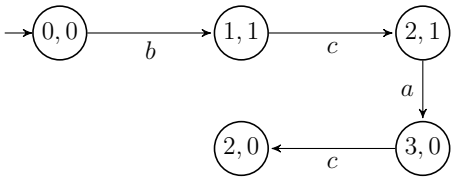


Figure 2.3: Automaton G_{par} of Example 2.3.

to observe all events of interest. To represent that, we introduce the notion of unobservable events. Thus, the event set Σ

Definition 2.12 (Unobservable reach [1]) *The unobservable can be partitioned as $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, where $\dot{\cup}$ represents the disjoint union.*

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

$$UR(x) = \{y \in X : (\exists t \in \Sigma_{uo}^*)[f(x,t) = y]\}.$$

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

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

The unobservable reach of a state x is the set of states composed of all states reached from x by sequence of transitions labeled with unobservable events and the state itself. It is possible to build a deterministic automaton from G that generates the observed language of G, $P_{\Sigma_o}^{\Sigma}(L)$, using the unobservable reach. This automaton is called the observer automaton of G, denoted as $Obs(G, \Sigma_o)$ [1]. To do so, the states of the observer automaton are the state estimates of the system. Since the states are the state estimates, the number of the states of the observer automaton may grow exponentially because each state of te observer automaton is in 2^X.

In the sequel, we present an example of the construction of the observer $Obs(G, \Sigma_o)$ of G.

Example 2.4 *Consider automaton G shown in Figure 2.4a. The set of states is $X = \{0, 1, 2, 3\}$ and the set of events is $\Sigma = \{e, h, \sigma_1, \sigma_2\}$, where $\Sigma_o = \{e, h\}$ and $\Sigma_{uo} = \{\sigma_1, \sigma_2\}$. The observer automaton of G, $Obs(G, \Sigma_o)$, is shown in Figure 2.4b. Let us suppose that sequence $s = h\sigma_1 eh$ has been executed. In this case, the observed*

sequence is $P^{\Sigma}_{\Sigma_o}(s) = heh$*, where* $P^{\Sigma}_{\Sigma_o} : \Sigma^* \to \Sigma_o^*$*. After observing the sequence* heh*, it is impossible to be certain in which state the system is, but it is possible to estimate it, and in this case the estimated states are* 1, 2 *and* 3*. This can be seen in Figure 2.4b, where each state represent the state estimate of* G *after observing a trace* $s \in P^{\Sigma}_{\Sigma_o}(L)$*.* □

(a) Automaton G. (b) $Obs(G, \Sigma_o)$.

Figure 2.4: Observer $Obs(G, \Sigma_o)$ of Example 2.4.

2.3 Current-state Opacity

Opacity is an important information flow property related to privacy and security. A system is said to be opaque if an intruder observes the information flow and cannot infer that the secret has been revealed. In this section, we present a notion of opacity that is based on states. In this notion, the secret information is modeled as a set of states, called set of secret states, $X_S \subset X$. In this work, the notion of opacity is related to the current-state of the system, the objective is that the intruder cannot infer that the current-state of the system is a secre state, and this notion is called current-state opacity.

Definition 2.13 (Current-State Opacity [56]) *Given* $G = (X, \Sigma, f, x_0)$*, a projection* $P^{\Sigma}_{\Sigma_o}$*, a set of secret states* $X_S \subset X$*, and a set of non-secret states* $X_{NS} = X \setminus X_S$*,* G *is current-state opaque if* $\forall t \in L(G)$ *satisfying* $f(x_0, t) \in X_S$*,* $\exists t' \in L(G)$*, such that* $f(x_0, t') \in X_{NS}$ *and* $P^{\Sigma}_{\Sigma_o}(t) = P^{\Sigma}_{\Sigma_o}(t')$*.* □

Example 2.5 *Let us consider the system modeled by* G*, presented in Figure 2.5, where the generated Language is* $L(G) = \{Pr(bac), Pr(acb)\}$*,* $\Sigma_o = \{a, c\}$*, and* $\Sigma_{uo} = \{b\}$*. Also, consider* $X_S = \{3\}$*. Thus, it is possible to say that* G *is CSO with respect to* X_S *and* $P^{\Sigma}_{\Sigma_o}$ *since the secret state* 3 *is reached from the initial state 0 only with sequence* $t = bac$ *and a non-secret state* 6 *is reached with sequence* $t' = acb$ *and* $P^{\Sigma}_{\Sigma_o}(t) = P^{\Sigma}_{\Sigma_o}(t') = ac$*.* □

According to Definition 2.13, if the system is CSO with respect to X_S and $P^{\Sigma}_{\Sigma_o}$, then the attacker is not able to infer when the system reaches a state in X_S by observing only the events generated by the system that belong to Σ_o.

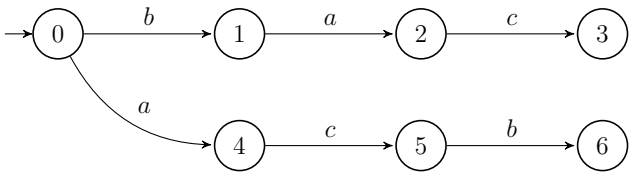


Figure 2.5: Automaton G for Example 2.5.

It is important to remark that the verification of CSO with respect to X_S and $P^{\Sigma}_{\Sigma_o}$ is PSPACE-complete [49], and has, in the worst-case, exponential complexity with respect to the number of system states [42], since it requires the computation of the observer automaton of G, $G_{obs} = Obs(G, \Sigma_o)$. The system G is not CSO with respect to X_S and P_o if, and only if, G_{obs} has at least one state formed only of states of X_S. Thus, the verification of CSO is carried out with complexity $O(2^{|X|})$.

2.4 Complete Sets of Distinct Representatives

In this section, we present a method to verify a necessary and sufficient condition for situations where distinct elements can be chosen from a collection of overlapping finite sets and a method to make these choices, which will be used to ensure CSO and CSU.

2.4.1 Hall's Marriage Theorem

The Hall's Marriage Theorem [81] gives a necessary and sufficient condition for situations where distinct elements can be chosen from a collection of overlapping finite sets. For example, consider a group of women, and each woman has a set of men she intends to marry. Hall's Marriage Theorem provides a necessary and sufficient condition to determine whether it is possible for each woman to marry a different man. The Theorem has two equivalent formulations, the combinatorial formulation and the graph theoretic formulation. The former is related to whether a finite collection of sets has a transversal, which is whether an element can be picked from each set without repetition. The latter is related to whether a finite bipartite graph has a perfect matching, which is a way to match each vertex from one group uniquely to an adjacent vertex from the other group.

Combinatorial Formulation

Let $S = \{a_1, a_2, \ldots, a_\ell\}$ be a set and $F = \{F_1, F_2, \ldots, F_m\}$ be a family of subsets of S, where $\ell \geq m$. Consider the problem of verifying the existence of complete sets of distinct representatives (CDR) for the family F, *i.e.*, a set of m distinct elements of S, $a_1, a_2, \ldots, a_m$, such that $a_i \in F_i$, for each $i = 1, 2, \ldots, m$. Then, the following

theorem provides a necessary and sufficient condition for the existence of a CDR for the family F [81].

Theorem 2.1 *Let $S = \{a_1, a_2, \ldots, a_\ell\}$ be a set and $F = \{F_1, F_2, \ldots, F_m\}$ be a family of subsets of S, where $\ell \geq m$. Let $I_m = \{1, 2, \ldots, m\}$. Then, there exists a CDR for F if, and only if, for all $B \in 2^{I_m}$,*

$$|B| \leq \left| \bigcup_{i \in B} F_i \right|.$$

Example 2.6 *Consider family $F = \{F_1, F_2, F_3, F_4\}$, where $F_1 = \{A, B, C\}$, $F_2 = \{B, C\}$, $F_3 = \{C, D\}$, and $F_4 = \{A, C, D\}$. The set S is formed as $F_1 \cup F_2 \cup F_3 \cup F_4$. Thus, $S = \{A, B, C, D\}$. Let us now verify the existence of a CDR for family F.*

Let $I_m = \{1, 2, 3, 4\}$. Note that for all $B \in 2^{I_m}$, $|B| \leq \left| \bigcup_{i \in B} F_i \right|$. For example, if $B = \{2, 3\}$, $|B| = 2 \leq |\{B, C, D\}| = 3$, and, in this case, a solution could be choosing $\{B\}$ for family F_2 and $\{D\}$ for family F_3. Since there exists a solution for all $B \in 2^{I_m}$, then, there exists a CDR for family F. □

Example 2.7 *Consider family $F = \{F_1, F_2, F_3, F_4, F_5\}$, where $F_1 = \{B, C\}$, $F_2 = \{B, C, E\}$, $F_3 = \{C, D\}$, $F_4 = \{B, C, D\}$, and $F_5 = \{B, D\}$. The set S is formed as $F_1 \cup F_2 \cup F_3 \cup F_4 \cup F_5$. Thus, $S = \{A, B, C, D, E\}$. Let us verify the existence of a CDR for family F.*

Let $I_m = \{1, 2, 3, 4, 5\}$. Note that if $B = \{1, 3, 4, 5\}$, $|B| = 4 > |\{B, C, D\}| = 3$. Thus, there does not exist a CDR for family F. □

2.4.2 Edmonds-Karp Algorithm

For a selected family F, after verifying if there exist a CDR for F with Theorem 2.1, it is possible to obtain a CDR using the Edmonds-Karp algorithm [82]. The Edmonds-Karp algorithm was originally presented for obtaining the maximum flow in a flow network, but with adjustments, can be used to obtain a CDR for a family F as presented in Algorithm 1.

Example 2.8 *Consider family $F = \{F_1, F_2, F_3, F_4\}$ presented in Example 2.6, where $F_1 = \{A, B, C\}$, $F_2 = \{B, C\}$, $F_3 = \{C, D\}$, and $F_4 = \{A, C, D\}$. In Example 2.6 is shown that there exists a CDR for family F. Now, let us find a possible CDR with Algorithm 1.*

In line 1 of Algorithm 1, the vertex for F_1, F_2, F_3, and F_4 are created. Then, in line 2, the vertex for A, B, C, and D are created. In lines 3 to 5, each vertex that represent F_i is connected to each element $e \in F_i$, labeled with $0/\infty$, for example, F_1 is connected with A, B, and D. In line 6, a source vertex, s, is created, and, in lines

Algorithm 1: Edmonds-Karp algorithm [82] applied to obtain a CDR for family F

Input : Family F, set of elements S.
Output: a CDR solution

Create a vertex for each element $F_i \in F$;
Create a vertex for each element in $e \in S$;
foreach $F_i \in F$ **do**
  Connect F_i to the elements of $e \in S$ such that $e \in F_i$ labeled with $0/\infty$
end
Create a source vertex s;
foreach $F_i \in F$ **do**
  Connect s to F_i labeled with $0/1$
end
Create a destiny vertex d;
foreach $e \in S$ **do**
  Connect e to d labeled with $0/1$
end
while *there exists a connection between* s *and* F_i *labeled with* $0/\infty$ **do**
  Find the shortest path from s to d in which the connection between s with F_i and e with d are labeled with $0/1$;
  Update the labels from the connections s with F_i and e with d to $1/1$;
  Update the labels from the connection from F_i to e from $0/\infty$ to $1/\infty$;
  Update the labels from the connection from e to F_i from $1/\infty$ to $0/\infty$;
end
The solution of the CDR is, for each F_i, the element in which the connection is labeled as $1/\infty$.

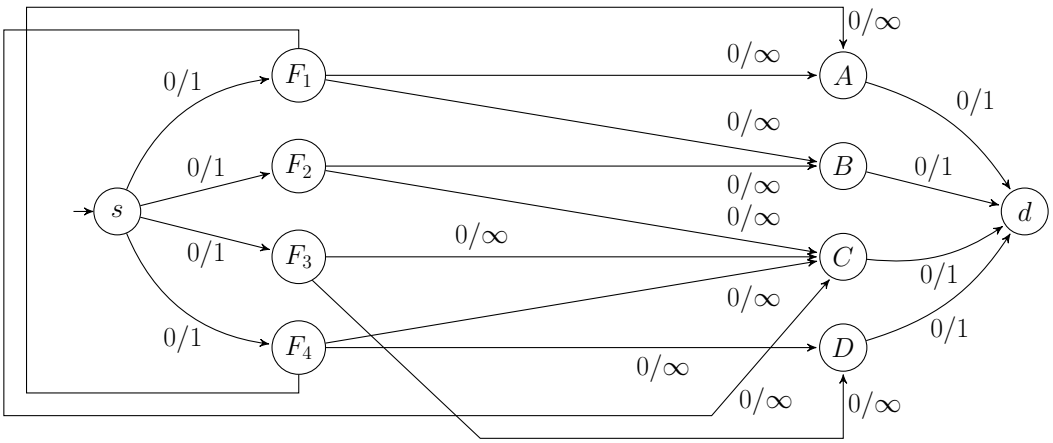


Figure 2.6: Edmonds-Karp algorithm example.

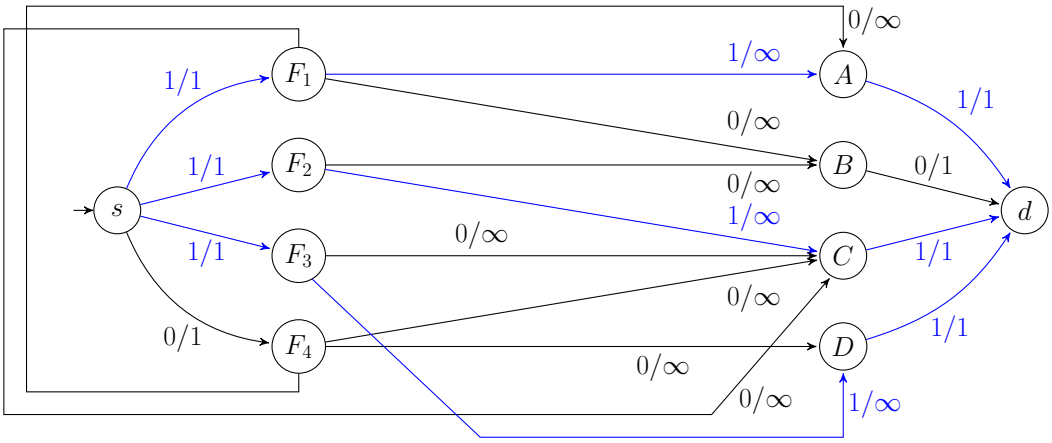


Figure 2.7: Edmonds-Karp algorithm example.

7 to 9, s is connected to F_1, F_2, F_3, and F_4, each connection labeled with $0/1$. Then, in line 10 a destiny vertex, d, is created, and in lines 11 to 13 each element $e \in S$ is connected to d labeled with $0/1$. This steps are represented in Figure 2.6.

Then, in lines 14 to 19 the choice of the distinct representative $e \in S$ for each family F_i is performed. First, the shortest path from s to d must be selected. At this point, there are several paths with the same length. Thus, let us choose (s, F_1, A, d). Then, according to line 16, the label of the connection from s to F_1 is updated to $1/1$. According to line 17, the label of the connection from F_1 to A is updated to $1/\infty$, and, according to line 18, the label of the connection from A to d is updated to $1/1$. Then, since the label of the connection from s to F_1 and from A to d are $1/1$, then it is not possible to connect s to F_1 nor A to d again.

Then, let us choose the paths (s, F_2, C, d) and (s, F_3, D, d), and update the labels according to lines 16 to 18. Figure 2.7 represents how the choices have been made, i.e., $\{A\}$ was chosen to F_1, $\{C\}$ was chosen to F_2, and $\{D\}$ was chosen to F_3. The blue arrows are highlighted to represent the paths already defined. Note that the vertex F_4 connects with the vertex A, C and D, and these vertex cannot be connected to vertex d since their flow is already fulfilled. In addition, it is possible to choose the path (s, F_4, C, F_2, B, d), and according to line 18, since the connection from C to F_2 is labeled with $1/\infty$, it is updated to $0/\infty$. The result is presented in Figure 2.8, where the chosen paths are highlighted in blue. Thus, the CDR solution obtained following the steps of Algorithm 1 is choosing $\{A\}$ for family F_1, choosing $\{B\}$ for family F_2, choosing $\{D\}$ for family F_3, and choosing $\{C\}$ for family F_4. It is important to remark that choosing $\{C\}$ for F_2 was a wrong choice that would lead to a no solution of the CDR problem. However, Algorithm 1 correct this wrong choices with the procedure in line 18. Thus, it is not necessary to restart the choice of the elements, resulting in the reduction in the computational complexity. □

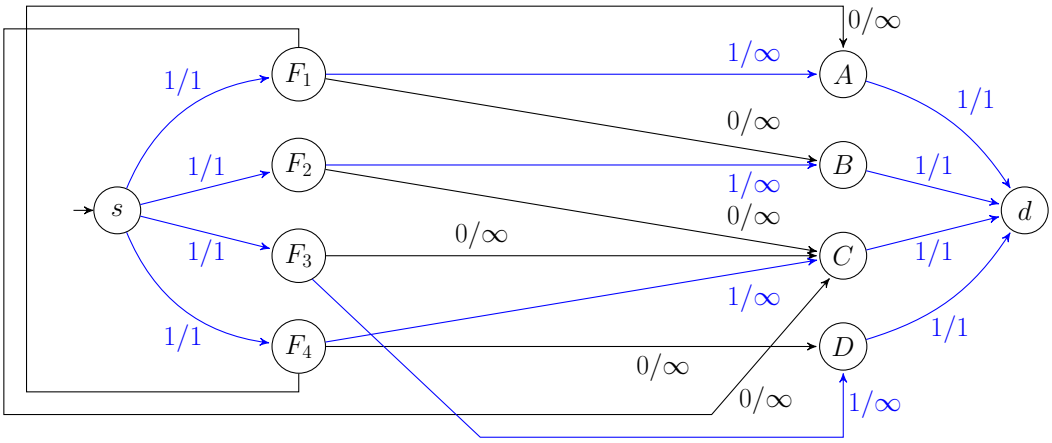


Figure 2.8: Edmonds-Karp algorithm example.

The computational complexity of Algorithm 1 is $O\left(|V| \times |E|^2\right)$, where V are the vertex and E are the edges.

2.5 Concluding remarks

In this chapter, the background of DES has been presented. This background includes the definition of languages and their operations, the automaton formalism to represent DES, automata with partially observed events, and the concepts of opacity of DES. In addition, we have also presented the Hall's marriage theorem [81], which provides a necessary and sufficient condition for choosing distinct elements from a collection of overlapping finite sets. We have also presented the Edmonds-Karp algorithm [82] to obtain a CDR for a family F. In the next chapter, we present a new method to enforce current-state opacity, focusing on networked discrete events systems, using network segmentation.

Chapter 3

Network Segmentation with the Aim of Ensuring Current-State Opacity

In this chapter, we propose a new approach to enforce current-state opacity (CSO) of cyberphisical systems (CPS) abstracted as networked discrete eventsystems (NDES). We assume that the NDES communicates with an external observer through one communication network, and that this network is vulnerable to eavesdropping attacks. We consider that the secret information is modeled by secret states of the plant, and the attacker is capable of observing the events transmitted through the attacked network. Thus, after a successful attack, the attacker is capable of inferring the system secrets. We also consider that the attacker knows the plant model represented by an automaton. The external observer considered in this work is any entity that receives data from the plant through a communication network, such as a supervisory control and data acquisition (SCADA) system, or a diagnoser. The proposed method uses a network segmentation approach to split the communication of the events into different subnets and we assume that the attacker is capabe of invading only one subnet at a time. To protect the system, we propose to split the communication of the observed events into different subnets in a manner that the attacker is not capable of inferring the secret states after invading one subnet, *i.e.*, CSO is ensured considering the events communicated through each subnet individually.

3.1 Optimal segmented network current-state opacity problem

Let us consider that the CPS is abstracted as an NDES, modeled by automaton $G = (X, \Sigma, f, x_0)$, and let us consider that all events of Σ are transmitted to an intended receiver, which monitors the system behavior. Let us also assume that the

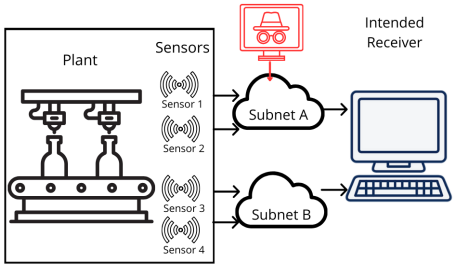


Figure 3.1: Segmented network where only one subnet is attacked.

system secret is modeled by a set of secret states $X_S \subset X$. Since all events of Σ are observable for the intended receiver, then G is not current-state opaque with respect to the set of secret states X_S and the observation of the events of Σ.

Since communication networks are used for the transmission of event occurrences between the plant and the monitoring system (intended receiver), the transmitted information is subject to eavesdropping attacks, where the intruder invades the communication network with the aim of discovering the system secret. If an attacker successfully invades the communication network, then they can observe all the events communicated through the invaded network and the secret is revealed. Thus, strategies must be employed to ensure the security of the system. Among the security measures that can be adopted, we use the network segmentation, which consists of separating the network into isolated and independent subnets, limiting lateral movements of attackers. Thus, if an attacker invades only one subnet of the segmented network, then they can only observe the events transmitted through this specific subnet.

Let us assume that the attacker can invade any subnet of the segmented network but only one at a time, as depicted in Figure 3.1. Since the secret of the system is modeled by secret states, then, it is necessary to ensure that the system is current-state opaque with respect to the set of secret states and the observation of the events transmitted in each subnet of the segmented network. Thus, the choice of which events are transmitted through each subnet is crucial for guaranteeing the security of the information. This leads to the need for partitioning the event set as $\Sigma = \Sigma_1 \dot{\cup} \Sigma_2 \dot{\cup} \ldots \dot{\cup} \Sigma_n$, such that the system is CSO with respect to X_S and $P^{\Sigma}_{\Sigma_i} : \Sigma^* \to \Sigma_i^*$, for $i = 1, \ldots, n$. After obtaining the partition, each subnet is used for the transmission of events belonging to a given subset Σ_i. In this case, the number of isolated subnets of the segmented network is equal to n. Each subset Σ_i is called, in this paper, a partition subset (PS). We call the problem of finding a partition of the event set, such that the system is CSO with respect to each one of the PS, the segmented network current-state opacity (SN-CSO) problem. In the following we present a system in which the network segmentation was applied in

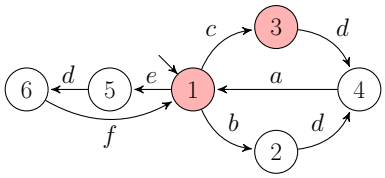
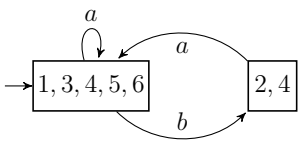


Figure 3.2: Automaton G representing the system.

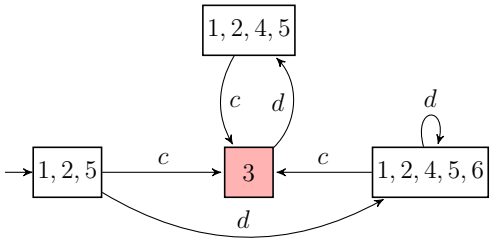
two different cases. In the first case, if the attacker successfully invades one specific subnet, then the secret is revealed. In the second, the system is CSO with respect to any of the PS.

