Designing Lightweight Cryptographic Primitives for Securing Industrial Control Systems

Shalini Banerjee

A thesis submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy in Computer Science,
The University of Auckland, 2024.

Abstract

The risk of cyber attacks against Industrial Control Systems (ICS) has marked a significant growth over the past few years. Given ICS has large-scale applications in critical infrastructures such as nuclear-enrichment facilities, oil and gas, etc., the consequences of such attacks have been fatal, leading to damage of critical equipment, economic crisis and loss of human life. Investigation into the high-profile attacks reported against industrial infrastructures indicate that legacy equipment and proprietary protocols are the primary cause behind their evolving threat surface. Quite evidently, the cost-benefit analysis inhibits device-level hardening, and the ICS security practitioners resolve to patch management, compliance-based regimes, and retrofitting IT security protocols with bump-in-the-wire techniques.

In this work, we focus upon the security threats and vulnerabilities that exist in ICS, and emphasize upon the significance of *lightweight cryptographic primitives* in defending against ICS focused cyber attacks. In particular, we perform a detailed analysis of the attack progression framework, along with the threat surface exploited to deliver the attacks. Thereafter, we identify the security essentials and design lightweight cryptographic implementations to increase adversary's cost of attack at various stages of the attack framework. In particular, we design (a) *a framework that accurately identifies the manipulation of process parameters without compromising the network bandwidth used for communicating control information*, (b) *a framework that aims to achieve application-specific security while incurring significantly low overhead*, and (c) *a platform that employs cryptographic obfuscation to prevent extraction of process semantics*. Our obfuscation platform takes inspiration from our construction of (d) *an efficient virtual black-box obfuscator for binary decision trees*. Finally, we design (e) *verifiable schemes for defending against malicious obfuscators that incorporate trigger-based initiation of malicious payloads to manipulate critical control parameters*. The last two constructions are of independent interest, and have much wider applicability in software obfuscation.

Dedicated to my parents - Ajoy and Chameli.

Acknowledgements

In the following, I attempt to express my acknowledgements towards all those who have been a part of this memorable journey.

To begin with, I express my deep sense of respect and gratitude towards my supervisors *Prof. Giovanni Russello* and *Prof. Steven D. Galbraith.* I convey my most heartfelt thanks to them for their invaluable guidance in every aspect of academic life, assisting with the necessary setup for implementing the proposed approaches, bringing out in me the analytical and critical perspective towards conducting research and giving me the freedom to pursue my own interests. I consider myself fortunate to have the opportunity to work with such wonderful persons.

I would like to extend my gratitude to *Dr. Tariq Khan* for helping me implement the prototypes of some of the proposed approaches.

I would like to thank *Emeritus Prof. Gosta Pada Biswas* from Indian Institute of Technology, Dhanbad, who has been my inspiration behind pursuing research.

I am profoundly grateful to *Ram Venkatesalu, Jianli Bai and Judith Perera* for making Auckland a *home away from home* for me, extending their support whenever and wherever possible. I consider myself lucky to find best friends in them. I cannot thank enough my colleagues *Zhijie Li, Lukas Zobernig* and all the PhD students in the School of Computer Science at the University of Auckland for making my office the most warm, welcoming and engaging place.

I express my acknowledgement towards my family members who are the warmest persons I have known *Tapas Mukherjee, Tushar Mukherjee, Mahua Mukherjee* and *Shirini Banerjee* for their endless support and faith in me.

I thank my parents for always believing in me and supporting me pursue my dreams through all thick and thin. *Without your kind words and encouragement, this journey would not have been possible. I dedicate this thesis to you, with all my love and respect.*

Finally, I thank the Marsden Fund of the Royal Society of New Zealand for funding my stay in this beautiful country.

Contents

List of Figures

List of Tables

List of Acronyms

AEAD Authenticated Encryption with Associated Data

AES Advanced Encryption Standard

ANSI American National Standards Institute

CI Critical Infrastructures

CISA Cybersecurity and Infrastructure Security Agency

CRC Cyclic Redundancy Check

CPU Central Processing Unit

DTLS Datagram Transport Layer Security

GE Gate Equivalent

IEC International Electrotechnical Commission

ICS Industrial Control Systems

ISA International Society of Automation

ISO International Organization for Standardization

IT Information Technology

TCP Transmission Control Protocol

TLS Transport Layer Security

PERA Purdue Enterprise Reference Architecture

PDU Protocol Data Unit

PLC Programmable Logic Controllers

RAM Random Access Memory

ROM Read Only Memory