Example 3.1 *Let G, depicted in Figure 3.2, be the plant model, where $X = \{1,2,3,4,5,6\}$ and $\Sigma = \{a,b,c,d,e,f\}$. In addition, let $X_S = \{1,3\}$ be the set of secret states, which are marked in red in Figure 3.2. Let us now partition the set of events in three PS, $\Sigma_1 \dot{\cup} \Sigma_2 \dot{\cup} \Sigma_3$, where $\Sigma_1 = \{a,b\}$, $\Sigma_2 = \{c,d\}$, and $\Sigma_3 = \{e,f\}$.*

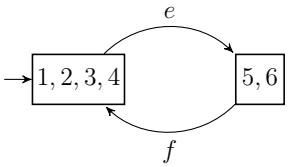
In this case, to verify if the system is CSO with respect to each PS, we have to compute the observer considering Σ_1, Σ_2, and Σ_3, depicted in Figures 3.3a, 3.3b, and 3.3c, respectively. Note, in Figure 3.3b, that the system is not CSO with respect to X_S and $P^{\Sigma}_{\Sigma_2} : \Sigma^* \to \Sigma_2^*$ since after observing event c, the only possible state for the system is 3, which is a secret state.



(a) Observer automaton $Obs(G, \Sigma_1)$.



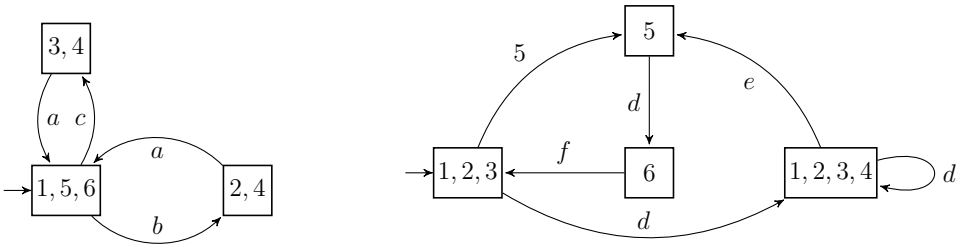
(b) Observer automaton $Obs(G, \Sigma_2)$.



(c) Observer automaton $Obs(G, \Sigma_3)$.

Figure 3.3: Observer automata for partition $\Sigma = \Sigma_1 \dot{\cup} \Sigma_2 \dot{\cup} \Sigma_3$ for Example 3.1.

On the other hand, let us now partition the set of events in two PS, $\Sigma = \Sigma'_1 \dot{\cup} \Sigma'_2$, where $\Sigma'_1 = \{a,b,c\}$, $\Sigma'_2 = \{d,e,f\}$. In this case, to verify if the system is CSO with respect to each PS, we have to compute the observer considering Σ'_1 and Σ'_2, depicted in Figures 3.4a and 3.4b, respectively. Note, in Figures 3.4a and 3.4b, that it is not possible to infer that the system is in a secret state. Thus, since the system is CSO with respect to X_S and $P^{\Sigma}_{\Sigma'_1}$, and with respect to X_S and $P^{\Sigma}_{\Sigma'_2}$, the partition $\Sigma = \Sigma'_1 \dot{\cup} \Sigma'_2$ is a solution to the SN-CSO problem. □



(a) Observer automaton $Obs(G, \Sigma'_1)$. (b) Observer automaton $Obs(G, \Sigma'_2)$.

Figure 3.4: Observer automata for partition $\Sigma = \Sigma'_1 \dot{\cup} \Sigma'_2$ for Example 3.1.

It is important to remark that the cost of the implementation of a segmented network increases with the number of isolated subnets. Thus, it is desirable to find a partition with the minimum number of PS satisfying the SN-CSO problem. The problem of finding the partition of Σ with the smallest possible number of PS is called the optimal segmented network current-state opacity (OSN-CSO) problem, formally defined as follows.

OSN-CSO Problem: Let $G = (X, \Sigma, f, x_0)$. Find a partition of the set of events $\Sigma = \Sigma_1 \dot{\cup} \Sigma_2 \dot{\cup} \ldots \dot{\cup} \Sigma_n$, such that the following two conditions are satisfied:

C1. G is CSO with respect to X_S and $P^{\Sigma}_{\Sigma_i}$, for all $i \in \{1, 2, \ldots, n\}$.

C2. For all event set partition $\Sigma = \Sigma_1 \dot{\cup} \Sigma_2 \dot{\cup} \ldots \dot{\cup} \Sigma_{n'}$ where $n' < n$, there exists $j \in 1, 2, ..., n'$ such that G is not CSO with respect to X_S and $P^{\Sigma}_{\Sigma_j}$.

Example 3.2 *Let us consider again the system G from Example 3.1, where $X = \{1, 2, 3, 4, 5, 6\}$, $\Sigma = \{a, b, c, d, e, f\}$, and $X_S = \{1, 3\}$. Also, consider the partition $\Sigma = \Sigma'_1 \dot{\cup} \Sigma'_2$, where $\Sigma'_1 = \{a, b, c\}$, $\Sigma'_2 = \{d, e, f\}$ that is a solution to the SN-CSO problem. Let us verify if the partition $\Sigma = \Sigma'_1 \dot{\cup} \Sigma'_2$, is a solution the OSN-CSO problem.*

Since G is CSO with respect to X_S and $P^{\Sigma}_{\Sigma'_1}$ and is CSO with respect to X_S and $P^{\Sigma}_{\Sigma'_2}$, Condition **C1** *is satisfied. In addition, the system is not CSO with respect to any $P^{\Sigma}_{\Sigma_\alpha}$ such that $\{c, d\} \subseteq \Sigma_\alpha$ since if both events c and d are observable, it is possible to be certain that the system visit secret state 3. Then, since events c and d must be communicated through different subnets, these events are in different PS. Thus, there must exist at least 2 PS, since if we have $n' = 1$ PS, we have that $\Sigma = \Sigma_1$ resulting in G being not CSO with respect to X_S and $P^{\Sigma}_{\Sigma_j}$. Thus, the partition $\Sigma = \Sigma'_1 \dot{\cup} \Sigma'_2$, is a solution the OSN-CSO problem.* □

3.2 Solution to the OSN-CSO problem

We propose a solution to the OSN-CSO problem that is obtained in two steps: (i) compute all maximal subsets of Σ, $\Sigma_{max,i} \subset \Sigma$, called the maximal subsets for

opacity (MSO), such that G is CSO with respect to X_S and $P^{\Sigma}_{\Sigma_{max,i}} : \Sigma^* \to \Sigma^*_{max,i}$; (ii) form a partition of Σ from the MSO with the smallest possible number of PS.

A straightforward method to compute all MSO is to verify the CSO of the system with respect to each subset of Σ, and select the maximal ones. Thus, this search has complexity $O(2^{|\Sigma|})$. However, as shown in the following theorem, in some cases, the number of subsets of Σ that must be verified to find all MSO can be reduced.

Theorem 3.1 *Let $G = (X, \Sigma, f, x_0)$ be the automaton that models the system, X_S be the set of secret states, and $P^{\Sigma}_{\Sigma_1} : \Sigma^* \to \Sigma_1^*$ and $P^{\Sigma}_{\Sigma_2} : \Sigma^* \to \Sigma_2^*$ be projections, where $\Sigma_1 \subseteq \Sigma_2$. Then, if G is not current-state opaque with respect to X_S and P_{Σ_1}, then G is also not current-state opaque with respect to X_S and P_{Σ_2}.*

Proof: The proof is straightforward from the fact that if, for a sequence $t \in L(G)$, there does not exist a sequence $t' \in L(G)$ such that $P^{\Sigma}_{\Sigma_1}(t) = P^{\Sigma}_{\Sigma_1}(t')$, then there does not exist a sequence $t'' \in L(G)$ such that $P^{\Sigma}_{\Sigma_2}(t) = P^{\Sigma}_{\Sigma_2}(t'')$. □

Using Theorem 3.1, the number of opacity verifications can be greatly reduced if the system is not CSO for a subset of Σ with a small number of elements.

Example 3.3 *Let us consider again the system G from Example 3.1, where $X = \{1, 2, 3, 4, 5, 6\}$, $\Sigma = \{a, b, c, d, e, f\}$, and $X_S = \{1, 3\}$. In order to compute all MSO it is necessary to compute $2^6 - 1$ observers since the empty set does not need to be computed.*

However, According to Theorem 3.1, the number of subsets of Σ that must be verified to find all MSO can be reduced. As shown in Example 3.1, G is not CSO with respect to X_S and $\{c, d\}$. Then, according to Theorem 3.1, G is not CSO with respect to X_S and any set Σ_i that satisfies $\Sigma_i \supseteq \{c, d\}$. Thus, the number of observers required to verify CSO is reduced from 63 to 48. □

It is important to remark that the opacity verification is based on the construction of an observer automaton, and thus, has complexity $O(2^{|X|})$. In order to mitigate the complexity of computing an observer automaton for each subset of Σ, we present the following sufficient condition for non-opacity of the system.

Theorem 3.2 *Let G be the automaton model of the system and $\Sigma_k \subset \Sigma$. If there exists $x_S \in X_S$ such that, for all $s \in L(G)$ satisfying $f(x_0, s) = x_S$, there does not exist a sequence $t \in L(G)$ such that $f(x_0, t) \in X \setminus X_S$ and $P^{\Sigma}_{\Sigma_k}(s) = P^{\Sigma}_{\Sigma_k}(t)$, then G is not current-state opaque with respect to X_S and $P^{\Sigma}_{\Sigma_k}$.*

Proof: Let $L(G, x_S) = \{s \in L(G) : f(x_0, s) = x_S\}$. If there does not exist a sequence $t \in L(G)$ such that $f(x_0, t) \in X \setminus X_S$ and $P^{\Sigma}_{\Sigma_k}(t) = P^{\Sigma}_{\Sigma_k}(s)$, for all $s \in L(G, x_S)$, then the state estimate of $G_k = Obs(G, \Sigma_k)$ contains only secret states

after the observation of $P^{\Sigma}_{\Sigma_k}(s)$, for any $s \in L(G, x_S)$, which shows, according to Definition 2.13, that G is not CSO with respect to X_S and $P^{\Sigma}_{\Sigma_k}$. □

According to Theorem 3.2, G is not current-state opaque with respect to X_S and $P^{\Sigma}_{\Sigma_k}$ if the observation of any sequence $s \in L(G)$ that leads to a secret state $x_S \in X_S$, does not have the same observation of any sequence $t \in L(G)$ that leads to a non-secret state of G. Thus, after the observation of s the attacker is certain that a secret state has been reached observing only the events in Σ_k.

We present in Algorithm 2 a method to verify the sufficient condition for non-opacity presented in Theorem 3.2. To do so, we first introduce the renaming function $\gamma : \Sigma \to \Sigma_\gamma$ as follows: $\gamma(\sigma) = \sigma$, if $\sigma \in \Sigma_k$, and $\gamma(\sigma) = \sigma_\gamma$, if $\sigma \in \Sigma \setminus \Sigma_k$ [16]. In addition, $\Sigma_\gamma = \gamma(\Sigma) = \bigcup_{\sigma \in \Sigma}\{\gamma(\sigma)\}$.

Algorithm 2: Verification of the sufficient condition for non-opacity

Input : G, Σ_k.
Output: $D \in \{True, False\}$
1 Create automaton $G_\gamma = (X, \Sigma_\gamma, f_\gamma, x_0)$, where $f_\gamma(x, \sigma) = f(x, \sigma)$ for all $x \in X$ and $\sigma \in \Sigma_k$, and $f_\gamma(x, \sigma_\gamma) = f(x, \sigma)$ for all $x \in X$ and $\sigma \in \Sigma \setminus \Sigma_k$;
2 Compute $G_V = G \| G_\gamma = (X_V, \Sigma \cup \Sigma_\gamma, f_V, (x_0, x_0))$;
3 $D = False$;
4 **for each** $x_1 \in X_S$ **do**
5 | **if** $\nexists (x_1, x_2) \in X_V$ *such that* $x_2 \in X \setminus X_S$ **then**
6 | | $D = True$
7 | **end**
8 **end**

Note that Algorithm 2 suggests that, if automaton G and Σ_k satisfy Theorem 3.2, then $D = True$. Before we present the proof of correctness of Algorithm 2, we present the following lemma [83].

Lemma 3.1 *Let $G_1 = (X_1, \Sigma_1, f_1, x_{0,1})$ and $G_2 = (X_2, \Sigma_2, f_2, x_{0,2})$, where $\Sigma_1 = \Sigma_2 = \Sigma$, let $\Sigma_k \subset \Sigma$, and let γ be the renaming function.*

Construct automaton $G_{\gamma_1} = (X_1, \Sigma_{\gamma_1}, f_{\gamma_1}, x_{0,1})$, where $\Sigma_{\gamma_1} = \gamma(\Sigma_1)$, $f_{\gamma_1}(x_1, \gamma(\sigma)) = f_1(x_1, \sigma)$, for all $\sigma \in \Sigma_1$. Let $V_{12} = G_{\gamma_1} \| G_2 = (X_\nu, \Sigma_\nu, f_\nu, x_{0,\nu})$. Then, for every $v \in L(V_{12})$ there exist $t_1 \in L(G_1)$ and $t_2 \in L(G_2)$, such that $P^{\Sigma}_{\Sigma_k}(t_1) = P^{\Sigma}_{\Sigma_k}(t_2)$, and conversely.

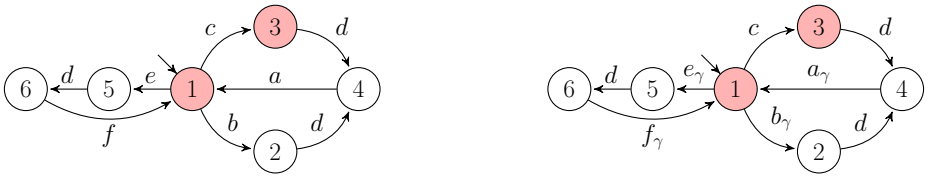
Proof: See [83]. □

Theorem 3.3 *Let G be the automaton that models the system and $\Sigma_k \subset \Sigma$. Then, the output of Algorithm 2 is $D = True$ if, and only if, there exists $x_S \in X_S$ such that, for all $s \in L(G)$ satisfying $f(x_0, s) = x_S$, and for all sequence $t \in L(G)$ satisfying $f(x_0, t) \in X \setminus X_S$ and $P^{\Sigma}_{\Sigma_k}(s) \neq P^{\Sigma}_{\Sigma_k}(t)$.*

Proof: Let $s, t \in L(G)$ be sequences of events such that $f(x_0, s) = x_S \in X_S$ and $f(x_0, t) = x_{NS} \in X \setminus X_S$. Let us also assume that $P^{\Sigma}_{\Sigma_k}(s) = P^{\Sigma}_{\Sigma_k}(t)$. Let us call sequences s and t, ambiguous sequences. Then, according to Lemma 3.1, there is a sequence $v \in L(G_V)$, associated with the ambiguous sequences s and t, such that $f_V((x_0, x_0), v) = (x_S, x_{NS})$. Thus, the condition of line 5 of Algorithm 2 is false for x_S. If for all $x_S \in X_S$ there exist ambiguous sequences s and t, then the result of the algorithm is $D = False$.

Let us consider that for all $s \in L(G)$ such that $f(x_0, s) = x_S \in X_S$, there does not exist a sequence $t \in L(G)$ such that $f(x_0, t) \in X \setminus X_S$ and $P^{\Sigma}_{\Sigma_k}(s) = P^{\Sigma}_{\Sigma_k}(t)$. In this case, according to Lemma 3.1, there does not exist a sequence $v \in L(G_V)$ such that $f_V((x_0, x_0), v) = (x_S, x_{NS})$ for any $x_{NS} \in X \setminus X_S$. Thus, according to lines 4 to 7 of Algorithm 2, $D = True$. □

Example 3.4 *Let us consider again the system G from Example 3.1, depicted in Figure 3.5a, where $X = \{1, 2, 3, 4, 5, 6\}$, $\Sigma = \{a, b, c, d, e, f\}$, $\Sigma_k = \{c, d\}$, and $X_S = \{1, 3\}$. Let us verify the sufficient condition of Theorem 3.1 by following the steps of Algorithm 2. In line 1, automaton G_γ, depicted in Figure 3.5b, is computed by renaming the events from $\Sigma \setminus \{c, d\}$. Then, in line 2, $G_V = G \| G_\gamma$ is computed, depicted in Figure 3.6. Finally, in lines 4 to 8 it is verified that, for $x_1 = 3$, the only state $x_V = (x_1, x_2) \in X_V$ where $x_1 = 3$ is the state $(3, 3)$. Thus, there does not exist a state $(x_1, x_2) \in X_V$ such that $x_2 \in X \setminus X_S$ for $x_1 = 3$ and $D = True$. Then, according to Theorem 3.3, for all $s \in L(G)$ satisfying $f(x_0, s) = 3$, there does not exist a sequence $t \in L(G)$ such that $f(x_0, t) \in X \setminus X_S$ and $P^{\Sigma}_{\Sigma_k}(s) = P^{\Sigma}_{\Sigma_k}(t)$. Consequently, according to Theorem 3.2, G is not CSO with respect to X_S and $\{c, d\}$.* □



(a) System G. (b) G_γ, with events from $\Sigma \setminus \{c, d\}$ renamed.

Figure 3.5: Automata G and G_γ for Example 3.4.

Theorem 3.3 shows that the verification of the sufficient condition for non-opacity presented in Theorem 3.2 can be carried out using Algorithm 2. Note that since Algorithm 2 has complexity $O(|X|^2 \times |\Sigma|)$, then it can be used to mitigate the computational complexity of verifying CSO with respect to the observation of a subset $\Sigma_k \subset \Sigma$, *i.e.*, Algorithm 2 can be used first to verify if G is not CSO with respect to Σ_k, and if $D = False$, then the observer $G_k = Obs(G, \Sigma_k)$ must be

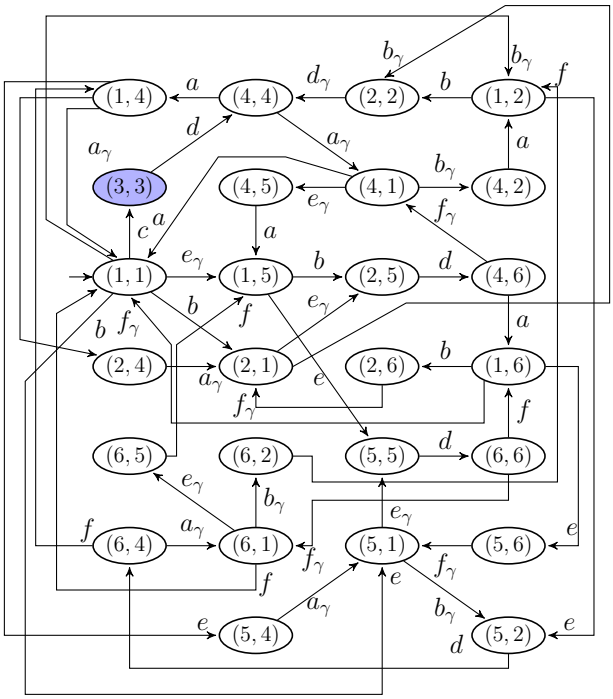


Figure 3.6: Automaton $G_V = G \| G_\gamma$ of Example 3.4, where $\Sigma_k = \{c, d\}$. The unique state which has the secret state 3 as a coordinate is state $(3, 3)$, represented in blue.

computed to verify if G is CSO, since it may be the case where G is not CSO with respect to Σ_k but Algorithm 2 still returns D = False.

After computing all MSO for the system, it is necessary to find a partition of Σ with smallest number of PS to define the number of isolated subnets of the network, and which events will be transmitted through each subnet. This problem is a set covering problem, stated as follows [78].

The set covering problem: Given a set of elements S and a collection of subsets of S, S_i, where $i = 1, \ldots, m$, referred to as $C = \{S_1, S_2, \ldots, S_m\}$, the set covering problem is to identify a smallest sub-collection of S whose union equals S.

The set covering problem is known to be NP-hard [78]. Algorithms to solve the set covering problem are presented in [84]. In the following, we present an example of the solution of the OSN-CSO problem, by first computing all MSO.

Example 3.5 *Let us consider again the system G from Example 3.1, depicted in Figure 3.5a, where $X = \{1, 2, 3, 4, 5, 6\}$, $\Sigma = \{a, b, c, d, e, f\}$, and $X_S = \{1, 3\}$. In order to find a solution to the OSN-CSO problem, let us first compute all MSO.*

To do so, the first step is the verification of CSO of G with respect to X_S and each $\Sigma_k \in 2^\Sigma \setminus \{\emptyset, \Sigma\}$. Then, the maximal ones are selected as the MSO. Starting with $\Sigma_k = \{a\}$, we first verify, using Algorithm 2, whether the sufficient condition of Theorem 3.2 is true for $\Sigma_k = \{a\}$. Since the verifier G_V, obtained considering as the observable event set $\Sigma_k = \{a\}$, has states $x_v = (x_1, x_2) \in X_V$ such that $x_1 \in X_S$

and $x_2 \in X_{NS}$, then the condition of Theorem 3.2 is false and it is necessary to compute the observer automaton $G_k = Obs(G, \{a\})$, depicted in Figure 3.7. Since there does not exist in G_k a state $x \in X_k$ such that $x \subseteq 2^{X_S}$, then G is CSO with respect to X_S and $P^{\Sigma}_{\Sigma_k}$.

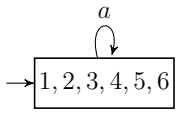


Figure 3.7: Observer automata $G_k = Obs(G, \{a\})$ for Example 3.5.

This procedure is repeated for each singleton subset $\{b\}$, $\{c\}$, $\{d\}$, $\{e\}$, and $\{f\}$. In all cases the system is current-state opaque. Then, we need to verify if G is CSO with respect to X_S and subsets of events with cardinality equal to two. We recall from Example 3.4 that the subset $\Sigma_k = \{c, d\}$ results in G being not CSO with respect to X_S and $\{c, d\}$. Consequently, according to Theorem 3.1, G is not CSO with respect to X_S and all subsets $\Sigma'_k \supset \{c, d\}$, and as presented in Example 3.3, the verification of CSO with respect to 15 subsets of Σ is avoided.

Continuing with the search for all subsets of Σ that leads to the CSO of G, we obtain the following MSO: $\{a, b, c, f\}$; $\{a, b, d, f\}$; $\{a, b, e, f\}$; $\{a, c, e, f\}$; and $\{a, d, e, f\}$. It is important to remark that only 41 of 62 observers were needed to be computed in this case due to the property presented in Theorem 3.1 and the sufficient condition presented in Theorem 3.2. This shows the potential reduction of computational cost that can be obtained using Theorems 3.1 and 3.2.

After computing all MSO, a solution to the OSN-CSO problem can be computed solving the set covering problem using any algorithm proposed in the literature. In this work, since we have only five MSO, we have used the brute force method to find a solution to the OSN-CSO problem. In this case, a possible partition is $\Sigma = \{a, b, c\} \dot{\cup} \{d, e, f\}$, which means that we can divide the network into two subnets for event transmission ensuring CSO. □

Since the computational cost of solving the set covering problem can be very high, several approximate algorithms are proposed in the literature. In the next section, we propose an approximate algorithm that provides a sub-optimal solution to the SN-CSO problem.

3.3 A sub-optimal solution to the SN-CSO problem

For complex and large systems with a high number of states, the computation of the optimal solution to the SN-CSO problem can have a high computational cost.

Thus, in this section, we present an approximate solution that avoids the exponential complexity of solving the set covering problem. The method is presented in Algorithm 3.

Algorithm 3 is a greedy algorithm that computes event sets that ensure opacity by first adding events to the first partition set Σ_1, maintaining the CSO with respect to Σ_1. When it is no longer possible to add events to Σ_1, that is, when Σ_1 becomes an MSO, the second partition set Σ_2 is formed by searching in the remaining event set Σ_c, formed in line 14, for those that ensure CSO with respect to Σ_2. The method continues forming partition sets until all events of Σ have been added to one of the PS. Note that in this method only set Σ_1 is necessarily an MSO.

It is important to remark that an advantage of the method proposed in Algorithm 3 is that it is not necessary to compute all MSO to find a solution. This reduces significantly the computational cost of finding a partition of the event set that solves the SN-CSO problem.

Algorithm 3: Computation of a suboptimal partition of the set of events.

Input : G.
Output: Partition sets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$.

```
1  i ← 1;
2  Σ_c ← Σ;
3  Σ_n ← ∅;
4  while Σ_c ≠ ∅ do
5  |  Σ_i ← ∅;
6  |  for each σ ∈ Σ_c do
7  |  |  if G is CSO with respect to P^Σ_{Σ_i∪{σ}} : Σ* → (Σ_i ∪ {σ})* then
8  |  |  |  Σ_i ← Σ_i ∪ {σ};
9  |  |  else
10 |  |  |  Σ_n ← Σ_n ∪ {σ};
11 |  |  end
12 |  |  Σ_c ← Σ_c \ {σ};
13 |  end
14 |  Σ_c ← Σ_n;
15 |  Σ_n ← ∅;
16 |  i ← i + 1
17 end
```

Theorem 3.4 *The computational complexity of Algorithm 3 is* $O(|\Sigma|^2 \times 2^{|X|})$.

Proof: Note that the verification of line 7 of Algorithm 3 is carried out necessarily for $P^{\Sigma}_{\{\sigma\}}$, for all $\sigma \in \Sigma$. Thus, $|\Sigma|$ CSO tests are carried out. Then, according to lines 7 and 8, if G is CSO with respect to $P^{\Sigma}_{\Sigma_i \cup \{\sigma\}}$, Σ_i is updated as $\Sigma_i \leftarrow \Sigma_i \cup \{\sigma\}$. Thus, in the worst-case, the test of line 7 is carried out at most $\frac{|\Sigma| \times (|\Sigma|-1)}{2}$ times.

Since the verification of line 7 has computational complexity $2^{|X|}$, then Algorithm 3 is $O(|\Sigma|^2 \times 2^{|X|})$. □

The number of subsets of Σ that are tested in Algorithm 3 is $O(|\Sigma|^2)$, which is much smaller than the methods proposed in the literature to find an optimal solution to the set covering problem. Note that the verification of line 7 is still exponential with respect to the number of system states since the verification of CSO is PSPACE-complete. In this case, the result of Theorem 3.2 can be used to mitigate the computational cost of Algorithm 3.

Example 3.6 *Let us consider again the system G from Example 3.1, depicted in Figure 3.5a, where $X = \{1, 2, 3, 4, 5, 6\}$, $\Sigma = \{a, b, c, d, e, f\}$, and $X_S = \{1, 3\}$. Let us compute a sub-optimal solution to the SN-CSO problem using Algorithm 3.*

In line 1 of Algorithm 3, the counter i is initialized as 1. In lines 2 and 3, the auxiliary sets Σ_c and Σ_n are initialized as Σ and $\emptyset$, respectively. Then, iteratively, in lines 4 to 17, the PS are formed.

In line 6, it is verified if G is CSO with respect to X_S and $P^{\Sigma}_{\Sigma_1 \cup \{\sigma\}}$, for each event $\sigma \in \Sigma_c$, starting with $\Sigma_1 = \emptyset$. Let $\sigma = a$. Then, Algorithm 2 is used to verify if G is CSO with respect to X_S and $P^{\Sigma}_{\{a\}} : \Sigma^* \to \{a\}^*$. As presented in Example 3.5, the condition of Algorithm 2 is not satisfied and the observer must be computed to verify if G is CSO with respect to X_S and $P^{\Sigma}_{\{a\}}$. After computing the observer, it can be concluded that G is CSO with respect to X_S and $P^{\Sigma}_{\{a\}}$. Then, in line 8, $\Sigma_1 \leftarrow \{a\}$ and in line 12, $\Sigma_c \leftarrow \{b, c, d, e, f\}$.

Let us consider that the next arbitrarily selected event in line 6 is $\sigma = b$. Then, in line 7, it is verified if G is CSO with respect to $P^{\Sigma}_{\{a,b\}} : \Sigma^* \to \{a, b\}^*$ with the help of Algorithm 2, which returns $D = False$, and so, the observer automaton $Obs(G, \{a, b\})$ must be computed. The observer automaton, depicted in Figure 3.3a, shows that G is CSO with respect to $P^{\Sigma}_{\{a,b\}}$. Thus, in line 8, $\Sigma_1 \leftarrow \{a, b\}$, and, in line 12, $\Sigma_c \leftarrow \{c, d, e, f\}$.

The procedure continues until the first partition set $\Sigma_1 = \{a, b, c, f\}$ is obtained. Note that, according to the observer presented in Figure 3.8, the system is CSO with respect to X_S and P^{Σ}_1 since it is not possible to infer when the system visits a secret state. In Figures 3.9a and 3.9b, it is possible to note that the system is not CSO with respect to the subsets $\{a, b, c, d, f\}$ and $\{a, b, c, e, f\}$ since it is possible to estimate state 3 when observing the subset $\{a, b, c, d, f\}$ and state 1 when observing the subset $\{a, b, c, e, f\}$, Thus, Σ_1 is an MSO, which means that if any event is added to Σ_1, then the current-state opacity is violated.

According to Algorithm 3, the next partition set Σ_2 is constructed from the remaining event set $\Sigma_c = \{d, e\}$, computed in line 14. By following the steps of Algorithm 3, we then verify if the system is CSO with respect to $P^{\Sigma}_{\{d,e\}} : \Sigma^* \to \{d, e\}^*$ by building the observer $Obs(G, \Sigma_2)$, depicted in Figure 3.10, where no secret state can

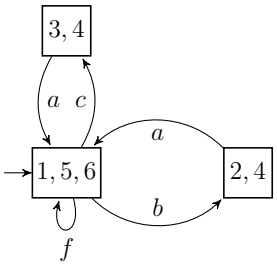


Figure 3.8: Observer automaton $Obs(G, \Sigma_1)$.

(a) $Obs(G, \{a, b, c, d, f\})$. (b) $Obs(G, \{a, b, c, e, f\})$.

Figure 3.9: Observers considering sets $\{a, b, c, d, f\}$ (a) and $\{a, b, c, e, f\}$ (b).

be inferred by observing events d and e. Thus, $\Sigma_2 = \{d, e\}$ *and since* Σ_c *is empty, meaning that events have been assigned to some partition set, the Algorithm 3 stops and returns partition sets* $\Sigma_1 = \{a, b, c, f\}$ *and* $\Sigma_2 = \{d, e\}$.

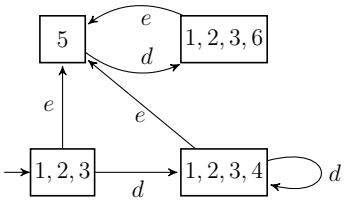


Figure 3.10: $Obs(G, \Sigma_2)$.

It is important to remark that only 7 observers have been computed in this example to obtain a solution to the SN-CSO problem, which is a much smaller number than the 41 observers computed in Example 3.5 to find a solution to the OSN-CSO problem. In addition, the solution obtained using Algorithm 3 also has two PS, which means that, in this example, we have obtained an optimal solution to the problem using Algorithm 3. Another important observation is that we have used Algorithm 2 to verify the condition of Theorem 3.2, which, in this case, has avoided the computation of one observer automaton to verify current-state opacity. □

3.4 Concluding remarks

In this chapter, we have presented a new strategy to enforce current-state opacity (CSO) while transmitting all events to the intended receiver without the need for encrypting them. The method is based on the segmentation of the network into isolated sub-networks, which limits lateral movements of the attacker, and choosing the events that are transmitted through each subnet such that the system is current-state opaque with respect to the observation of the events of each subnet in isolation. We consider that the attacker is capable of invading one subnet at a time. We show that the partitioning of the event set to satisfy the segmented network current-state opacity (SN-CSO) problem has a high computational cost, since the verification of current-state opacity is PSPACE-complete and the search for a partition with smallest number of sets is NP-complete. Thus, we propose in this chapter, methods to avoid the computation of observers to verify CSO, and a greedy algorithm to find a sub-optimal solution to the SN-CSO problem.

In this chapter we considered that the attacker is capable of invading one subnet at a time. However, it is possible that an attacker is capable of invading more than one subnet. Thus, in next chapter, we present a method to protect the secret information in a segmented network that can be used to defend the system for an attacker that attacks one or more than one vulnerable subnets.

Chapter 4

Enforcing current-state opacity using multipath event routing

In this chapter, we present a method to enforce current-state opacity (CSO) and current-state utility (CSU) of cyber-physical systems abstracted as networked discrete events systems (NDES) using event replacement with a strategy called multipath event routing. The multipath event routing is based on cryptography and routing control that leverages a segmented network architecture, in which events are routed and communicated to the receiver through different subnets. We assume that the NDES communicates with an external observer through different subnets, and that some of these subnets are vulnerable to eavesdropping attacks. We consider that the secret information is modeled by secret states of the plant, and the attacker tries to infer that a secret state has been reached by observing only the events transmitted in the attacked subnets. We also consider that the attacker knows the plant model represented by an automaton, and that, differently from Chapter 3 where only one subnet is attacked at a time, all vulnerable subnets are attacked at the same time. The external observer considered in this work is any entity that receives data from the plant through a communication network, such as a supervisory control and data acquisition (SCADA) system, or a diagnoser.

4.1 System architecture and attacker capabilities

We consider in this chapter that there is a unique intended receiver, as shown in Figure 4.1, that receives the communication of event occurrences through a segmented network. The segmentation of the network can be physical or logical, resulting in a set of isolated and independent sub-networks, or subnets. In addition, we consider that some of these subnets can be vulnerable to attacks. In these cases, each data transmission path is predetermined between sender and receiver.

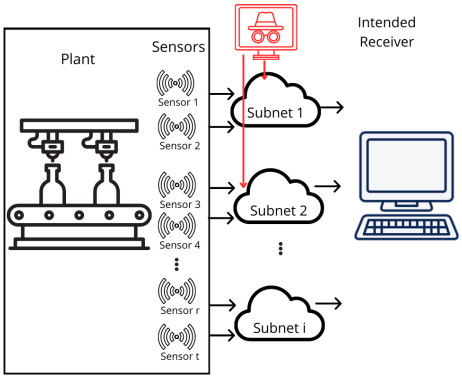


Figure 4.1: Communication between the plant and intended receiver, where two vulnerable subnets are attacked.

Let $G = (X, \Sigma, f, x_0)$ be the plant automaton, and consider that the plant events are transmitted to an external receiver through their own subnets SN_i, as depicted in Figure 4.1. Consider that the set of subnets indexes is denoted by $I = \{1, 2, \ldots, n\}$. Then, each subnet SN_i, for $i \in I$, is used for the transmission of events in the set $\Sigma_i \subset \Sigma$, where $\Sigma_i \cap \Sigma_j = \emptyset$, for $i \neq j$, such that $\Sigma = \cup_{i=1}^{n} \Sigma_i$. Thus, all events of Σ are observable by those who receive the information from SN_i, $i \in I$. Since the intended receiver gets information from all subnets, then all events are observable for the intended receiver. Assume also that the observation of events is not subject to delays or losses.

Let us consider that part of the subnets is vulnerable to eavesdropping attacks, and let $I_\alpha \subset I$ be the set of vulnerable subnets. Hence, the attacker is capable of eavesdropping on all vulnerable subnets SN_i, for $i \in I_\alpha$, at the same time. Then, after the execution of a sequence of events $s \in L(G)$, the attacker observes $P_{\Sigma_\alpha}^{\Sigma}(s)$, where $P_{\Sigma_\alpha}^{\Sigma} : \Sigma^* \to \Sigma_\alpha^*$ is a projection and $\Sigma_\alpha = \cup_{i \in I_\alpha} \Sigma_i$. In this case, if the system is not CSO with respect to X_S and projection $P_{\Sigma_\alpha}^{\Sigma}$, the secret can be inferred by the attacker observing only the events in the vulnerable subnets. Thus, in order to propose a defense mechanism in this paper, the following assumptions can be made regarding the attacker's knowledge and capability:

A1. The attacker may invade the system at any time, and does not know the current state of the plant when he/she invades the system;

A2. The attacker has partial observation of the system, observing only the events in the attacked subnets;

A3. The attacker knows the complete system model and the communication archi-

Figure 4.2: Naval fleet composed of an anti-submarine, an antiaircraft and a general purpose ship where an enemy submarine eavesdrops the communication between the anti-submarine and the command center.

tecture;

A4. The attacker and intended receiver obtain the same information from the observed messages communicated through the attacked subnets.

According to Assumptions **A1**-**A4**, the attacker can construct the current-state estimator of the system modeled by automaton $\mathcal{G} = (X, \Sigma, f, X)$, obtaining the deterministic automaton $G_\alpha = Obs(\mathcal{G}, \Sigma_\alpha) = (X_\alpha, \Sigma_\alpha, f_\alpha, x_{0,\alpha})$, where each state of X_α is composed of the state estimate of X after the observation of a sequence of events in $L(G_\alpha) \subseteq \Sigma_\alpha^*$. Based on the state estimator G_α, the attacker can infer if the system has reached a secret state of the system if, after the observation of $s \in L(G_\alpha)$, $f_\alpha(x_{0,\alpha}, s) \in 2^{X_S}$. Thus, the security mechanism must be capable of avoiding the intruder to be certain that a secret state has been reached when the system actually reaches a state $x \in X_S$. In the following, we present a practical example to motivate the necessity to protect the vulnerable subnets.

Example 4.1 *Let us suppose a naval fleet, composed of three ships, depicted in Figure 4.2, where each ship has a different role: an anti-submarine; an antiaircraft; and a general purpose ship. These ships communicate with each other by short range radio, and with a command center by long range radio. Consider that there exists an enemy submarine that is interested in attacking the naval fleet. It is important for the submarine to know if the anti-submarine ship is ready to engage or out of service, and this information is communicated to the command center only by the anti-submarine ship. Thus, since the useful information for the attacker is only communicated by the anti-submarine, then the submarine tries to invade the communication between the anti-submarine ship and the command center.*

In this scenario, the automaton that models the status of each ship for the command center is represented by automaton G_i, depicted in Figure 4.3. State N_i represents that ship i is in normal operation, while state B_i represents that ship i is out

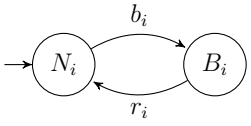


Figure 4.3: Automaton that represents the status of ship i.

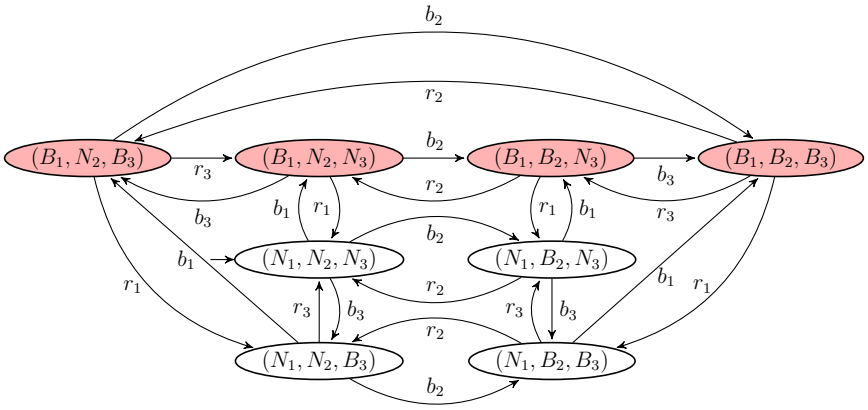


Figure 4.4: Automaton G representing the naval fleet from the external observer perspective. The red states are the secret states.

of service due to fire or flood in the machine room, for example. Additionally, event b_i represents that ship i is informing a breakdown to the command center, and event r_i represents that ship i has been repaired. The automaton that represents the model of the complete system is presented in Figure 4.4.

The event set of the plant is obtained as $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$, where $\Sigma_1 = \{b_1, r_1\}$, $\Sigma_2 = \{b_2, r_2\}$, and $\Sigma_3 = \{b_3, r_3\}$. Thus, $\Sigma = \{b_1, r_1, b_2, r_2, b_3, r_3\}$. As the attacker is interested in discovering if the anti-submarine ship is ready to engage or out of service and knows the system model, he/she will invade the subnet 1, SN_1, that communicates the events of the anti-submarine ship to the command center. Thus, $\Sigma_\alpha = \Sigma_1 = \{b_1, r_1\}$.

The secret states are, in this case, the states of the system associated with the submarine ship out of service, i.e., the set of secret states is $X_S = \{(B_1, N_2, N_3), (B_1, B_2, N_3), (B_1, N_2, B_3), (B_1, B_2, B_3)\}$. The state estimator, G_α, constructed by the attacker is presented in Figure 4.5, where the states of G are renamed as presented in Table 4.1. Note that after observing event b_1, the aataker is capable to infer that the system is in a secret state, i.e., the submarine is certain that the anti-submarine ship is broke down and can engage the fleet. □

In the communication architecture proposed in this work, the intended receiver observes all events communicated through subnets SN_i, for $i = 1, 2, \ldots, n$, and we consider that the useful information for the intended receiver is the knowledge

Table 4.1: States of plant G.

State	New label
(N_1, N_2, N_3)	0
(B_1, N_2, N_3)	1
(B_1, B_2, B_3)	2
(N_1, B_2, B_3)	3
(N_1, B_2, N_3)	4
(B_1, B_2, N_3)	5
(N_1, N_2, B_3)	6
(B_1, N_2, B_3)	7

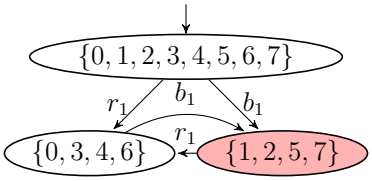


Figure 4.5: State estimator G_α that represents the system observation from the attacker's perspective. The red state represents the state estimate composed of secret states.

of the system current state after the observation of any sequence of events. This notion of utility is called, in this work, current-state utility (CSU). We propose the implementation of a security policy that ensures CSO and CSU. The security policy replaces an event $\sigma \in \Sigma_j$ that has occurred in the system with an event $\tilde{\sigma} \in \Sigma_i$, and immediately reports $\tilde{\sigma}$ to the intended receiver through subnet SN_i, where i is not necessarily different from j. We call this technique multipath event routing. It is important to remark that, as each event is transmitted through a unique subnet, the output of the obfuscation policy is only the reported event $\tilde{\sigma}$, *i.e.*, the subnet used for transmission is implicitly defined.

In this work, we consider that the attacker knows the system model, and does not know the time interval between the occurrence of events in the system. However, in practice, a relatively long period of time without communicating any event through the attacked subnets may lead the attacker to deduce that a security mechanism is being used. Let us consider that after an upper bound on the time elapsed between the occurrence of events, b_i, the attacker knows that the attacked subnet SN_i has been silenced by a security mechanism. Thus, to avoid the attacker from detecting the use of a security mechanism, it must transmit an event through the attacked subnet SN_i, $i \in I_\alpha$, after a predefined number n_i of consecutive events of Σ_i have been reported to the external observer through other subnets, and if, in the meantime, an event of $\Sigma \setminus \Sigma_i$ has not been replaced with an event of Σ_i and reported in SN_i. Note that the maximum value for n_i depends on the system dynamics and

on b_i, and must be verified whether it is sufficient so that the attacker does not suspect that a security mechanism is being used.

4.2 Formulation of the problem of enforcing CSO and CSU

Let us now introduce the set of reported events of the plant to distinguish the successful transmission of an event to the intended receiver from its occurrence in the plant. The set of reported events is defined as $\Sigma_\rho = \{\sigma_\rho : \sigma \in \Sigma\}$. It is important to remark that event $\sigma_\rho \in \Sigma_\rho$ is observed by the attacker and receiver as the original event σ. The set of reported events in an attacked subnet SN_i is denoted by $\Sigma_{\alpha,\rho_i} \subseteq \Sigma_\rho$. Then, the set of all events reported through attacked subnets is defined as $\Sigma_{\alpha,\rho} = \cup_{i\in I_\alpha}\Sigma_{\alpha,\rho_i}$. The set formed of all events reported through non-attacked subnets is given by $\Sigma_{\bar{\alpha},\rho} = \Sigma_\rho \setminus \Sigma_{\alpha,\rho}$.

Let us define the projections $P^{\Sigma_\rho}_{\Sigma_{\alpha,\rho}} : \Sigma^*_\rho \to \Sigma^*_{\alpha,\rho}$, $P^{\Sigma_\rho}_{\Sigma_{\alpha,\rho_i}} : \Sigma^*_\rho \to \Sigma^*_{\alpha,\rho_i}$, and $P^{\Sigma}_{\Sigma_i} : \Sigma^* \to \Sigma^*_i$. Then, the state observer from the attacker's perspective, can be modeled by automaton $G_{\alpha,\rho} = Obs(G_\rho, \Sigma_{\alpha,\rho}) = (X_\alpha, \Sigma_{\alpha,\rho}, f_{\alpha,\rho}, x_{0,\alpha})$, where G_ρ is identical to G with all events $\sigma \in \Sigma$ replaced with its corresponding event $\sigma_\rho \in \Sigma_\rho$. Note that $L(G_{\alpha,\rho}) = P^{\Sigma_\rho}_{\Sigma_{\alpha,\rho}}(L(G_\rho))$. Thus, the problem of ensuring CSO, with respect to the set of attacked subnets SN_i, $i \in I_\alpha$, and X_S, while also ensuring CSU, can be formulated as follows.

Problem 1: Find function $f_{ER} : \Sigma^* \to \Sigma^*_\rho$, which represents the event replacement function, such that the following conditions are satisfied:

- **C1.** For all $s \in L(G)$ such that $f(x_0, s) \in X_S$, $f_{\alpha,\rho}(x_{0,\alpha}, P^{\Sigma_\rho}_{\Sigma_{\alpha,\rho}}(\tilde{s})) \notin 2^{X_S}$, where $\tilde{s} = f_{ER}(s)$.
- **C2.** For all $s \in L(G)$, $P^{\Sigma_\rho}_{\Sigma_{\alpha,\rho}}(\tilde{s}) \in L(G_{\alpha,\rho})$, where $\tilde{s} = f_{ER}(s)$.
- **C3.** For all $s \in L(G)$, there does not exist a sequence $t \in \Sigma^*$, where $s = utv$, for $u, v \in \Sigma^*$, such that $\|P^{\Sigma}_{\Sigma_i}(t)\| > n_i$, for $i \in I_\alpha$, and $f_{ER}(s) = \tilde{u}\tilde{t}\tilde{v}$, with $\|\tilde{u}\| = \|u\|$, $\|\tilde{t}\| = \|t\|$, and $P^{\Sigma_\rho}_{\Sigma_{\alpha,\rho_i}}(\tilde{t}) = \varepsilon$.
- **C4.** f_{ER} must be reversible, *i.e.*, $\forall s_1, s_2 \in L(G)$, $f_{ER}(s_1) = f_{ER}(s_2)$ if, and only if, $s_1 = s_2$.

If Condition **C1** is satisfied, then all sequences generated by the plant s, that lead the system to a secret state $x \in X_S$, are transmitted as $\tilde{s}$ such that the attacker does not estimate that the system is in a secret state based on his/her observation of $\tilde{s}$, $P^{\Sigma_\rho}_{\Sigma_{\alpha,\rho}}(\tilde{s})$. Condition **C2** ensures that the observations by the attacker of all transmitted sequences $\tilde{s}$, $P^{\Sigma_\rho}_{\Sigma_{\alpha,\rho}}(\tilde{s})$, are feasible according to the state observer of

Figure 4.6: (a) Automaton G representing the system. (b) State estimator G_α from the attacker's perspective.

G_ρ, $G_{\alpha,\rho}$. Conditions **C1** and **C2** guarantee that CSO is ensured considering the events transmitted in the vulnerable subnets as the observable events. Condition **C3** ensures that there does not exist a subsequence t of a generated sequence s, such that the projection of t in an attacked subnet SN_i, $P^{\Sigma}_{\Sigma_i}(t)$, has more than n_i events and none of the events of t are transmitted through subnet SN_i, *i.e.*, $P^{\Sigma_\rho}_{\Sigma_{\alpha,\rho_i}}(\tilde{t}) = \varepsilon$. Thus, Condition **C3** guarantees that subnet SN_i must be used to transmit an event to the receiver before n_i event occurrences of Σ_i without communicating any event through SN_i, guaranteeing that the attacker does not suspect that a defense mechanism is being used by the absence of event transmissions. Condition **C4** ensures the CSU of the transmitted data for the intended receiver, which uses the inverse of f_{ER} to decipher the transmitted message.

Note that, according to the multipath event routing strategy, the transmission of events is carried out event by event, and an event is immediately transmitted to the intended receiver after the occurrence of a plant event. This implies that if $f_{ER}(u) = \tilde{u}$, where $u \in L(G)$, then for all $s = uv \in L(G)$, where $v \in \Sigma^*$, $\tilde{u} \in Pr(f_{ER}(uv))$.

Example 4.2 *Let G, depicted in Figure 4.6a, be the plant model, where $X = \{1, 2, 3, 4\}$ and $\Sigma = \{a, b, c, d\}$. In addition, let $X_S = \{4\}$ be the set of secret states (marked in red in Figure 4.6a). Let us also suppose that two different subnets, SN_1 and SN_2, communicate the occurrence of events to the external observer. The events are separated according to the transmission subnets as $\Sigma_1 = \{a, d\}$ and $\Sigma_2 = \{b, c\}$.*

Let us suppose that subnet SN_1 is vulnerable to eavesdropping attacks. In this case, $\Sigma_\alpha = \Sigma_1$. Thus, according to Assumptions **A1**-**A4**, *the attacker can construct the state estimator of the system $G_\alpha = Obs(\mathcal{G}, \Sigma_\alpha) = (X_\alpha, \Sigma_\alpha, f_\alpha, x_{0,\alpha})$, depicted in Figure 4.6b, computed considering that any state of G can be the initial state of the system. Note that G_α represents the state estimate of the system by the intruder after observing a sequence of events in Σ^*_α, communicated through the attacked subnet SN_1. In this case, as depicted in Figure 4.6b, the attacker is certain that the system is in a secret state when he/she observes event d, and, consequently, estimates state* 4. □

Example 4.2 presents a case where an attacker invades one subnet of the seg-

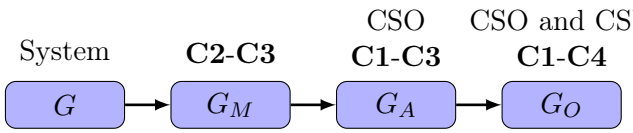


Figure 4.7: Flowchart for the synthesis of automaton G_O from which all f_{ER} that satisfy Conditions **C1-C4** can be extracted.

mented network and successfully discover the secret information. In next section, we present a method to protect the secret information by enforcing current-state opacity using event replacement.

4.3 Enforcing current-state opacity using event replacements

In REIS *et al.* [55], we propose a method to enforce CSO using event replacements that does not consider the utility of the information for the intended receiver. Thus, the obfuscation policy presented in [55] do not pursue reversibility (Condition **C4**). Thus, we propose a modification in the method proposed in [55] to compute a function f_{ER} which satisfies all Conditions **C1-C4**.

The method proposed in this work consists of obtaining an automaton G_O from which a function f_{ER} that satisfies all Conditions **C1-C4** can be extracted, as depicted in the flowchart of Figure 4.7. To compute G_O, it is first necessary to compute, from G, the automaton G_M in which all possible f_{ER} that satisfy Conditions **C2** and **C3** are represented. Then, automaton G_A, is obtained from G_M with a view to satisfying also Condition **C1**. Finally, automaton G_O, in which all f_{ER} that satisfy Conditions **C1-C4** are represented, is obtained from G_A.

In [55], to obtain automaton G_M, three automata are introduced: (i) G_r, that represents how each plant event $\sigma \in \Sigma_i$ can be reported to the receiver depending on the vulnerability of subnet SN_i; (ii) $G_{r,a}$, that models the system state estimator from the attacker's perspective; and (iii) G_b, that models the restriction on the obfuscation policy to avoid the silencing of the attacked subnet SN_i for more than a predefined value n_i, for $i \in I_\alpha$. Then, $G_M = G \| G_r \| G_{r,a} \| G_b$ is computed.

The first modification proposed in this work with respect to the method presented in [55] is in automaton G_r. According to Condition **C4** of Problem 1, the intended receiver must be capable of identifying the current system state after the observation of any obfuscated sequence of events. Thus, differently from [55], it may be necessary to communicate the occurrence of an event associated with a non-attacked subnet as any other event to the intended receiver. To do so, after the occurrence of a plant event $\sigma \in \Sigma$, any event $e \in \Sigma_\rho$ can be reported. This behavior is modeled by the

interleaver automaton $G_\iota = (X_\iota, \Sigma \cup \Sigma_\rho, f_\iota, x_{0,\iota})$, computed according to Algorithm 4.

In line 1 of Algorithm 4, the initial state $x_{0,\iota}$ is created, and in line 2, the set of states X_ι is initialized with $x_{0,\iota}$. In lines 3 to 8, for each event $\sigma \in \Sigma$, a new state x_σ is created and added to the set of states X_ι. In addition, a transition from the initial state $x_{0,\iota}$ to the new state x_σ is created labeled with event σ, and a transition from state x_σ to $x_{0,\iota}$ is created labeled with all events $e \in \Sigma_\rho$. In the sequel, we present the computation of the interleaver automaton G_ι, for the plant of Example 4.2.

Algorithm 4: Computation of G_ι

Input : G, Σ_ρ.
Output: $G_\iota = (X_\iota, \Sigma \cup \Sigma_\rho, f_\iota, x_{0,\iota})$.

```
1 Create an initial state x_{0,ι};
2 X_ι = {x_{0,ι}};
3 for each σ ∈ Σ do
4 |   Create a new state x_σ;
5 |   X_ι ← X_ι ∪ {x_σ};
6 |   Define f_ι(x_{0,ι}, σ) = x_σ;
7 |   Define f_ι(x_σ, e) = x_{0,ι}, ∀e ∈ Σ_ρ;
8 end
```

Example 4.3 *Let us consider the CPS presented in Example 4.2, where the event set of the plant is* $\Sigma = \{a, b, c, d\}$*, and the set of reported events is* $\Sigma_\rho = \{a_\rho, b_\rho, c_\rho, d_\rho\}$*. Following the steps of Algorithm 4, the initial state* $x_{0,\iota}$ *is created in line 1 and the set of states* X_ι *is initialized as* $\{x_{0,\iota}\}$ *in line 2. In the sequel, for each event* $\sigma \in \Sigma$*, a new state* x_σ *is created.*

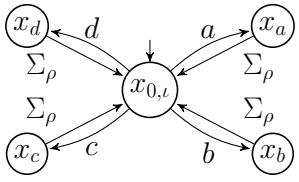


Figure 4.8: Automaton G_ι

Starting with $\sigma = a$*, in line 4, state* x_a *is created, and added to* X_ι *in line 5. In line 6, a transition from* $x_{0,\iota}$ *to* x_a*, labeled with event* a*, is created. Then, a transition from* x_a *to* $x_{0,\iota}$ *is created labeled with each event in* Σ_ρ *in line 7. Algorithm 4 continues until automaton* G_ι *of Figure 4.8 is computed.* □

In this work, differently from the method proposed in [55], it is considered that if an event that should be transmitted on a non-attacked subnet, is transmitted through an attacked subnet SN_i, that transmission is taken into account to consider

that the silencing of SN_i has been broken. This is formalized in Condition **C3** of Problem 1. In Algorithm 5, automaton G_{β_i} is computed to model the restriction on the event replacement function to ensure Condition **C3**, considering the attacked subnet SN_i. Then, $G_\beta = \|_{i \in I_\alpha} G_{\beta_i}$ is computed to take into account all vulnerable subnets together.

Algorithm 5: Computation of G_{β_i}.

Input : n_i, Σ_i, Σ_ρ, Σ_{α,ρ_i}
Output: $G_{\beta_i} = (X_{\beta_i}, \Sigma_i \cup \Sigma_\rho, f_{\beta_i}, x_{0,\beta_i})$

```
1  x_{0,β_i} ← (ε, 0);
2  c ← 0, X_{β_i} ← ∅;
3  while c ≤ n_i do
4  |  X_{β_i} ← X_{β_i} ∪ {(ε, c), (ν, c)};
5  |  for e ∈ Σ_ρ \ Σ_{α,ρ_i} do
6  |  |  f_{β_i}((ε, c), e) = (ε, c)
7  |  end
8  |  for e ∈ Σ_{α,ρ_i} do
9  |  |  f_{β_i}((ε, c), e) = (ε, 0)
10 |  end
11 |  for σ ∈ Σ_i do
12 |  |  f_{b_i}((ε, c), σ) = (ν, c)
13 |  end
14 |  for e ∈ Σ_{α,ρ_i} do
15 |  |  f_{β_i}((ν, c), e) = (ε, 0)
16 |  end
17 |  c ← c + 1;
18 |  if c − 2 ≥ 0 then
19 |  |  for e ∈ Σ_ρ \ Σ_{α,ρ_i} do
20 |  |  |  f_{β_i}((ν, c − 2), e) = (ε, c − 1)
21 |  |  end
22 |  end
23 end
```

In line 1 of Algorithm 5, the initial state x_{0,β_i} is defined as $(\varepsilon, 0)$. In line 2, the counter c is initialized with 0 and the set of states X_{β_i} is created as the empty set. Then, from lines 3 to 23, as long as counter c is smaller than or equal to n_i, new states and transitions of G_{β_i} are created. In line 4, a pair of states, (ε, c) and (ν, c), is created and added to the set of states X_{β_i}, where ν represents the occurrence of an event of Σ_i. From lines 5 to 7, a self-loop labeled with all events $e \in \Sigma_\rho \setminus \Sigma_{\alpha,\rho_i}$ is created in state (ε, c), and from lines 8 to 10, transitions are created from state (ε, c) to $(\varepsilon, 0)$ labeled with events of Σ_{α,ρ_i}, representing that an event that should be transmitted through any subnet SN_j, $j \neq i$, has been reported in the attacked subnet SN_i, avoiding that SN_i is kept silenced. From lines 11 to 13, a transition from (ε, c) to (ν, c) is created labeled with all events $\sigma \in \Sigma_i$, representing that an event

in the attacked subnet SN_i has occurred. Then, from lines 14 to 16, a transition from (ν, c) to $(\varepsilon, 0)$ is created labeled with all events $e \in \Sigma_{\alpha,\rho_i}$, representing that an event has been reported in SN_i. Then, in line 17, the counter c is incremented, and finally, in lines 18 to 22, if $c-2$ is greater than or equal to zero, a transition from $(\nu, c-2)$ to $(\varepsilon, c-1)$ is created labeled with all events $e \in \Sigma_\rho \setminus \Sigma_{\alpha,\rho_i}$, representing that an event in SN_i has been executed but reported through another subnet. In the following, we present an example of the computation of automaton $G_{\beta,1}$ for the system presented in Example 4.2 when $n_1 = 1$.

Example 4.4 *Consider again the CPS presented in Example 4.2, where the event set of the plant is $\Sigma = \{a, b, c, d\}$, and the attacked subnet is SN_1. Thus, $\Sigma_\alpha = \Sigma_1 = \{a, d\}$. In this example, $\Sigma_\rho = \{a_\rho, b_\rho, c_\rho, d_\rho\}$ and $\Sigma_{\alpha,\rho_1} = \{a_\rho, d_\rho\}$. Let $n_1 = 1$, i.e., after the occurrence of an event of Σ_1 reported through another channel, if no other event is reported using the attacked subnet SN_1, the next occurrence of an event of SN_1 must be reported through it.*

In order to compute G_{β_1}, the initial state $x_{0,\beta_1} = (\varepsilon, 0)$ is created in line 1 of Algorithm 5. In line 2, a counter c is initialized as 0, and the set of states X_{β_1} is initialized as the empty set. In line 3, as $c \leq 1$, a loop is started, and X_{β_1} is updated to $\{(\varepsilon, 0), (\nu, 0)\}$ in line 4. A self-loop is defined in state $(\varepsilon, 0)$ with each event in $\Sigma_\rho \setminus \Sigma_{\alpha,\rho_1}$ in lines 5 to 7, and from lines 8 to 10, self-loop transitions are created in state $(\varepsilon, 0)$ labeled with the events of Σ_{α,ρ_1}, representing that an event has been reported in SN_1. In addition, from line 11 to 13, in order to register that an event in SN_1 has occurred, transitions from $(\varepsilon, 0)$ to $(\nu, 0)$ are created labeled with events in Σ_1, i.e., a and d. Then, from line 14 to 16, transitions from $(\nu, 0)$ to $(\varepsilon, 0)$ are created labeled with events in Σ_{α,ρ_1}, i.e., a_ρ and d_ρ, representing that an event has been reported in the attacked subnet. In the sequel, in line 17, the counter c is incremented, and, as $c - 2 = -1 \leq 0$, the loop is restarted with $c = 1$.

Starting the new iteration with $c = 1$, the set of states X_{β_1} is updated to $\{(\varepsilon, 0), (\nu, 0), (\varepsilon, 1), (\nu, 1)\}$ in line 4. From line 5 to 7, a self-loop is defined in state $(\varepsilon, 1)$ labeled with each event in $\Sigma_\rho \setminus \Sigma_{\alpha,\rho_1}$, and from line 8 to 10, transitions are created from state $(\varepsilon, 1)$ to $(\varepsilon, 0)$ labeled with events from Σ_{α,ρ_1}, representing that an event has been reported in SN_1. From line 11 to 13, in order to register that an event in SN_1 has occurred, transitions from $(\varepsilon, 1)$ to $(\nu, 1)$ are created labeled with events in Σ_1, i.e., a and d. From line 14 to 16, transitions from $(\nu, 1)$ to $(\varepsilon, 0)$ are created labeled with events in Σ_{α,ρ_1}, i.e., a_ρ and d_ρ, representing that the event has been reported in the attacked subnet. Then, counter c is incremented to $c = 2$ in line 17, and, as $c - 2 = 0$, from line 19 to 21, transitions from $(\nu, 0)$ to $(\varepsilon, 1)$ are created labeled with each event in $\Sigma_\rho \setminus \Sigma_{\alpha,\rho_1} = \{b_\rho, c_\rho\}$.

Since, in this example, only one subnet is attacked, automaton G_β is equal to G_{β_1}, depicted in Figure 4.9.

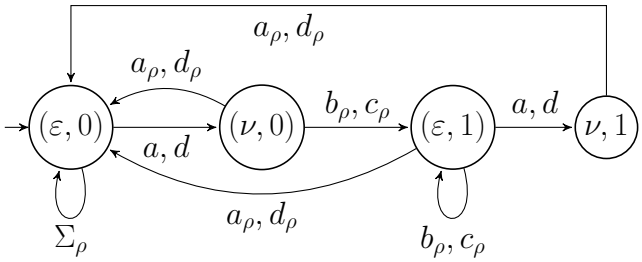


Figure 4.9: Automaton G_{β_1} for Example 4.5.

Note that state $(\varepsilon, 0)$ represents that any event of SN_1, a or d, has not occurred yet, or has occurred, and one of the events a_ρ or d_ρ has been reported. State $(\nu, 0)$ represents that an event of SN_1 has occurred but no event has been reported yet. State $(\varepsilon, 1)$ represents that an event in the attacked subnet SN_1 has occurred, and an event from a non-attacked subnet, SN_2 has been reported. Finally, state $(\nu, 1)$ represents that two events of SN_1 have occurred and no event has been reported through it. □

After the computation of G_ι and G_β, a new automaton $G_M = (X_M, \Sigma \cup \Sigma_\rho, f_M, x_{0,M}) = G \| G_\iota \| G_{\alpha,\rho} \| G_\beta$ is computed, where $G_{\alpha,\rho}$ is constructed considering the observation of the reported events in the attacked subnets and is equal to G_α, except that the events $\sigma \in \Sigma_\alpha$ are replaced with the corresponding events $\tilde{\sigma} \in \Sigma_{\alpha,\rho}$. Note that $P_\Sigma^{\Sigma \cup \Sigma_\rho}(L(G_M)) = L(G)$, where $P_\Sigma^{\Sigma \cup \Sigma_\rho} : (\Sigma \cup \Sigma_\rho)^* \to \Sigma^*$. In the following, we present an example where G_M is computed for the plant of Example 4.2.

Example 4.5 *Consider automaton G that models the behavior of the CPS presented in Example 4.2 with $X_S = \{4\}$, where the events are separated according to the subnets as $\Sigma_1 = \{a, d\}$ and $\Sigma_2 = \{b, c\}$. Let us consider that subnet SN_1 is vulnerable to eavesdropping attacks and let $n_1 = 1$.*

In order to compute automaton $G_M = G \| G_\iota \| G_{\alpha,\rho} \| G_\beta$, where $G_{\alpha,\rho}$, it is first necessary to compute automata G_ι, $G_{\alpha,\rho}$, and G_β. G_ι was computed in Example 4.3 and presented in Figure 4.8. $G_{\alpha,\rho} = Obs(G_\rho, \Sigma_{\alpha,\rho})$ is equal to G_α, presented in Figure 4.6b, replacing events a and d with a_ρ and d_ρ, respectively, and is depicted in Figure 4.10. Since the vulnerable subnet is only SN_1, $G_\beta = G_{\beta_1}$. G_β was computed in Example 4.4, and is depicted in Figure 4.9. In this example, G_M has 40 states and 79 transitions and is depicted in Figure 4.11, where its states have been renamed due to lack of space. □

In the sequel, we show how to extract an event replacement function f_{ER} from G_M by following the steps of Algorithm 6. To do so, let us first obtain a partition of the set of states of $X_M = X_{M,1} \cup X_{M,2}$. In line 1, create set $X_{M,1} = \{x_M \in X_M : (\exists x \in X, \exists x_\alpha \in X_\alpha, \exists x_\beta \in X_\beta)[x_M = (x, x_{0,\iota}, x_\alpha, x_\beta)]\}$, formed of all the

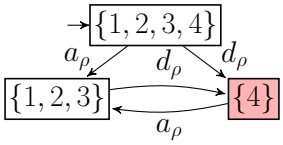


Figure 4.10: Automaton $G_{\alpha,\rho}$ for Example 4.5.

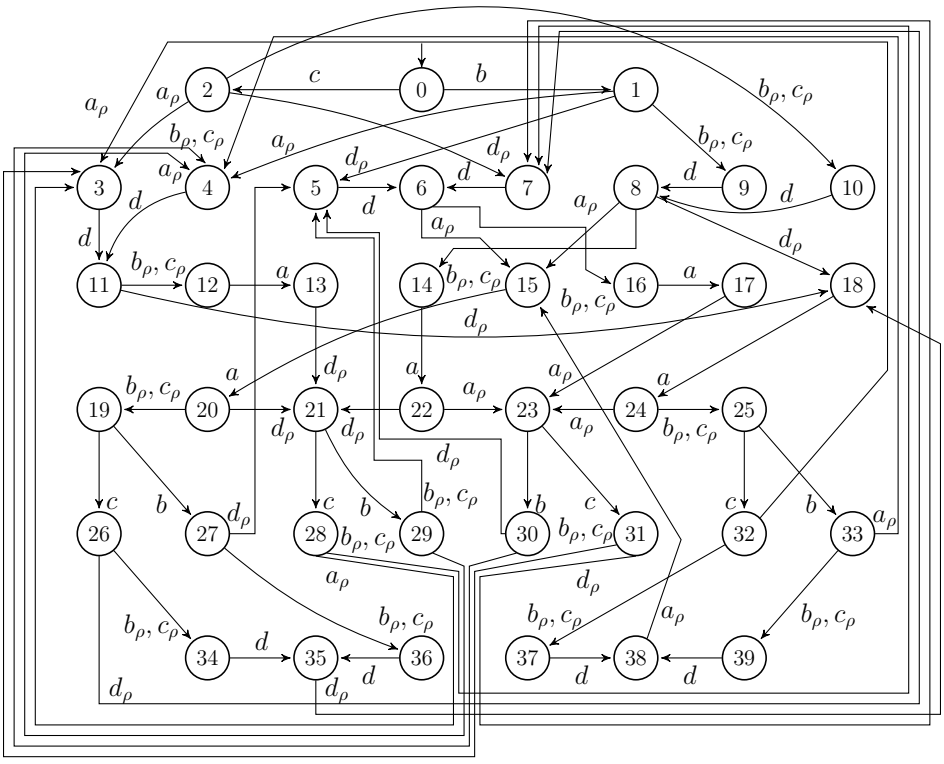


Figure 4.11: Automaton G_M of Example 4.5.

states such that all feasible events are plant events in Σ. In line 2, create set $X_{M,2} = \{x_M \in X_M : (\exists x \in X, \exists x_\alpha \in X_\alpha, \exists x_\beta \in X_\beta)[x_M = (x, x_\sigma, x_\alpha, x_\beta)]\}$, where all feasible events are in Σ_ρ. Thus, after the occurrence of a plant event $\sigma \in \Sigma$, from a state $x_M \in X_{M,1}$, a state $x'_M \in X_{M,2}$ is reached, and after the occurrence of an event $e \in \Gamma_{G_M}(x'_M)$, which represents how event σ is reported to the external observer, a state in $X_{M,1}$ is reached. Then, an event replacement function f_{ER} can be extracted from G_M by erasing transitions labeled with reported events in Σ_ρ from states in $X_{M,2}$, obtaining a new automaton G_N, such that $|\Gamma_{G_N}(x_M)| = 1$, for all $x_M \in X_{M,2}$ in line 3. Then, in line 4, taking the accessible part of G_N, we obtain an automaton $G_P = (X_P, \Sigma \cup \Sigma_\rho, f_P, x_{0,P})$, which represents the chosen event replacement function f_{ER}. Function f_{ER} is extracted from G_P, as follows: for all $x \in X_P \cap X_{M,1}$, find $s_P \in L(G_P)$ such that $x = f_P(x_{0,P}, s_P)$. Let $s = P_\Sigma^{\Sigma\cup\Sigma_\rho}(s_P)$. Then, $f_{ER}(s) = P_{\Sigma_\rho}^{\Sigma\cup\Sigma_\rho}(s_P)$, where $P_{\Sigma_\rho}^{\Sigma\cup\Sigma_\rho} : (\Sigma \cup \Sigma_\rho)^* \to \Sigma_\rho^*$.

Algorithm 6: Automaton G_P that models a f_{ER} extracted from G_M.

Input : G_M, X_S.
Output: $G_P = (X_P, \Sigma \cup \Sigma_\rho, f_P, x_{0,P})$.

1 Create set $X_{M,1} = \{x_M \in X_M : (\exists x \in X, \exists x_\alpha \in X_\alpha, \exists x_\beta \in X_\beta)[x_M = (x, x_{0,\iota}, x_\alpha, x_\beta)]\}$;
2 Create set $X_{M,2} = \{x_M \in X_M : (\exists x \in X, \exists x_\alpha \in X_\alpha, \exists x_\beta \in X_\beta)[x_M = (x, x_\sigma, x_\alpha, x_\beta)]\}$;
3 Create automaton $G_N = (X, \Sigma \cup \Sigma_\rho, f_N, x_{0,N})$ by erasing transitions labeled with reported events in Σ_ρ from states in $x_{M,2} \in X_{M,2}$ such that $|\Gamma_{G_N}(x_{M,2})| = 1$;
4 $G_P = (X_P, \Sigma \cup \Sigma_\rho, f_P, x_{0,P}) = Ac(G_N)$;

Example 4.6 *Let us consider G_M computed in Example 4.5. For this example, to obtain automaton G_N, let us choose events d_ρ in state 1, d_ρ in state 2, c_ρ in state 6, d_ρ in state 1, a_ρ in state 17, d_ρ in state 30, and a_ρ in state 31. The others $X_{M,2}$ states will became non-accessible. Thus, for the sake of simplicity, their choices are omitted in this example. After taking the accessible part of G_N in line 4 of Algorithm 6, automaton G_P is computed, depicted in Figure 4.12. Function f_{ER}, extracted from G_M is modeled by G_P.*

Note that, after the sequence $s_P = bd_\rho dc_\rho aa_\rho cd_\rho$ G_P reaches state 7. In this case, the system has executed $P_\Sigma^{\Sigma\cup\Sigma_\rho}(s_P) = bdac$, reaching state 3 of the plant, while the defense mechanism communicates $P_{\Sigma_\rho}^{\Sigma\cup\Sigma_\rho}(s_P) = \tilde{s} = d_\rho c_\rho a_\rho d_\rho$. Since the attacker observes only the events communicated through subnet SN_1, the attacker observes $P_{\Sigma_{\alpha,\rho}}^{\Sigma_\rho}(\tilde{s}) = d_\rho a_\rho d_\rho$ and estimates that the system is in state 4.

In Theorem 4.1 we prove that G_M represents all possible f_{ER} which satisfies Conditions **C2** and **C3**. Prior to Theorem 4.1, let us, first, introduce Lemma 4.1.

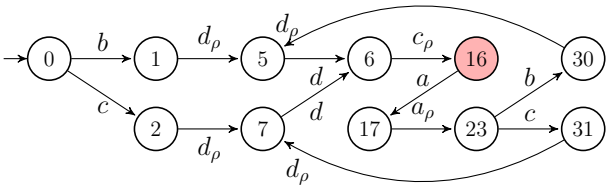


Figure 4.12: Automaton G_P of Example 4.6 that represents the chosen event replacement function f_{ERP}.

Lemma 4.1 *Let $G_2 = G\|G_\iota\|G_{\alpha,\rho}$, and let f_{ER} be an event replacement function. Then, f_{ER} satisfies Condition* **C2** *if, and only if, for all $s = \sigma_1\sigma_2\ldots\sigma_k \in L(G)$, there exists a sequence $s_2 = \sigma_1\tilde{\sigma}_1\sigma_2\tilde{\sigma}_2\ldots\sigma_k\tilde{\sigma}_k \in L(G_2)$, where $f_{ER}(s) = \tilde{\sigma}_1\tilde{\sigma}_2\ldots\tilde{\sigma}_k$.*

Proof: ($\Leftarrow$) Note that the interleaver automaton G_ι ensures that an event in Σ_ρ is generated in G_2 after the occurrence of any plant event. Thus, for all $s = \sigma_1\sigma_2\ldots\sigma_k \in L(G)$, there exists at least one sequence $s_2 = \sigma_1\tilde{\sigma}_1\sigma_2\tilde{\sigma}_2\ldots\sigma_k\tilde{\sigma}_k \in L(G_2)$. Let $f_{ER}(s)$ be defined as $f_{ER}(s) = P_{\Sigma_\rho}^{\Sigma\cup\Sigma_\rho}(s_2) = \tilde{s}$, for all $s \in L(G)$, where $s_2 \in L(G_2)$ is a chosen sequence such that $P_{\Sigma}^{\Sigma\cup\Sigma_\rho}(s_2) = s$. Since $G_2 = G\|G_\rho\|G_{\alpha,\rho}$, then, $P_{\Sigma_{\alpha,\rho}}^{\Sigma_\rho}(\tilde{s}) \in L(G_{\alpha,\rho})$, *i.e.*, f_{ER} satisfies Condition **C2**.

($\Rightarrow$) Let $s = \sigma_1\sigma_2\ldots\sigma_k \in L(G)$, and consider that $f_{ER}(s) = \tilde{s} = \tilde{\sigma}_1\tilde{\sigma}_2\ldots\tilde{\sigma}_k$ satisfies Condition **C2** for all $s \in L(G)$, *i.e.*, $P_{\Sigma_{\alpha,\rho}}^{\Sigma_\rho}(\tilde{s}) \in L(G_{\alpha,\rho})$. Let us form $s_2 = \sigma_1\tilde{\sigma}_1\sigma_2\tilde{\sigma}_2\ldots\sigma_k\tilde{\sigma}_k$, and assume that $s_2 \notin L(G_2)$. Since, according to the definition of G_ι, after the occurrence of an event in Σ, any event from Σ_ρ can be transmitted, the unique possibility for $s_2 \notin L(G_2)$, is that $P_{\Sigma_{\alpha,\rho}}^{\Sigma_\rho}(\tilde{s}) \notin L(G_{\alpha,\rho})$, where $\tilde{s} = P_{\Sigma_\rho}^{\Sigma\cup\Sigma_\rho}(s_2)$, which contradicts the assumption that f_{ER} satisfies Condition **C2**. □

Theorem 4.1 *Let $G_M = G\|G_\iota\|G_{\alpha,\rho}\|G_\beta$, and let f_{ER} be an event replacement function. Then, f_{ER} satisfies Conditions* **C2** *and* **C3** *if, and only if, for all $s = \sigma_1\sigma_2\ldots\sigma_k \in L(G)$, there exists $s_M = \sigma_1\tilde{\sigma}_1\sigma_2\tilde{\sigma}_2\ldots\sigma_k\tilde{\sigma}_k \in L(G_M)$, where $f_{ER}(s) = \tilde{\sigma}_1\tilde{\sigma}_2\ldots\tilde{\sigma}_k$.*

Proof: Note that, since $L(G_M) \subseteq L(G_2)$, then, according to Lemma 4.1, all the event replacement functions represented in G_M also satisfy Condition **C2**.

($\Leftarrow$) According to Algorithm 5, G_{β_i} implements a counter that counts the number of occurrences of events of Σ_i replaced with events of other subnets SN_j, for $j \neq i$. If an event is reported through the attacked subnet SN_i, the counter is reset, and if the counter reaches n_i, then an event of the attacked subnet SN_i is forced to be reported after the occurrence of any event in Σ_i. Thus, since $G_\beta = \|_{i\in I_\alpha}G_{\beta_i}$ and $G_M = G\|G_\iota\|G_{\alpha,\rho}\|G_\beta$, for any sequence $s_M \in L(G_M)$, there does not exist a subsequence t_M such that $\|P_{\Sigma_i}^{\Sigma}(P_{\Sigma}^{\Sigma\cup\Sigma_\rho}(t_M))\| > n_i$ and $P_{\Sigma_{\alpha,\rho_i}}^{\Sigma_\rho}(P_{\Sigma_\rho}^{\Sigma\cup\Sigma_\rho}(t_M)) = \varepsilon$, for all $i \in I_\alpha$. Thus, Condition **C3** is satisfied for any f_{ER} obtained from G_M.

($\Rightarrow$) The proof is carried out by finite induction on the length of the sequences $s \in L(G)$. Let f_{ER} be an event replacement function that satisfies Conditions **C2** and **C3**. Let $s = \sigma_1\sigma_2\ldots\sigma_k$ and $f_{ER}(s) = \tilde{s} = \tilde{\sigma}_1\tilde{\sigma}_2\ldots\tilde{\sigma}_k$, and consider that there exists a sequence $s_M = \sigma_1\tilde{\sigma}_1\sigma_2\tilde{\sigma}_2\ldots\sigma_k\tilde{\sigma}_k \in L(G_M)$. Let us now prove that for any sequence $s\sigma_{k+1} \in L(G)$ and $\tilde{s}\tilde{\sigma}_{k+1} \in \Sigma_\rho^*$ such that Conditions **C2** and **C3** are not violated, sequence $s_M\sigma_{k+1}\tilde{\sigma}_{k+1} \in L(G_M)$. Let $x'_M = f_M(x_{0,M}, s_M) = (x', x_{0,\iota}, x'_\alpha, x'_\beta)$. Since, $G_M = G\|G_\iota\|G_{\alpha,\rho}\|G_\beta$, then, $\sigma_{k+1} \in \Gamma_{G_M}(x'_M) \cap \Sigma$. Thus, there exists a state $x''_M = f_M(x'_M, \sigma_{k+1}) = (x'', x_{\sigma_{k+1}}, x''_\alpha, x''_\beta)$ where $\Gamma_{G_M}(x''_M) \subseteq \Sigma_\rho$. Since Condition **C3** is satisfied, there does not exist a subsequence $t \in \Sigma^*$ of s, such that $\|P^{\Sigma}_{\Sigma_i}(t)\| > n_i$, for $i \in I_\alpha$, $\sigma_{k+1} \in t$, and $f_{ER}(s) = \tilde{u}\tilde{t}\tilde{v}$, with $\|\tilde{u}\| = \|u\|$, $\|\tilde{t}\| = \|t\|$, $\tilde{\sigma}_{k+1} \in \tilde{t}$, and $P^{\Sigma_\rho}_{\Sigma_{\alpha,\rho_i}}(\tilde{t}) = \varepsilon$. Then, according to Algorithm 5, $\tilde{\sigma}_{k+1} \in \Gamma_{G_\beta}(x''_\beta)$. In addition, since f_{ER} satisfies Condition **C2**, $\tilde{\sigma}_{k+1} \in \Gamma_{G_2}((x'', x_{k+1}, x''_\alpha))$. Thus, $\tilde{\sigma}_{k+1} \in \Gamma_{G_M}(x''_M)$ and $s_M\sigma_{k+1}\tilde{\sigma}_{k+1} \in L(G_M)$.

To conclude the proof, let the initial state of G_M be given by $x_{0,M} = (x_0, x_{0,\iota}, x_{0,\alpha}, x_{0,\beta})$. Note that since $G_M = G\|G_\iota\|G_{\alpha,\rho}\|G_\beta$, then any $\sigma \in \Gamma_G(x_0)$ also belongs to $\Gamma_{G_M}(x_{0,M})$. Thus, $\sigma \in L(G_M)$. Let $x''_M = f_M(x_{0,M}, \sigma) = (x'', x_\sigma, x_{0,\alpha}, x''_\beta)$. Note that $\Gamma_{G_\iota}(x_\sigma) = \Sigma_\rho$ and $\Gamma_{G_{\alpha,\rho}}(x_{0,\alpha}) = \Sigma_\rho$. In addition, to satisfy Conditions **C2** and **C3**, $\Gamma_{G_\beta}(x''_\beta) = \Sigma_\rho$, if $\sigma \in \Sigma_i$ with $n_i \geq 1$ or $\sigma \in \Sigma_{\bar{\alpha}}$, or $\Gamma_{G_\beta}(x''_\beta) = \Sigma_{\alpha,\rho_i}$, if $\sigma \in \Sigma_i$ and $n_i = 0$. Thus, $\sigma\tilde{\sigma} \in L(G_M)$ for any $\tilde{\sigma}$ satisfying Conditions **C2** and **C3**. □

In the following example, we show that there may exist a f_{ER} represented in G_M that does not satisfy Condition **C1**.

Example 4.7 *Consider automaton G that models the behavior of the CPS presented in Example 4.2 with $X_S = \{4\}$. The events are separated according to the transmission subnets as $\Sigma_1 = \{a, d\}$ and $\Sigma_2 = \{b, c\}$. Let us consider that subnet SN_1 is vulnerable to eavesdropping attacks. Also, let us consider function f_{ER} extracted from G_M in Example 4.6, modeled by automaton G_P depicted in Figure 4.12.*

Note that, after the sequence $s_P = bd_\rho dc_\rho$ G_P reaches state 16. In this case, the system has executed $P^{\Sigma\cup\Sigma_\rho}_{\Sigma}(s_P) = bd$, reaching the secret state 4 of the plant, while the defense mechanism communicates $P^{\Sigma\cup\Sigma_\rho}_{\Sigma_\rho}(s_P) = \tilde{s} = d_\rho c_\rho$. Since the attacker observes only the events communicated through subnet SN_1, the attacker observes $P^{\Sigma_\rho}_{\Sigma_{\alpha,\rho}}(\tilde{s}) = d_\rho$ and estimates that the system is in state 4, which shows that this f_{ER} does not protect the secret.

According to Condition **C1**, *since the defense mechanism reports an event immediately after the occurrence of a plant event, the states of G_M that represent that the secret is revealed to the attacker are the states $x_M \in X_{M,1}$ for which the first component $x \in X_S$ and the third component satisfies $x_\alpha \in 2^{X_S}$. In Figure 4.13, we present automaton G_M with states 16 and 18, highlighted in red. These are the states of G_M in which the secret is revealed to the attacker. Thus, if G_P obtained*

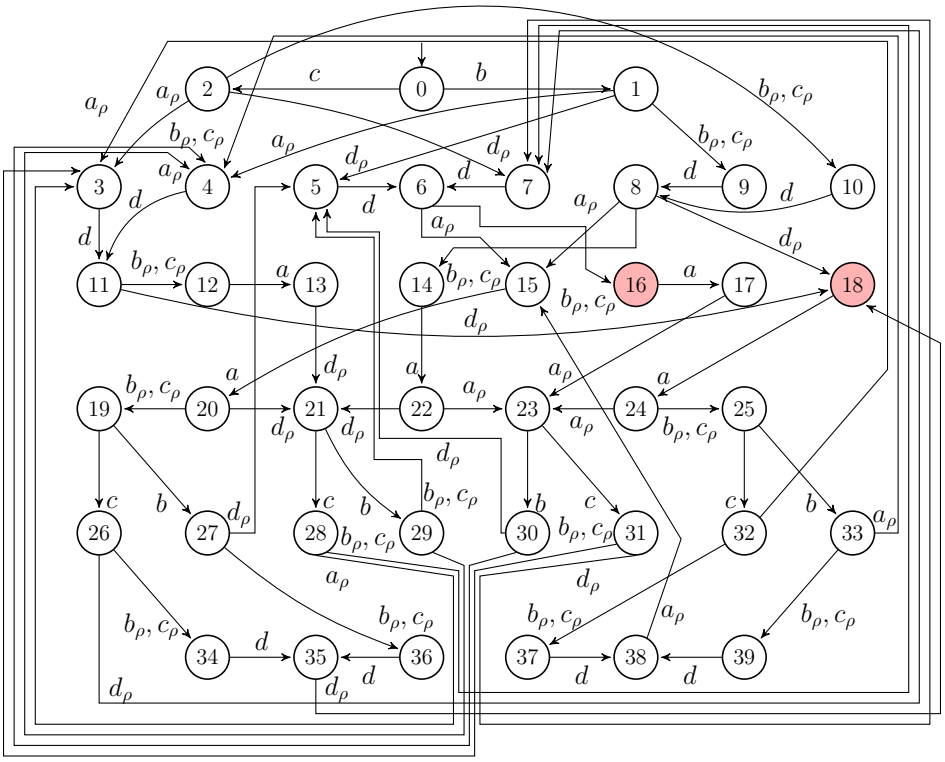


Figure 4.13: Automaton G_M of Example 4.7. States 16 and 18 represent the states in which the secret is revealed for the attacker.

from G_M has state 16 or 18, then the event replacement function f_{ER} extracted from G_P does not satisfy Condition **C1**. *In the sequel, we present a method to satisfy Condition* **C1**, *when possible.* □

In order to obtain an automaton that represents all f_{ER} satisfying Conditions **C1-C3**, it is necessary to eliminate from G_M the states where the system has reached a secret state of X_S and the intruder is certain that a secret state has been reached, obtaining a new automaton $\hat{G}_M$. It is important to remark that, after the elimination of these states, two types of undesirable states may emerge in automaton $\hat{G}_M$: (i) states $x_M = (x, x_\iota, x_\alpha, x_\beta) \in X_{M,1}$ such that $\Gamma_G(x) \neq \Gamma_{\hat{G}_M}(x_M)$, which implies that the event replacement function obtained from $\hat{G}_M$ disables a plant event, which is not possible since the obfuscator is a passive device; and (ii) states in $X_{M,2}$ without feasible events, which represents that there is a plant event that cannot be obfuscated. If any of the two types of states is observed in $\hat{G}_M$, there exist event replacement functions represented in $\hat{G}_M$ that are not admissible, *i.e.*, that cannot be realized. Thus, it is necessary to eliminate these states from $\hat{G}_M$, obtaining a new automaton denoted as G_A, which represents all admissible event replacement functions. Automaton $G_A = Prune(G_M, X_R)$ is obtained according to Algorithm 7, where $X_R := \{x_M = (x, x_\iota, x_\alpha, x_\beta) \in X_M : x \in X_S, x_\iota = x_{0,\iota}, x_\alpha \in 2^{X_S}, x_\beta \in X_\beta\}$, that represents all states of X_M that reveal the secret, violating Condition **C1**.

In line 1 of Algorithm 7, all states of $G_{in} = G_M$ that reveal the secret to the attacker are eliminated, generating automaton G'_{in}. Then, in line 2, the accessible part of G'_{in} is computed, generating automaton $\hat{G}_{in}$. In line 3, a binary variable v is initialized as 1. In lines 4 to 18, we recursively eliminate states of $\hat{G}_{in}$ such that a feasible event of the plant is disabled by the event replacement function (lines 7 to 9), or that there is a plant event that cannot be obfuscated (lines 10 to 12). The elimination of states continues until there are no more undesirable states in $\hat{G}_{in}$, which is verified in line 15. Finally, in line 19, $G_A = G_{out}$ is obtained.

Theorem 4.2 *Let $G_M = (X_M, \Sigma \cup \Sigma_\rho, f_M, x_{0,M}) = G \| G_\iota \| G_{\alpha,\rho} \| G_\beta$ and $G_A = (X_A, \Sigma \cup \Sigma_\rho, f_A, x_{0,A}) = Prune(G_M, X_R)$. Then, G_A is different from the empty automaton if, and only if, there exists a $f_{ER}(s)$ which satisfies Conditions* **C1**, **C2**, *and* **C3** *for all sequences $s \in L(G)$.*

Proof: Since $G_M = G \| G_\iota \| G_{\alpha,\rho} \| G_\beta$, then G_M cannot be the empty automaton. In addition, according to Theorem 4.1, G_M represents all $f_{ER}(s)$ which satisfies Conditions **C2**, and **C3** for all sequences $s \in L(G)$.

($\Rightarrow$) The proof is carried out by finite induction on the length of the sequences $s \in L(G)$. Let $s = \sigma_1 \sigma_2 \ldots \sigma_k$ be any sequence of length k generated by the plant, and let us assume that there exists a sequence of events $s_A = \sigma_1 \tilde{\sigma}_1 \sigma_2 \tilde{\sigma}_2 \ldots \sigma_k \tilde{\sigma}_k \in L(G_A)$

Algorithm 7: $G_{out} = Prune(G_{in}, X_{prune})$.

```
1  Compute G'_in by eliminating from G_in all states x_in ∈ X_prune;
2  Ĝ_in = Ac(G'_in) = (X̂_in, Σ ∪ Σ_ρ, f̂_in, x̂_{0,in});
3  v ← 1;
4  while v = 1 do
5  |   X_old ← X̂_in;
6  |   for each x_in = (x, x_ι, x_α, x_β) ∈ X̂_in do
7  |   |   if x_ι = x_{0,ι} and Γ_{Ĝ_in}(x_in) ≠ Γ_G(x) then
8  |   |   |   Eliminate x_in from X̂_in;
9  |   |   end
10 |   |   if x_ι ≠ x_{0,ι} and Γ_{Ĝ_in}(x_in) = ∅ then
11 |   |   |   Eliminate x_in from X̂_in;
12 |   |   end
13 |   end
14 |   Ĝ_in ← Ac(Ĝ_in);
15 |   if X_old = X̂_in then
16 |   |   v ← 0
17 |   end
18 end
19 G_out ← Ĝ_in
```

that maps the sequence of events of the plant s. Thus, sequence s is obfuscated as sequence $\tilde{s} = f_{ER}(s) = \tilde{\sigma}_1\tilde{\sigma}_2 \ldots \tilde{\sigma}_k$ for transmission. Let us now prove that, according to Algorithm 7, there exists in G_A an obfuscation mapping for any sequence $s\sigma_{k+1} \in L(G)$. Since $s_A \in L(G_A)$, then there exists state $x_A = (x, x_{0,\iota}, x_\alpha, x_\beta) \in X_A$ such that $x_A = f_A(x_{0,A}, s_A)$. Since, according to lines 7 to 9 of Algorithm 7, all states $x_{in} \in X_{M,1}$ such that $\Gamma_{\hat{G}_{in}}(x_{in}) \neq \Gamma_G(x)$ are recursively removed from $\hat{G}_{in}$ before the computation of G_A, then $\Gamma_{G_A}(x_A) = \Gamma_G(x)$. Thus, all sequences $s\sigma_{k+1} \in L(G)$ are mapped in G_A, *i.e.*, $s_A\sigma_{k+1} \in L(G_A)$. Since in lines 10 to 12 of Algorithm 7 all states $x_{in} \in X_{M,2}$ such that $\Gamma_{\hat{G}_{in}}(x_{in}) = \emptyset$ are removed from $\hat{G}_{in}$, then there exists a sequence $s_A\sigma_{k+1}\tilde{\sigma}_{k+1} \in L(G_A)$, *i.e.*, $s\sigma_{k+1}$ is obfuscated as $f_{ER}(s\sigma_{k+1}) = \tilde{s}\tilde{\sigma}_{k+1}$ for transmission. In order to conclude the proof, note that since G_A is not the empty automaton, then G_A has an initial state $x_{0,E} = (x_0, x_{0,\iota}, x_{0,\alpha}, x_{0,\beta})$ that has not been removed by Algorithm 7, which implies that $\Gamma_{G_A}(x_{0,A}) = \Gamma_G(x_0)$. After the execution of any event σ in $\Gamma_G(x_0)$, a state $x_E = (x, x_\sigma, x_{0,\alpha}, x_\beta)$ is reached. According to Algorithm 7, $\Gamma_{G_A}(x_A) \neq \emptyset$, and the proof is concluded.

($\Leftarrow$) Let us suppose that $G_A = Prune(G_M, X_R)$ is the empty automaton and there exists a f_{ER} which satisfies Conditions **C1**-**C3** for all sequences $s \in L(G)$. Note that, according to Theorem 4.1, G_M represents all f_{ER} that satisfy Conditions **C2** and **C3**. Then, in line 1 of Algorithm 7, only the states that contradict Condition **C1** are removed. Note that $\hat{G}_{in}$ has all f_{ER} that ensure that the secret is not revealed,

Table 4.2: Upper bound in the number of states and transitions of automata G, G_ι, $G_{\alpha,\rho}$, and G_β.

	No. of states	No. of transitions
G	$\|X\|$	$\|X\| \times \|\Sigma\|$
G_ι	$\|\Sigma\| + 1$	$(\|\Sigma\| + 1) \times \|\Sigma\|$
$G_{\alpha,\rho}$	$2^{\|X\|}$	$2^{\|X\|} \times \|\Sigma_\alpha\|$
G_β	$2 \times \prod_{i \in I_\alpha}(n_i + 1)$	$2 \times \prod_{i \in I_\alpha}(n_i + 1)\,[\|\Sigma_\rho\| + \|\Sigma_\alpha\|]$

satisfying Condition **C1**. However, there may exist in $\hat{G}_{in}$ states that belong to $X_{M,2}$ that do not have any possible reported event, or states in $X_{M,1}$ that restrict the occurrence of feasible plant events. In these cases, there are event replacement functions represented in $\hat{G}_{in}$ which are not admissible. In order to ensure that G_A has only admissible event replacement functions, in lines 7 to 12 of Algorithm 7, only states that violate the admissibility conditions are recursively removed until there does not exist any state that violates these conditions. Thus, if there is any admissible f_{ER} that satisfies Conditions **C1**-**C3**, then it must be represented in G_A. □

Note that, according to Algorithm 7, the computational complexity for obtaining automaton G_A is equal to the complexity of computing automaton G_M. Since $G_M = G \| G_\iota \| G_{\alpha,\rho} \| G_\beta$, and an upper bound for the number of states and transitions of G, G_ι, $G_{\alpha,\rho}$, and G_β can be obtained as presented in Table 4.2, the number of states of G_A is $O\left(|X| \times |\Sigma| \times 2^{|X|} \times \prod_{i \in I_\alpha}(n_i + 1)\right)$. In addition, the number of transitions in G_A is $O\left(|X| \times |\Sigma|^2 \times 2^{|X|} \times \prod_{i \in I_\alpha}(n_i + 1)\right)$.

Example 4.8 *Let us consider automaton G_M, computed in Example 4.5, depicted in Figure 4.14. Le us now compute automaton G_A from G_M using Algorithm 7.*

First, the red states 16 and 18, which reveal the secret to the intruder, are removed. Then, we also remove the yellow states which: (i) become non-accessible after eliminating the states 16 and 18 (states 17, 24, 25, 32, 33, 37, 38, 39); or (ii) represent states such that no event can be reported by the obfuscator after the occurrence of a plant event (state 35); or (iii) states that a feasible event of the plant should be disabled by the obfuscator (states 34 and 36). In this case, $G_A = Prune(G_M, X_R)$, depicted in Figure 4.15, has 27 states and 51 transitions. Since G_A is different from the empty automaton, then, according to Theorem 4.2, there exist functions f_{ER} which satisfy Conditions **C1**-**C3**. □

Theorem 4.2 states that G_A is different from the empty automaton if, and only if, there exists a $f_{ER}(s)$ which satisfies Conditions **C1**, **C2**, and **C3** for all sequences $s \in L(G)$. However, in this work, we consider that the useful information for the intended receiver is the knowledge of the system current state after the execution of any sequence of events of the plant, which means that data confidentiality must be

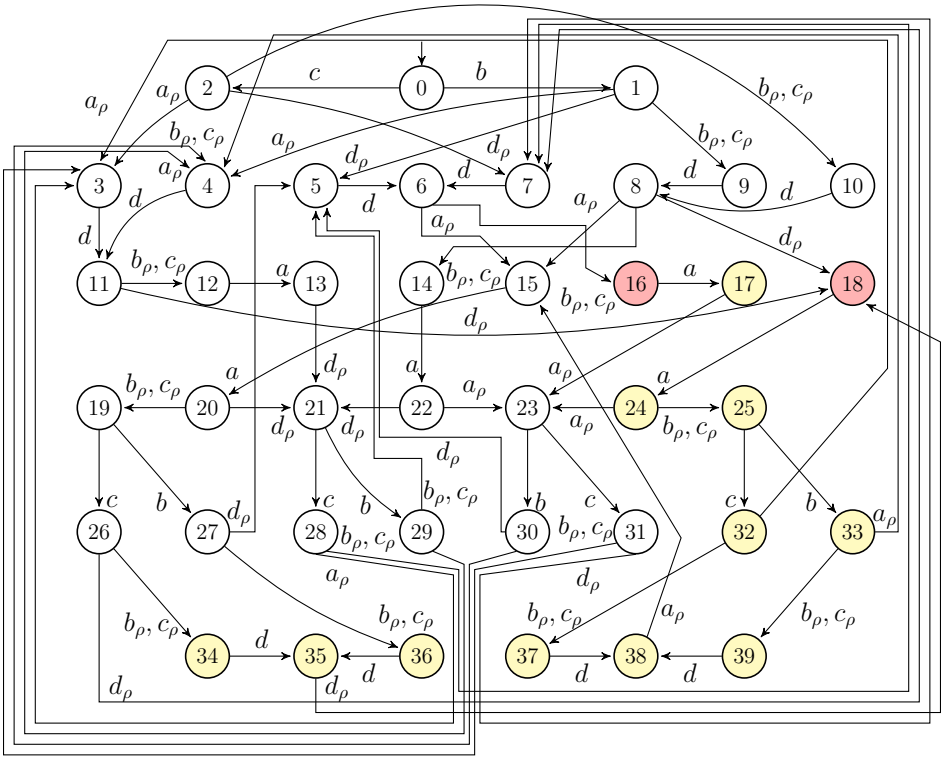


Figure 4.14: Automaton G_M of Example 4.8. The red and yellow states are removed by Algorithm 7.

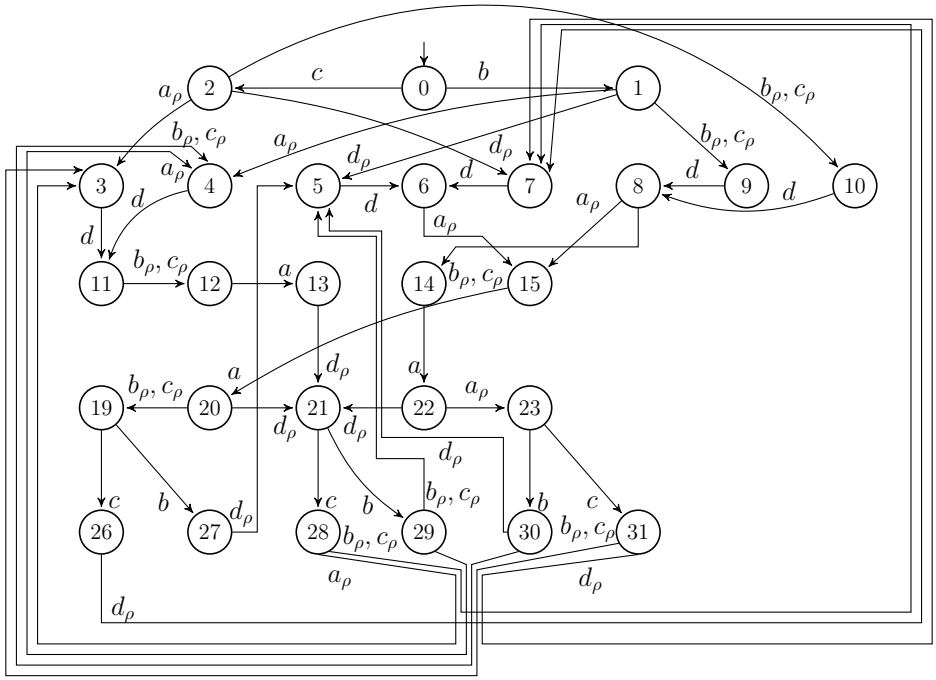


Figure 4.15: Automaton $G_A = Prune(G_M, X_R)$ of Example 4.8.

guaranteed. In order to do so, in the next section, we present the conditions for the intended receiver to be certain about the event that has occurred and the current state of the system, *i.e.*, we present the conditions for obtaining a reversible event replacement function.

4.4 Reversible event replacement function

Let f_{ER}, represented by G_P, be obtained from G_A. In order to recover the original sequence of events $s \in L(G)$, the intended receiver must be able to decipher the observed sequence of events $\tilde{s} = f_{ER}(s)$. To do so, a recovering automaton, $\bar{G}_P$, can be constructed using Algorithm 8.

In line 1 of Algorithm 8, the set of states $\bar{X}_P$ is initialized as $X_P \cap X_{M,1}$. In lines 2 to 16, each state $x_P \in X_P \cap X_{M,1}$ is explored, and for each sequence $\sigma\tilde{\sigma} \in \Sigma\Sigma_\rho$, if the transition $f_P(x_P, \sigma\tilde{\sigma})$ is defined, a new state $x_P^{\sigma\tilde{\sigma}}$ is created in line 5, and the transitions related to this new state are created as follows. In lines 6 to 8, if the transition $\bar{f}_P(x_P, \tilde{\sigma})$ is defined, it is updated to $\bar{f}_P(x_P, \tilde{\sigma}) \cup \{x_P^{\sigma\tilde{\sigma}}\}$, otherwise, in line 9, it gets $\{x_P^{\sigma\tilde{\sigma}}\}$. Then, in line 11, $\bar{f}_P(x_P^{\sigma\tilde{\sigma}}, \sigma)$ is defined as $f_P(x_P, \sigma\tilde{\sigma})$ and the set of states $\bar{X}_P$ is updated to $\bar{X}_P \cup \{x_P^{\sigma\tilde{\sigma}}\}$ in line 12. Finally, in line 16, $\bar{G}_P = (\bar{X}_P, \Sigma \cup \Sigma_\rho, \bar{f}_P, x_{0,P})$.

Algorithm 8: Computation of automaton $\bar{G}_P$.

Input : $G_P = (X_P, \Sigma \cup \Sigma_\rho, f_P, x_{0,P})$.
Output: $\bar{G}_P = (\bar{X}_P, \Sigma \cup \Sigma_\rho, \bar{f}_P, x_{0,P})$.

```
1  X̄_P ← X_P ∩ X_M,1;
2  for x_P ∈ X_P ∩ X_M,1 do
3  |  for (σ ∈ Σ) ∧ (σ̃ ∈ Σ_ρ) do
4  |  |  if f_P(x_P, σσ̃)! then
5  |  |  |  Create state x_P^{σσ̃};
6  |  |  |  if f̄_P(x_P, σ̃)! then
7  |  |  |  |  f̄_P(x_P, σ̃) ← f̄_P(x_P, σ̃) ∪ {x_P^{σσ̃}}
8  |  |  |  else
9  |  |  |  |  f̄_P(x_P, σ̃) ← {x_P^{σσ̃}}
10 |  |  |  end
11 |  |  |  f̄_P(x_P^{σσ̃}, σ) ← {f_P(x_P, σσ̃)};
12 |  |  |  X̄_P ← X̄_P ∪ {x_P^{σσ̃}}
13 |  |  end
14 |  end
15 end
16 Ḡ_P = (X̄_P, Σ ∪ Σ_ρ, f̄_P, x_{0,P});
```

After the observation of an event $\tilde{\sigma} \in \Sigma_\rho$ from a state $x_P \in \bar{X}_P \cap X_{M,1}$, an auxiliary state $x_P^{\sigma\tilde{\sigma}}$ of $\bar{G}_P$ is reached, whose unique feasible event is σ, indicating

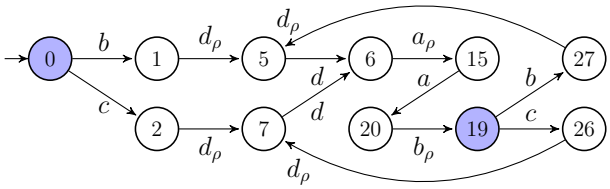


Figure 4.16: Automaton G_P of Example 4.9 that represents the chosen event replacement function f_{ERP}. States 0 and 19, highlighted in blue, represents states that will become non-deterministic in $\bar{G}_P$.

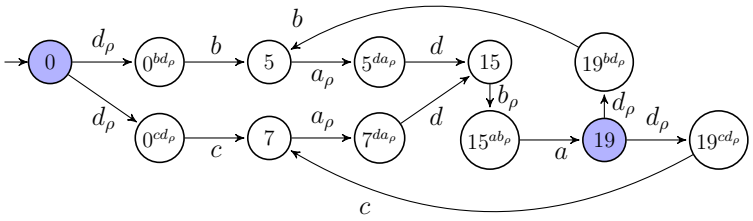


Figure 4.17: Automaton $\bar{G}_P$ of Example 4.9. States 0 and 19, highlighted in blue, are non-deterministic states.

to the receiver the plant event associated with $\tilde{\sigma}$. By following $\bar{G}_P$, the receiver can infer which plant event may have occurred after the observation of a reported event.

Example 4.9 *Consider automaton G_A computed in Example 4.8 and let us extract from G_A a function f_{ER} by following the steps of Algorithm 6. Automaton G_P, depicted in Figure 4.16 models the extracted function f_{ER}.*

Let us compute automaton $\bar{G}_P$ from G_P by following the steps of Algorithm 8. In line 1, the set of states $\bar{X}_P$ is initialized as $\{0, 5, 7, 15, 19\}$. In lines 2 to 16, each state $x_P \in \{0, 5, 7, 15, 19\}$ is explored, and for each sequence $\sigma\tilde{\sigma} \in \Sigma\Sigma_\rho$, if the transition $f_P(x_P, \sigma\tilde{\sigma})$ is defined, a new state $x_P^{\sigma\tilde{\sigma}}$ is created. Let us consider state 0, since transition $f_P(0, bd_\rho)$ is defined, a new state 0^{bd_ρ} is created in line 5. Then, $\bar{f}_P(0, d_\rho)$ gets $\{0^{bd_\rho}\}$ in line 9 and, in line 11, $\bar{f}_P(0^{bd_\rho}, b)$ gets $\{5\}$. In line 12 $\bar{X}_P$ is updated as $\bar{X}_P = \{0, 5, 7, 15, 19, 0^{bd_\rho}\}$. Then, since $f_P(0, cd_\rho)$ is defined, a new state 0^{cd_ρ} is created in line 5. Then, since $\bar{f}_P(0, d_\rho)$ is defined, $\bar{f}_P(0, d_\rho)$ gets $\{0^{bd_\rho}, 0^{cd_\rho}\}$ in line 9 and, in line 11, $\bar{f}_P(0^{cd_\rho}, c)$ gets $\{7\}$. In line 12 $\bar{X}_P$ is updated as $\bar{X}_P = \{0, 5, 7, 15, 19, 0^{bd_\rho}, 0^{cd_\rho}\}$.

Algorithm 8 continues until $\bar{G}_P$ is completed, depicted in Figure 4.17. Note that states 0 and 19 are non-deterministics, and since automaton $\bar{G}_P$ is used by the intended receiver to discover the original plant event that has occurred, with a non-deterministic state the intended receiver cannot be sure of the plant event that has occurred. For example, let us suppose that the plant is in the initial state 0, and the intended receiver observes event d_ρ. Then, the intended receiver is not sure if the

system has reached state 5 after event b or state 7 after event c. □

In the sequel, we show that the receiver is certain of the event σ generated by the plant, after observing a reported event $\tilde{\sigma}$, if, and only if, $\bar{G}_P$ is deterministic.

Theorem 4.3 *Let f_{ER} be an event replacement function satisfying Conditions* **C1**-**C3**, *and let G_P be the subautomaton extracted from G_A which represents the chosen f_{ER}. Then, f_{ER} is reversible if, and only if, $\bar{G}_P$ is deterministic.*

Proof: Let x_P be the current state of G_P and let σ be the event generated by the plant, and $\tilde{\sigma}$ be the event reported to the receiver after the occurrence of σ, according to the f_{ER} represented by G_P. Let us prove that, if $\bar{G}_P$ is deterministic, then event σ can be recovered from the observation of $\tilde{\sigma}$. According to the computation of $\bar{G}_P$, considering that x_P is its current state, after the observation of $\tilde{\sigma}$, state ${x_P}^{\sigma\tilde{\sigma}}$ is reached. Since σ is the unique feasible event of ${x_P}^{\sigma\tilde{\sigma}}$, then, event σ is recovered by the receiver.

Note that, since G deterministic and, according to the computation of $\bar{G}_P$ using Algorithm 8, $|\Gamma_{\bar{G}_P(x_P^{\sigma\tilde{\sigma}})}| = 1$, then $\bar{G}_P$ is nondeterministic if, and only if, there exist $\tilde{\sigma} \in \Sigma_\rho$ and $x_P \in \bar{X}_P$ such that $|\bar{f}_P(x_P, \tilde{\sigma})| > 1$.

Let us consider now that the receiver estimates correctly the current state of G_P, $x_P \in \bar{X}_P \cap X_{M,1}$. Then, after the observation of an event $\tilde{\sigma}$ such that $|\bar{f}_P(x_P, \tilde{\sigma})| > 1$, at least two different states ${x_P}^{\sigma\tilde{\sigma}}$ and ${x_P}^{\sigma'\tilde{\sigma}}$ can be reached, and the receiver is not certain whether σ or σ' has occurred in the system. □

According to Theorem 4.3, to obtain a reversible f_{ER}, it is necessary to choose G_P from G_A such that $\bar{G}_P$, computed using Algorithm 8 from G_P, is deterministic. In the sequel, we present a necessary and sufficient condition for the reversibility of the event replacement function associated with G_P. In order to do so, we need to first make the following definition.

Definition 4.1 *Given a f_{ER} represented in G_P, a state $x_P \in X_P \cap X_{M,1}$ is a reversible-policy state (RP state) if for all $\sigma_1, \sigma_2 \in \Sigma$, such that $\sigma_1 \neq \sigma_2$, and $\tilde{\sigma}_1, \tilde{\sigma}_2 \in \Sigma_\rho$, if $f_P(x_P, \sigma_1\tilde{\sigma}_1)$ and $f_P(x_P, \sigma_2\tilde{\sigma}_2)$ are defined, then $\tilde{\sigma}_1 \neq \tilde{\sigma}_2$.*

Example 4.10 *Let us consider a new function f_{ER} modeled by G_P, depicted in Figure 4.18. Let us verify the RP states from G_P. In order to be an RP state, first x_P must be in $X_{M,1}$. Thus, we have to verify if $0, 3, 4, 5, 7, 12, 15, 19, 21$ are RP states.*

Note that states 3, 4, 5, 7, 12 and 15 are RP states since there is only one active event in each of these states. Then, there does not exist any $\sigma_1, \sigma_2 \in \Sigma$, such that $\sigma_1 \neq \sigma_2$ such that $f_P(x_P, \sigma_1\tilde{\sigma}_1)$ and $f_P(x_P, \sigma_2\tilde{\sigma}_2)$ are defined. Thus, Definition 4.1 is not violated.

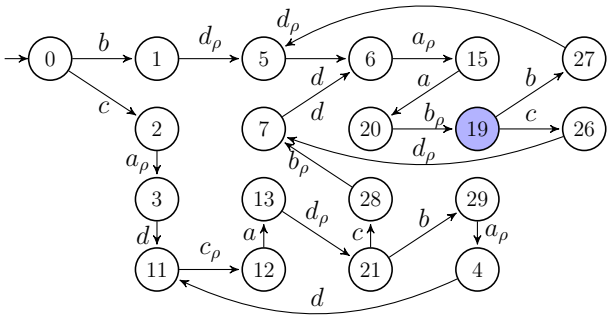


Figure 4.18: Automaton G_P of Example 4.10 that represents the chosen event replacement function f_{ER}. State 19, highlighted in blue, represents the state that is not an RP state of G_P.

Now, regarding state 0, note that $f_P(0, bd_\rho)$ and $f_P(0, ca_\rho)$ are defined, but $d_\rho \neq a_\rho$. Regarding state 21, note that $f_P(21, ba_\rho)$ and $f_P(21, cb_\rho)$ are defined, but $a_\rho \neq b_\rho$. However, considering state 19, highlighted in blue in Figure 4.18, $f_P(19, bd_\rho)$ and $f_P(19, cd_\rho)$ are defined, $b \neq c$ but $d_\rho = d_\rho$. Thus, state 19 is not an RP state. □

Theorem 4.4 *G_P represents a reversible f_{ER} if, and only if, all states $x_P \in X_P \cap X_{M,1}$ are RP states.*

Proof: ($\Leftarrow$) If all states $x_P \in X_P \cap X_{M,1}$ are RP states, then for all $\sigma_1, \sigma_2 \in \Sigma$, such that $\sigma_1 \neq \sigma_2$, there do not exist $\tilde{\sigma}_1, \tilde{\sigma}_2 \in \Sigma_\rho$, such that $f_P(x_P, \sigma_1\tilde{\sigma}_1)$ and $f_P(x_P, \sigma_2\tilde{\sigma}_2)$ are defined and $\tilde{\sigma}_1 = \tilde{\sigma}_2$. Then, $\bar{G}_P$, obtained according to Algorithm 8 from G_P, is deterministic, since the condition of line 6 of Algorithm 8 is always false, and $|\bar{f}_P(x_P, \tilde{\sigma})| = 1$, resulting in a deterministic transition function $\bar{f}_P$. Thus, according to Theorem 4.3, G_P represents a reversible f_{ER}.

($\Rightarrow$) Assuming that $x_P \in X_P \cap X_{M,1}$ is not an RP state, there exist $\sigma_1, \sigma_2 \in \Sigma$, such that $\sigma_1 \neq \sigma_2$, and there exist $\tilde{\sigma}_1$, $\tilde{\sigma}_2 \in \Sigma_\rho$, such that $f_P(x_P, \sigma_1\tilde{\sigma}_1)$ and $f_P(x_P, \sigma_2\tilde{\sigma}_2)$ are defined and $\tilde{\sigma}_1 = \tilde{\sigma}_2$. In this case, $\bar{G}_P$, obtained from G_P according to Algorithm 8, is nondeterministic, and according to Theorem 4.3, the receiver cannot be certain about which event σ has occurred after the observation of $\tilde{\sigma}$, which means that G_P does not represent a reversible f_{ER}. □

In some cases, a state $x_A \in X_A \cap X_{M,1}$ of G_A is not an RP state of any G_P obtained from G_A. In these cases, x_A is called a non-RP state (NRP state). In the cases that x_A can be an RP state of a chosen G_P, G_A is called a possible RP state (PRP state). In the following, we present an example to illustrate a PRP state and an NRP state.

Example 4.11 *Let us consider automaton G_A computed in Example 4.8. Since G_P is computed from G_A by following the steps of Algorithm 6, in line 3, the transitions*

of the states $x_{M,2} \in X_{M,2}$ are removed until $|\Gamma_{G_N}(x_{M,2})| = 1$, there is only one feasible event for each state $x_{M,2} \in X_P$.

Now, let us consider state 0 from G_A, depicted in Figure 4.15. Note that after the occurrence of event b, G_A reaches state 1, and $\Gamma_{G_A}(1) = \{a_\rho, b_\rho, c_\rho, d_\rho\}$. In addition, after the occurrence of event c, G_A reaches state 2, and $\Gamma_{G_A}(2) = \{a_\rho, b_\rho, c_\rho, d_\rho\}$. Then, it is possible to obtain a G_P in which $\Gamma_{G_A}(1) = \{a_\rho\}$ and $\Gamma_{G_A}(2) = \{c_\rho\}$. Thus, 0 is a PRP state.

Now, let us consider state 19 from G_A. After the occurrence of event b, G_A reaches state 27 and $\Gamma_{G_A}(27) = \{d_\rho\}$, and after the occurrence of event c, G_A reaches state 26 and $\Gamma_{G_A}(26) = \{d_\rho\}$. Thus, since for any G_P that contains state 19, $f_P(19, bd_\rho)$ and $f_P(19, cd_\rho)$ are defined, $b \neq c$ and $d_\rho = d_\rho$, then 19 is an NRP state.

The problem of verifying if a state $x_A \in X_A \cap X_{M,1}$ is a PRP state is equivalent to the problem of verifying the existence of a CDR for the family of sets $F = \{F_1, \ldots, F_m\}$, where $m = |\Gamma_{G_A}(x_A)|$ and $F_i = \Gamma_{G_A}((f_A(x_A, \sigma_i)))$, with $\sigma_i \in \Gamma_{G_A}(x_A)$, for $i = 1, \ldots, m$. According to Theorem 2.1, x_A is a PRP state, if, and only if, for all $B \in 2^{I_m}$, $|B| \leq \left|\bigcup_{i \in B} F_i\right|$. Note that if a state of $X_A \cap X_{M,1}$ is not PRP, then it is an NRP state. Thus, automaton G_O, that models all possible reversible f_{ER}, is computed by following the steps of Algorithm 9. In line 1, the set X_{NRP} is created with the elements $x \in X_A \cap X_{M,1}$ such that x is an NRP state. In line 2, automaton G'_A is created and gets G_A. In lines 3 to 7, the NRP states are removed recursively using Algorithm 7. In line 4, $G_{aux} = Prune(G'_A)$ is computed. In line 5, the set X_{NRP} is computed again such that $X_{NRP} = \{x \in X_{aux} \cap X_{M,1}$ such that x is an NRP state$\}$, and in line 6, G'_A gets G_{aux}. Finally, in line 8 $G_O = G'_A$ is obtained with no NRP state. Thus, if G_O is not the empty automaton, all states of G_O that belongs to $X_{M,1}$ are PRP states.

Algorithm 9: Computation of automaton G_O.

Input : $G_A = (X_A, \Sigma \cup \Sigma_\rho, f_A, x_{0,A})$.
Output: $G_O = (X_O, \Sigma \cup \Sigma_\rho, f_O, x_{0,O})$.

1 $X_{NRP} = \{x \in X_A \cap X_{M,1} : x$ is an NRP state$\}$;
2 $G'_A \leftarrow G_A$;
3 **while** $X_{NRP} \neq \emptyset$ **do**
4 $\quad G_{aux} = (X_{aux}, \Sigma \cup \Sigma_\rho, f_{aux}, x_{0,aux}) = Prune(G'_A, X_{NRP})$;
5 $\quad X_{NRP} = \{x \in X_{aux} \cap X_{M,1} : x$ is an NRP state$\}$;
6 $\quad G'_A \leftarrow G_{aux}$;
7 **end**
8 $G_O \leftarrow G'_A$

Theorem 4.5 *G_O is different from the empty automaton, if, and only if, there exists a f_{ER} such that Conditions* **C1**-**C4** *are satisfied.*

Proof: Similar to the proof of Theorem 4.2.

Note that, since all states $x_O \in X_O \cap X_{M,1}$ are PRP states, then it is possible to choose, for each event in $\Gamma_{G_O}(x_O)$, a different event in Σ_ρ to be reported. This guarantees that we can obtain from G_O an automaton G_P such that all its states are RP states. The Edmonds-Karp algorithm [82], can be used to choose different reported events for the feasible events of a state $x_O \in X_O \cap X_{M,1}$ with computational complexity $O\left(|\Sigma| \times |\Sigma_\rho|^2\right)$. Thus, G_P can be computed from G_O with complexity $O\left(|X_{M,1}| \times |\Sigma| \times |\Sigma_\rho|^2\right)$.

Example 4.12 *Consider automaton G_A computed in Example 4.8, and let us compute automaton G_O, by following the steps of Algorithm 9. In line 1, the set X_{NRP} is created as $\{19\}$, since state 19, represented in blue in Figure 4.19, is an NRP state as presented in example 4.10. In line 2, $G'_A = G_A$, and in line 3 is verified that $X_{NRP} = \{19\}$ is different from the empty set. Then, in line 4, $G_{aux} = Prune(G'_A)$ is computed, and state 19 is removed. In addition, states 26 and 27 of G'_A, represented in gray in Figure 4.19, become non-accessible after the elimination of state 19 and are removed in the computation of G_{aux}. In line 5, the set X_{NRP} is computed again, and since there does not exist any NRP state, then $X_{NRP} = \emptyset$. Finally, in line 8 $G_O = G'_A$, depicted in Figure 4.20, is obtained with no NRP state. Since G_O is not an empty automaton, according to Theorem 4.5, there exists a f_{ER} which satisfies Conditions* **C1**-**C4** *for all sequences $s \in L(G)$.*

Now, let us extract from G_O a f_{ER} using Algorithm 6. In line 3 of Algorithm 6, for each state $x_{M,2} \in X_{M,2}$, one event $e \in \Sigma_\rho$ is chosen as the obfuscation event so that $|\Gamma_{G_P}(x_{M,2})| = 1$. To do so, we have to investigate each state $x_{M,1} \in X_{M,1}$ and choose the events for states $x_{M,2} \in X_{M,2}$. This choice is carried out with Algorithm 1. Let us consider state 0, since $f_O(0, b) = 1$, and $f_O(0, c) = 2$, then $F = \{F_1, F_2\}$, where F_1 represents state 1 and F_2 represents state 2. In addition, $F_1 = \Gamma_{G_O}(1) = \{a_\rho, b_\rho, c_\rho, d_\rho\}$,$F_2 = \Gamma_{G_O}(2) = \{a_\rho, b_\rho, c_\rho, d_\rho\}$, and $S = \{a_\rho, b_\rho, c_\rho, d_\rho\}$. Then, in line 1 of Algorithm 1, the vertex for F_1 and F_2 are created, and, in line 2, the vertex for a_ρ, b_ρ, c_ρ, and d_ρ are created. In lines 3 to 5, the vertex that represent F_1 is connected to a_ρ, b_ρ, c_ρ, and d_ρ, labeled with $0/\infty$, and the vertex that represent F_2 is connected to a_ρ, b_ρ, c_ρ, and d_ρ, labeled with $0/\infty$. In line 6, a source vertex, s_0, is created, and, in lines 7 to 9, s_0 is connected to F_1 and F_2, each connection labeled with $0/1$. Then, in line 10 a destiny vertex, d_0, is created, and in lines 11 to 13 a_ρ, b_ρ, c_ρ, and d_ρ are connected to d_0 labeled with $0/1$. This steps are represented in Figure 4.21.

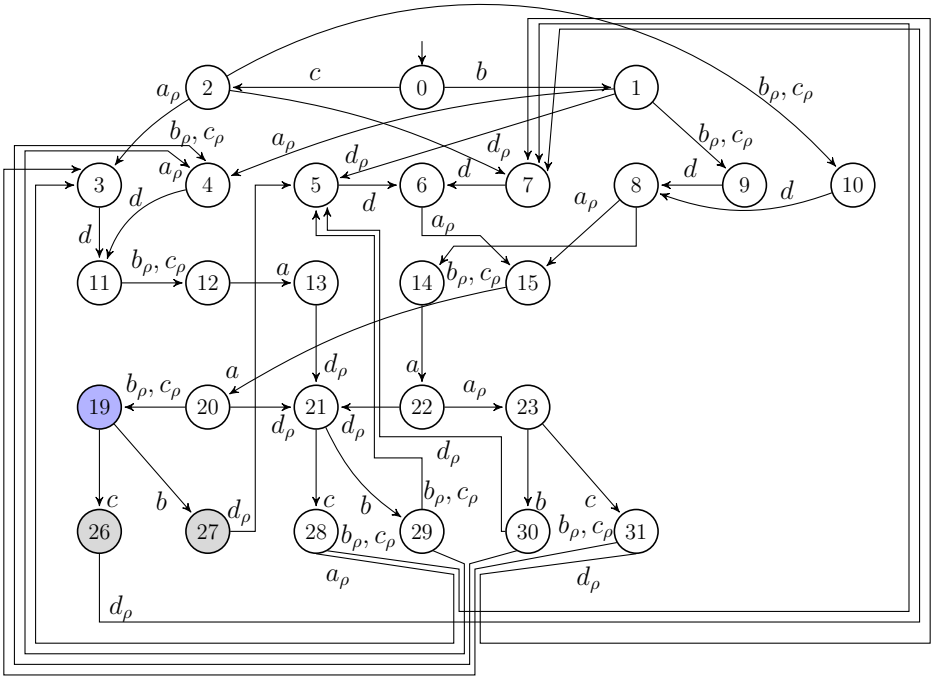


Figure 4.19: Automaton $G_A = Prune(G_M, X_R)$ of Example 4.8.

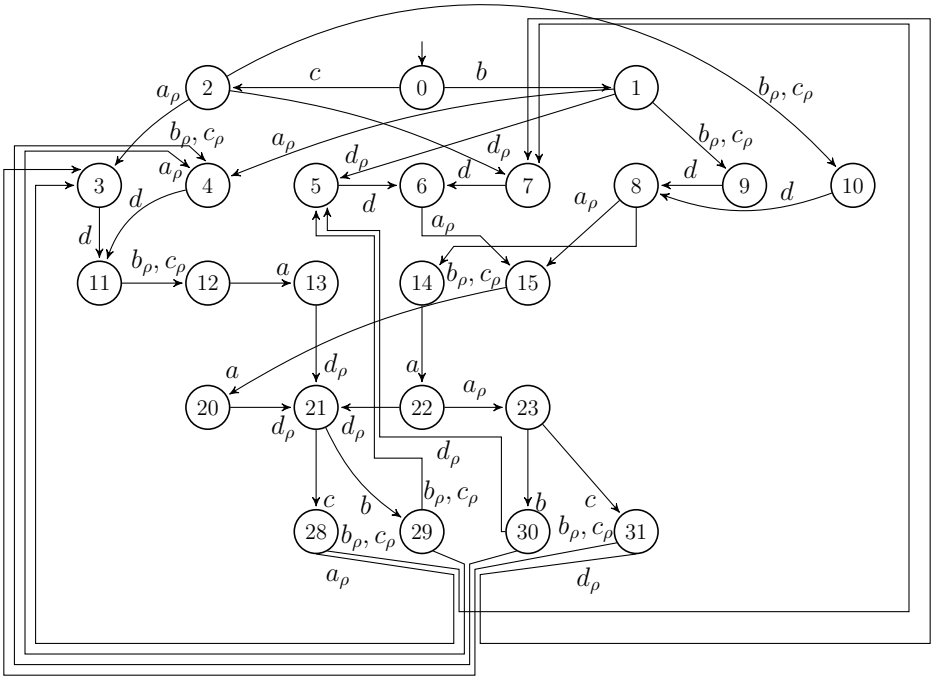


Figure 4.20: Automaton G_O of Example 4.12.

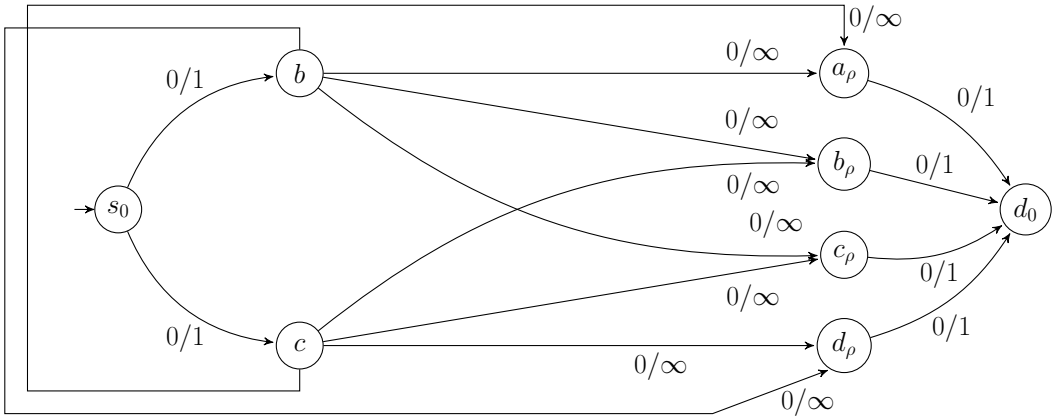


Figure 4.21: Initial configuration of the Edmonds-Karp algorithm applied to choose the events to obtain G_P.

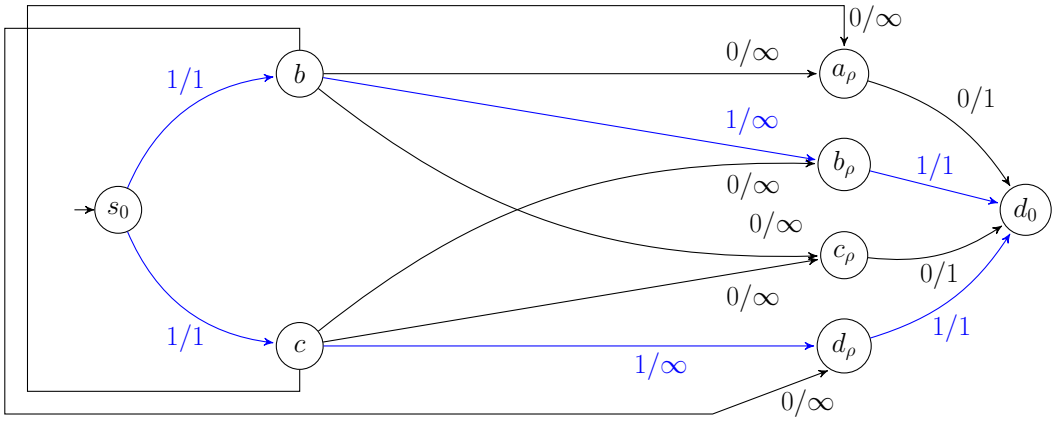


Figure 4.22: Final configuration of the Edmonds-Karp algorithm applied to choose the events to obtain G_P.

Then, in lines 14 to 19 the choice of the events associated with states 1 and 2 is performed. First, the shortest path from s_0 *to* d_0 *must be selected. In this case, there are several paths with the same length. Thus, let us choose* (s_0, F_1, b_ρ, d_0)*. Then, according to line 16, the label of the connection from* s_0 *to* F_1 *is updated to 1/1. According to line 17, the label of the connection from* F_1 *to* b_ρ *is updated to* $1/\infty$*, and, according to line 18, the label of the connection from* b_ρ *to* d_0 *is updated to 1/1. Then, since the label of the connection from* s_0 *to* F_1 *and from* b_ρ *to* d_0 *are* $1/1$*, then it is not possible to connect* s_0 *to* F_1 *nor* b_ρ *to* d_0 *again.*

Now, let us choose the shortest path paths (s_0, F_2, d_ρ, d_0)*, and update the labels according to lines 16 to 18. Figure 4.22 represents how the choices have been made, i.e.,* $\{b_\rho\}$ *was chosen to* F_1 *and* $\{d_\rho\}$ *was chosen to* F_2*. The blue arrows are highlighted to represent the paths already defined.*

This procedure is repeated for the other states $x_{M,1} \in X_{M,1}$ *of* G_O *in which* $|\Gamma_{G_O}(x_{M,1}| > 1$*, i.e., states 21 and 23. Then Algorithm 6 is completed and* G_P *is*

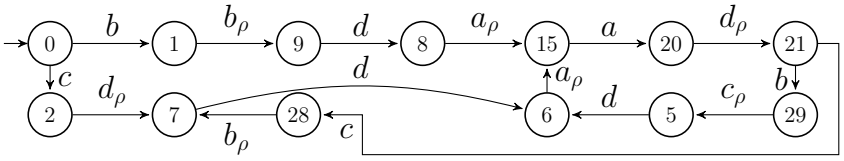


Figure 4.23: Automaton G_P of Example 4.12.

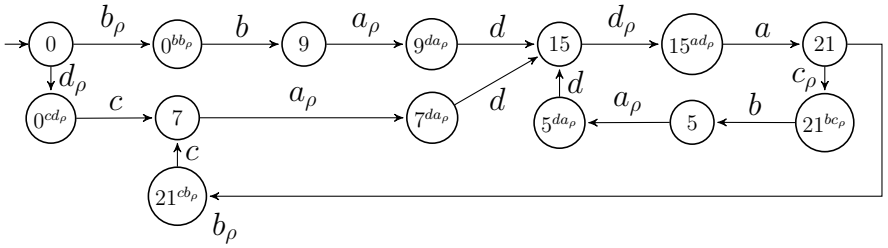


Figure 4.24: Automaton $\bar{G}_P$ of Example 4.12.

obtained, depicted in Figure 4.23.

Let us now compute the automaton $\bar{G}_P$ with Algorithm 8. Note, in Figure 4.24, that $\bar{G}_P$ is a deterministic automaton. Thus, according to Theorem 4.3, the function f_{ER} modeled by G_P is reversible and Conditions **C1**-**C4** *are satisfied. In order to illustrate it, note that after the occurrence of sequence $s = bdacd$, the system reaches the secret state 4, and s is obfuscated as $\tilde{s} = f_{ER}(s) = b_\rho a_\rho d_\rho b_\rho a_\rho$ for transmission. Then, the attacker observes $P_{\Sigma_\rho}^{\Sigma\cup\Sigma_\rho}(\tilde{s}) = a_\rho d_\rho a_\rho$ estimating that the system is in state 1, 2 or 3, while the receiver, using automaton $\bar{G}_P$ recovers sequence $s = bdacd$ and infers that the system is in state 4.* □

In the following, we present the computation of a function f_{ER} that satisfies Conditions **C1**-**C4** for the practical example of the submarine that intends to engage a naval fleet presented in Example 4.1.

Example 4.13 *Let us consider the situation presented in Example 4.1, where the naval fleet behavior is modeled by automaton G, depicted in Figure 4.25. In this example, the states have been renamed as presented in Table 4.1.*

In Example 4.1, an enemy submarine seeks to discover when the anti-submarine ship informs the command center that it is broke down. The communication network is segmented so the anti-submarine ship communicates with the command center through subnet 1, the antiaircraft ship communicates with the command center through subnet 2, and the general application ship communicates with the command center through subnet 3. Thus, the event set is partitioned as $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$, where $\Sigma_1 = \{b_1, r_1\}$, $\Sigma_2 = \{b_2, r_2\}$, and $\Sigma_3 = \{b_3, r_3\}$. It is shown, in Example 4.1, that if the submarine successfully invades subnet 1, he/she can discover when the

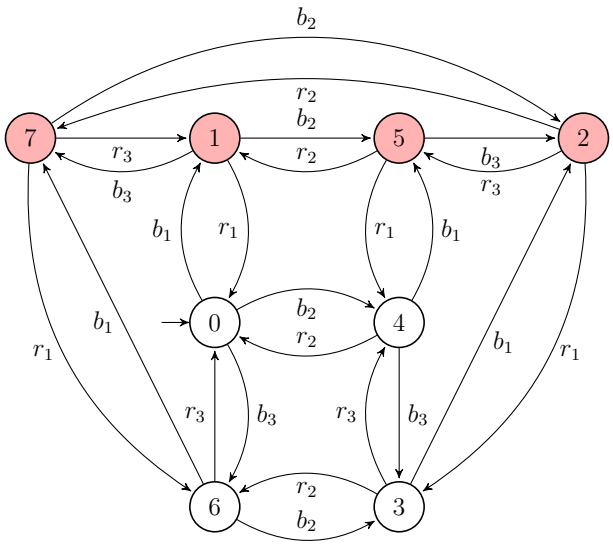


Figure 4.25: Automaton G representing the naval fleet from the external observer perspective. The red states are the secret states.

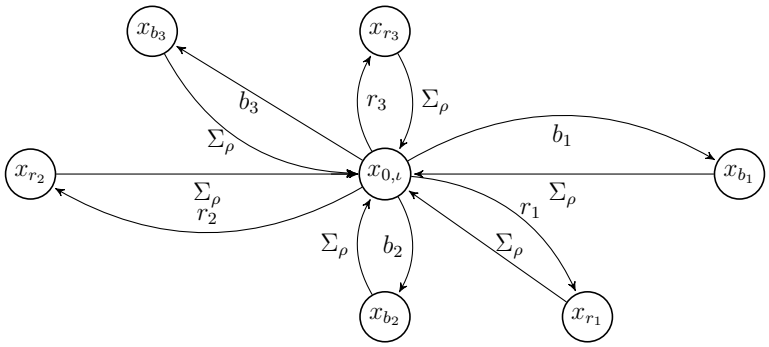


Figure 4.26: Automaton G_ι for Example 4.13.

anti-submarine ship is broke down. Thus, let us compute a function f_{ER} to ensure CSO while also maintain CSU.

To do so, first, we must compute automata G_ι, $G_{\alpha,\rho}$, and G_β. G_ι is computed by following the steps of Algorithm 4 and is depicted in Figure 4.26, where $\Sigma_\rho = \{b_{1\rho}, r_{1\rho}, b_{2\rho}, r_{2\rho}, b_{3\rho}, r_{3\rho}\}$. Automaton $G_{\alpha,\rho}$ is the same as G_α with events b_1 and r_1 relabeled as $b_{1\rho}$ and $r_{1\rho}$, respectively, and is depicted in Figure 4.27. Since the vulnerable subnet, in this example, is SN_1, Automaton $G_\beta = G_{\beta_1}$, depicted in Figure 4.28, is computed by following the steps of Algorithm 5, and in this example, we consider $n_1 = 1$.

Next step is to compute automaton $G_M = G \| G_\iota \| G_{\alpha,\rho} \| G_\beta$, from which all functions f_{ER} that satisfy Conditions **C2** *and* **C3** *can be extracted. In this case, G_M has 160 states and 664 transitions and is omitted. Then, to satisfy, also, Condition* **C1**, *automaton $G_A = Prune(G_M, X_R)$ is computed. In this example, G_A has 96 states and 360 transitions. Then, to also satisfy Condition* **C4**, *G_O must be com-*

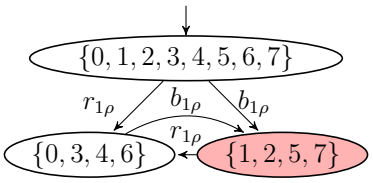


Figure 4.27: Automaton $G_{\alpha,\rho}$ for Example 4.13.

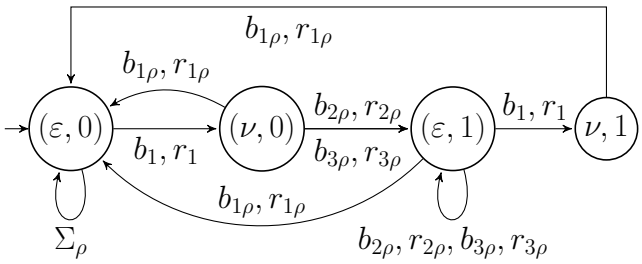


Figure 4.28: Automaton $G_\beta = G_{\beta_1}$ for Example 4.13.

puted by removing the NRP states from G_A, by following the steps of Algorithm 9. In this case, there does not exist any NRP state. Then, consequently, $G_O = G_A$. Finnaly, it is possible to obtain G_P by following the steps of Algorithm 6. In this example, G_P was chosen such that it has 48 states and 72 transitions. In this G_P, after the occurrence of event b_1, the only feasible event is $r_{1\rho}$, after the occurrence of event r_1, the only feasible event is $b_{1\rho}$. In addition, after the execution of b_1, r_2, b_3, and r_3, the only feasible events are, respectively, $b_{2\rho}$, $r_{2\rho}$, $b_{3\rho}$, and $r_{3\rho}$. In this case, automaton $\bar{G}_P$, computed with Algorithm 8, has 48 states and 72 transitions as well. For example, if the general application ship breaks down, get repaired and, in the sequel, the anti-submarine ship breaks down, the events communicated to the command center are $s = b_2 r_2 b_1$. This sequence of events is obfuscated with function f_{ER} as $f_{ER}(s) = \tilde{s} = b_{2\rho} r_{2\rho} r_{1\rho}$. In this case, the enemy submarine observes sequence $P^{\Sigma_\rho}_{\Sigma_{\alpha,\rho}}(\tilde{s}) = r_{1\rho}$ and estimates that the anti-submarine ship has been repaired when it is broke down. Thus, the defense mechanism has protected the naval fleet. □

4.5 Concluding remarks

In this chapter, we have presented a method to allow the intended receiver to recover the original message. In addition, a necessary and sufficient condition to ensure that a function f_{ER} is reversible has been provided, *i.e.*, a necessary and sufficient condition for the receiver to be able to completely recover the sequence of events generated by the plant. Finally, we have presented the methodology to synthesize the obfuscation map, which represents all reversible f_{ER}. In the next chapter we present the next steps in this research.

Chapter 5

Conclusion and Future Works

In this work, we propose to enforce current state opacity (CSO) of the system to protect the system secret, modeled as secret states while using the automaton formalism. In addition, we consider a new notion of utility, called current state utility (CSU), in which the intended receiver must be certain of the current state of the system after the communication of any event. In order to achieve CSO while also ensuring CSU, two strategies have been proposed. The first one is based on network segmentation, where the network segmentation is applied with the aim of enforcing CSO in each of the subnets. Thus, if an attacker invades one of the subnets, the secret is safe. In addition, since the intended receiver gets the information from all the subnets, the intended receiver have all the information to be certain of the system current state, *i.e.*, the CSU is ensured [77]. The second strategy must be used in a segmented network and is based on cryptography and routing control, called multipath event routing. The defense mechanism chooses how to replace an event and through which subnet the replaced event is transmitted in a manner that CSO and CSU are ensured [27].

The strategy to implement the network segmentation with the aim of achieving CSO consists of partitioning the set of observable events such that the system is CSO with respect to each subset of the partition, called partition subset (PS). We defined the problem of finding a partition of the event set, such that the system is CSO with respect to each one of the PS, as the segmented network current-state opacity (SN-CSO) problem. Since the cost of implementation of a segmented network increases with the number of isolated subnets, then it is desirable to find a partition with the minimum number of PS satisfying the SN-CSO problem. Thus, we defined the problem of finding the partition of Σ with the smallest possible number of PS, called the optimal segmented network current-state opacity (OSN-CSO) problem. We also shown that the solution to the OSN-CSO has a high computational cost, thus, we presented a greedy algorithm to compute a sub-optimal solution to the SN-CSO problem. In addition, we presented a sufficient condition for non-opacity of the

system that mitigates the computational cost of verifying the opacity of the system.

For the multipath event routing, we presented a method to obtain an automaton in which all event replacement function that ensures CSO are represented. In addition, a necessary and sufficient condition for the existence of a event replacement function that ensures CSO. Then, we presented a method to obtain an automaton in which all event replacement function that ensures also CSU and a necessary and sufficient condition for the existence of an event replacement function that also ensures CSU.

The main contributions of this work are described as follows:

- We present a greedy algorithm to find a sub-optimal solution for the SN-CSO problem;
- We present a sufficient condition to verify the non-opacity of the system, mitigating the computation os the sub-optimal solution;
- We present a necessary and sufficient condition for the existence of an event replacement function that ensures current-state opacity of CPS with segmented network satisfying Conditions **C1-C3**;
- We present a recovering automaton to allow the intended receiver to be able to recover the original sequence from the transmitted sequence of events;
- We present a necessary and sufficient condition for the existence of a reversible event replacement function that ensures current-state opacity of CPS with segmented network satisfying Conditions **C1-C4**;
- We have implemented the methods in python. The source codes are available in [85].

The results of this thesis have been published or submitted for publication. In the sequel we preset the contribution related to this work.

(i) Network segmentation with the aim to ensure CSO [77];

(ii) Verification of the existence of an event replacement function that ensures CSO [80];

(iii) Synthesis of an event replacement function that ensures CSO [55];

(iv) Synthesis of a reversible event replacement function that ensures CSO [27];

In some cases, the strategy presented in this work may not achieve a reversible obfuscation policy using event replacements. Thus, this raises questions about the possibility of using other obfuscation mechanisms, such as the insertion of events. In addition, other notions of utility may also be addressed, such as diagnosability. If the notion of utility is related to diagnosability, CSU is not necessary, thus, other approaches may be used. Also, in this work, we do not consider delay of transmission nor loss of observation. Thus, it can be addressed in future works.

The next steps of this research can be summarized as follows.

(i) Investigate the insertion of events to enforce current-state opacity with reversible event replacement functions. Propose a defense mechanism that satisfies Conditions **C1**-**C4** using insertion and replacement of events;

(ii) Relaxing the current-state utility and investigate the adoption of event replacement functions to protect systems while the intended receiver remains capable to diagnose the fault occurrences;

(iii) Investigate the use of reversible event replacement functions is systems with delay and loss of observation.

(iv) Investigate the use of event replacement functions to ensure current-state opacity and current-state utility in segmented networks when events can be transmitted through more than one subnet.

References

[1] CASSANDRAS, C., LAFORTUNE, S. *Introduction to Discrete Event Systems*. 2 ed. Secaucus, NJ, Springer-Verlag New York Inc., 2008.

[2] HOPCROFT, J. E., MOTWANI, R., ULLMAN, J. D. *Introduction to Automata Theory Languages and Computation*. 3 ed. Boston, Addison Wesley, 2006.

[3] MIYAGI, P. E. *Controle Programável: Fundamentos do Controle de Sistemas a Eventos Discretos*. Edgard Blucher, 2001.

[4] LAWSON, M. V. *Finite Automata*. Florida, CRC Press, 2003.

[5] DAVID, R., ALLA, H. *Discrete, Continuous and Hybrid Petri Nets*. Springer, 2005.

[6] RAMADGE, P. J., WONHAM, W. M. "The control of discrete event systems", *Proceedings of the IEEE*, v. 77, n. 1, pp. 81–98, 1989.

[7] CAI, K., ZHANG, R., WONHAM, W. M. "Relative observability of discrete-event systems and its supremal sublanguages", *IEEE Transactions on Automatic Control*, v. 60, n. 3, pp. 659–670, 2014.

[8] ALVES, M. V., CARVALHO, L. K., BASILIO, J. C. "New algorithms for verification of relative observability and computation of supremal relatively observable sublanguage", *IEEE Transactions on Automatic Control*, v. 62, n. 11, pp. 5902–5908, 2016.

[9] ALVES, M. V., CARVALHO, L. K., BASILIO, J. C. "Supervisory control of networked discrete event systems with timing structure", *IEEE Transactions on Automatic Control*, v. 66, n. 5, pp. 2206–2218, 2020.

[10] HADJICOSTIS, C. N. *Estimation and inference in discrete event systems*. Springer, 2020.

[11] ALVES, M. V., BASILIO, J. C. "State estimation and detectability of networked discrete event systems with multi-channel communication net-

works", *IEEE Transactions on Automation Science and Engineering*, v. 21, n. 3, pp. 2622–2637, 2023.

[12] DE FREITAS, B. I., TOGUYENI, A., BASILIO, J. C. "Online Representation-based State Estimation of λ-free Labeled Petri Nets", *IFAC-PapersOnLine*, v. 58, n. 1, pp. 72–77, 2024.

[13] SAMPATH, D., SENGUPTA, R., LAFORTUNE, S., et al. "Diagnosability of Discrete-Event Systems", *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, v. 40, n. 9, pp. 1555–1575, set. 1995.

[14] SAMPATH, D., SENGUPTA, R., LAFORTUNE, S., et al. "Failure diagnosis using discrete-event models", *IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY*, v. 4, n. 2, pp. 105–124, mar. 1996.

[15] REIS, L. N. R., MOREIRA, M. V. "Optimal Selection of Subsystems for Synchronous Diagnosis". In: *15th Simpósio Brasileiro de Automação Inteligente*, pp. 1466–1471, Rio Grande, Brazil, 2021.

[16] MOREIRA, M. V., JESUS, T. C., BASILIO, J. C. "Polynomial time verification of decentralized diagnosability of discrete event systems", *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, v. 56, n. 7, pp. 1679–1684, 2011.

[17] GENC, S., LAFORTUNE, S. "Predictability of event occurrences in partially-observed discrete-event systems", *Automatica*, v. 45, n. 2, pp. 301–311, 2009.

[18] JIANG, S., KUMAR, R. "Failure diagnosis of discrete-event systems with linear-time temporal logic specifications", *IEEE Transactions on Automatic Control*, v. 49, n. 6, pp. 934–945, 2004.

[19] BASILIO, J. C., REZENDE, C. H., VIANA, G. S. "Fault diagnosis of discrete event systems modeled by time-interval automaton", *IEEE Transactions on Automation Science and Engineering*, 2025.

[20] REIS, L. N., MOREIRA, M. V. "Computation of synchronous diagnosis bases of discrete-event systems", *Information Sciences*, v. 707, pp. 122031, 2025.

[21] LIMA, P. M., CARVALHO, L. K., MOREIRA, M. V. "Detectable and undetectable network attack security of cyber-physical systems", *IFAC-PapersOnLine*, v. 51, n. 7, pp. 179–185, 2018.

[22] ALVES, M. V., BARCELOS, R. J., CARVALHO, L. K., et al. "Robust decentralized diagnosability of networked discrete event systems against DoS and deception attacks", *Nonlinear Analysis: Hybrid Systems*, v. 44, pp. 101162, 2022.

[23] OLIVEIRA, S., LEAL, A. B., TEIXEIRA, M., et al. "Integrity of cyber-physical discrete event systems under covert actuator attacks", *IFAC-PapersOnLine*, v. 58, n. 1, pp. 198–203, 2024.

[24] OLIVEIRA, S., ANBARANI, M. T., BEAL, G., et al. "Robust recovery and control of cyber-physical discrete event systems under actuator attacks". In: *2025 IEEE 64th Conference on Decision and Control (CDC)*, pp. 1220–1226. IEEE, 2025.

[25] LIMA, P., CARVALHO, L. K., MOREIRA, M. V. "Ensuring confidentiality of cyber-physical systems using event-based cryptography", *Information Sciences*, v. 621, pp. 119–135, 2023.

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

[27] REIS, L. N. R., CARVALHO, L. K., MOREIRA, M. V. "Enforcing Current-State Opacity and Utility of Cyber-Physical Systems Using Multipath Event Routing", *IEEE Transactions on Automatic Control*, pp. 1–8, 2026.

[28] LIMA, P., ALVES, M. V. S., CARVALHO, L. K., et al. "Security of Cyber-Physical Systems: Design of a Security Supervisor to Thwart Attacks", *IEEE Transactions on Automation Science and Engineering*, v. 19, n. 3, pp. 2030–2041, 2022.

[29] OLIVEIRA, S., LEAL, A. B., TEIXEIRA, M., et al. "Security of Cyber-Physical Systems Against Actuator Attacks through Cryptography*". In: *2023 International Conference on Information Technology (ICIT)*, pp. 758–764, 2023.

[30] OLIVEIRA, S., LEAL, A. B., TEIXEIRA, M., et al. "A classification of cybersecurity strategies in the context of Discrete Event Systems", *Annual Reviews in Control*, v. 56, pp. 100907, 2023. ISSN: 1367-5788.

[31] LIMA, P. M., ALVES, M. V. S., CARVALHO, L. "Security Against Communication Network Attacks of Cyb er-Physical Systems", *Journal of Control, Automation and Electrical Systems*, v. 30, n. 1, pp. 125–135, 2019.

[32] SHI, J., WAN, J., YAN, H., et al. "A survey of cyber-physical systems". In: *International Conference on Wireless Communications and Signal Processing (WCSP)*, pp. 1–6, 2011.

[33] BAHETI, R., GILL, H. "Cyber-Physical Systems". In: *The Impact of Control Technology*, pp. 161–166, 2011.

[34] MO, Y., KIM, T. H. J., BRANCIK, K., et al. "Cyber–Physical Security of a Smart Grid Infrastructure", *Proceedings of the IEEE*, v. 100, n. 1, pp. 195–209, 2012.

[35] LEE, J., BAGHERI, B., KAO, H. "A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems", *Manufacturing Letters*, v. 3, pp. 18–23, 2015.

[36] STALLINGS, W. *Cryotography and Network Security.* 4 ed. India, Pearson Education, 2006.

[37] PARLIAMENT, E. "DIRECTIVE (EU) 2022/2555". `https://eur-lex.europa.eu/eli/dir/2022/2555/oj`, 2022. Accessed: 2025-12-10.

[38] JACOB, R., LESAGE, J.-J., FAURE, J.-M. "Overview of discrete event systems opacity: models, validation, and quantification", *Annual Reviews in Control*, v. 41, pp. 135–146, 2016.

[39] LAFORTUNE, S., LIN, F., HADJICOSTIS, C. N. "On the history of diagnosability and opacity in discrete event systems", *Annual Reviews in Control*, v. 45, pp. 257–266, 2018.

[40] BADOUEL, E., BEDNARCZYK, M., BORZYSZKOWSKI, A., et al. "Concurrent secrets", *Discrete Event Dynamic Systems: Theory and Applications*, v. 17, n. 4, pp. 425–446, 2007.

[41] DUBREIL, J., DARONDEAU, P., MARCHAND, H. "Opacity enforcing control synthesis". In: *9th International Workshop on Discrete Event Systems (WODES)*, pp. 28–35, 2008.

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

[43] BRYANS, J. W., KOUTNY, M., RYAN, P. Y. "Modelling Opacity Using Petri Nets", *Electronic Notes in Theoretical Computer Science*, v. 121, n. Supplement C, pp. 101–115, 2005. Proceedings of the 2nd International Workshop on Security Issues with Petri Nets and other Computational Models (WISP 2004).

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

[45] BRYANS, J. W., KOUTNY, M., MAZARÉ, L., et al. "Opacity generalised to transition systems", *International Journal of Information Security*, v. 7, pp. 421–435, 2008.

[46] SABOORI, A., HADJICOSTIS, C. N. "Opacity-enforcing supervisory strategies for secure discrete event systems". In: *47th IEEE Conference on Decision and Control (CDC)*, pp. 889–894, 2008.

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

[48] SABOORI, A., HADJICOSTIS, C. N. "Verification of infinite-step opacity and analysis of its complexity", *IFAC Proceedings Volumes*, v. 42, n. 5, pp. 46–51, 2009.

[49] CASSEZ, F., DUBREIL, J., MARCHAND, H. "Dynamic observers for the synthesis of opaque systems". In: *Automated Technology for Verification and Analysis: 7th International Symposium, ATVA 2009, Macao, China, October 14-16, 2009. Proceedings 7*, pp. 352–367. Springer, 2009.

[50] CASSEZ, F., DUBREIL, J., MARCHAND, H. "Synthesis of opaque systems with static and dynamic masks", *Formal Methods in System Design*, v. 40, n. 1, pp. 88–115, 2012.

[51] TAKAI, S., OKA, Y. "A formula for the supremal controllable and opaque sublanguage arising in supervisory control", *SICE Journal of Control, Measurement, and System Integration*, v. 1, pp. 307–311, 2008.

[52] TAKAI, S., KUMAR, R. "Verification and synthesis for secrecy in discrete-event systems". In: *American Control Conference*, pp. 471–476, 2009.

[53] BEN-KALEFA, M., LIN, F. "Supervisory control for opacity of discrete events systems". In: *49th Anual Allerton Conference on Comunication, Control, and Computing*, pp. 1113–1119, 2011.

[54] JI, Y., YIN, X., LAFORTUNE, S. "Enforcing opacity by insertion functions under multiple energy constraints", *Automatica*, v. 108, pp. 108476, 2019.

[55] REIS, L. N. R., CARVALHO, L. K., MOREIRA, M. V. "Enforcing current-state opacity of Cyber-Physical Systems with multiple channels using event replacements". v. 58, pp. 7–12, 2024. 17th IFAC Workshop on discrete Event Systems WODES 2024.

[56] WU, Y.-C., LAFORTUNE, S. "Synthesis of insertion functions for enforcement of opacity security properties", *Automatica*, v. 50, n. 5, pp. 1336–1348, 2014. ISSN: 0005-1098.

[57] WU, Y.-C., LAFORTUNE, S. "Synthesis of Optimal Insertion Functions for Opacity Enforcement", *IEEE Transactions on Automatic Control*, v. 61, n. 3, pp. 571–584, 2016.

[58] JI, Y., WU, Y.-C., LAFORTUNE, S. "Enforcement of opacity by public and private insertion functions", *Automatica*, v. 93, pp. 369–378, 2018. ISSN: 0005-1098.

[59] KEROGLOU, C., LAFORTUNE, S. "Embedded Insertion Functions for Opacity Enforcement", *IEEE Transactions on Automatic Control*, v. 66, n. 9, pp. 4184–4191, 2021.

[60] KEROGLOU, C., RICKER, L., LAFORTUNE, S. "Insertion Functions with Memory for Opacity Enforcement", *IFAC-PapersOnLine*, v. 51, n. 7, pp. 394–399, 2018. 14th IFAC Workshop on Discrete Event Systems WODES 2018.

[61] LI, X., HADJICOSTIS, C. N., LI, Z. "Extended Insertion Functions for Opacity Enforcement in Discrete-Event Systems", *IEEE Transactions on Automatic Control*, v. 67, n. 10, pp. 5289–5303, 2022.

[62] LI, X., HADJICOSTIS, C. N., LI, Z. "Opacity enforcement in discrete event systems using extended insertion functions under inserted language constraints", *IEEE Transactions on Automatic Control*, v. 68, n. 11, pp. 6797–6803, 2023.

[63] WU, Y.-C., RAMAN, V., RAWLINGS, B., et al. "Synthesis of Obfuscation Policies to Ensure Privacy and Utility", *Journal of Automated Reasoning*, v. 60, n. 1, pp. 107–131, 2018.

[64] JI, Y., LAFORTUNE, S. "Enforcing opacity by publicly known edit functions". In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 4866–4871, 2017.

[65] JI, Y., YIN, X., LAFORTUNE, S. "Opacity enforcement using nondeterministic publicly known edit functions", *IEEE Transactions on Automatic Control*, v. 64, n. 10, pp. 4369–4376, 2019.

[66] WINTENBERG, A., BLISCHKE, M., LAFORTUNE, S., et al. "Enforcement of K-Step Opacity with Edit Functions". In: *2021 60th IEEE Conference on Decision and Control (CDC)*, pp. 331–338, 2021.

[67] DUAN, W., LIU, R., FANTI, M. P., et al. "Edit Mechanism Synthesis for Opacity Enforcement Under Uncertain Observations", *IEEE Control Systems Letters*, v. 7, pp. 2041–2046, 2023.

[68] MAYER, P. C., CABRAL, F. G., LIMA, P. M., et al. "Current-State Opacity Based on State Outputs", *IFAC-PapersOnLine*, v. 58, n. 1, pp. 19–23, 2024.

[69] CARDOSO, J. M., MOREIRA, M. V., CARVALHO, L. K. "Synthesis of an Obfuscation Policy that Guarantees Utility Satisfying a New Privacy Criterion", *IFAC-PapersOnLine*, v. 56, n. 2, pp. 11344–11349, 2023. 22nd IFAC World Congress.

[70] WINTENBERG, A., BLISCHKE, M., LAFORTUNE, S., et al. "A Dynamic Obfuscation Framework for Security and Utility". In: *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 236–246, 2022.

[71] BARCELOS, R. J., BASILIO, J. C. "Ensuring utility while enforcing current-state opacity", *IFAC-PapersOnLine*, v. 56, n. 2, pp. 4595–4600, 2023.

[72] LIN, F., LAFORTUNE, S., WANG, C. "Diagnosability and attack detection for discrete event systems under sensor attacks", *Discrete Event Dynamic Systems*, v. 34, n. 3, pp. 465–495, 2024.

[73] DUAN, W., HADJICOSTIS, C. N., LI, Z. "Obfuscation mechanism for simultaneous public event information release and private event information hiding in discrete event systems", *Information Sciences*, v. 690, pp. 121554, 2025.

[74] LIU, R., LU, J., HADJICOSTIS, C. N. "Opacity Enforcement via Attribute-Based Edit Functions in the Presence of an Intended Receiver", *IEEE Transactions on Automatic Control*, v. 68, n. 9, pp. 5646–5652, 2023.

[75] MAYER, P. C., CABRAL, F. G., LIMA, P. M., et al. "Property-Based Transparency: a New Utility Definition". In: *2024 IEEE 20th International*

Conference on Automation Science and Engineering (CASE), pp. 2825–2831. IEEE, 2024.

[76] MOREIRA, M. V., LIMA, P. M. M., CARVALHO, L. K., et al. "A cryptographic system based on automata for Networked Automation Systems abstracted as Discrete-Event Systems". In: *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*, pp. 1462–1468, 2024. doi: 10.1109/CASE59546.2024.10711639.

[77] REIS, L. N. R., CARVALHO, L. K., BRANDÃO, D., et al. "Network Segmentation with the Aim of Ensuring Current-State Opacity". In: *18th International Workshop on Discrete Event Systems (WODES)*, 2026.

[78] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L. *Introduction to Algorithms*. 3 ed. Massachusetts, MIT-Press, 2009.

[79] WU, Y.-C., RAMAN, V., RAWLINGS, B. C., et al. "Synthesis of Obfuscation Policies to Ensure Privacy and Utility", *Journal of Automated Reasoning*, v. 60, n. 1, pp. 107–131, 2018.

[80] REIS, L. N. R., CARVALHO, L. K., MOREIRA, M. V. "Verificação da existência de um forçador de opacidade de estado atual de Sistemas a Eventos Discretos com múltiplos canais". In: *Congresso Brasileiro de Automática-CBA*, 2024.

[81] HALL, P. "On representatives of subsets", *Journal of the London Mathematical Society*, v. 10, pp. 26–30, 1935.

[82] EDMONDS, J., KARP, R. M. "Theoretical improvements in algorithmic efficiency for network flow problems", *Journal of the ACM (JACM)*, v. 19, n. 2, pp. 248–264, 1972.

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

[84] CAPRARA, A., TOTH, P., FISCHETTI, M. "Algorithms for the set covering problem", *Annals of Operations Research*, v. 98, n. 1, pp. 353–371, 2000.

[85] "Python Code - developed during this work". [online]. Available `https://github.com/LCA-UFRJ-I148/Opacity-in-Segmented-Networks`.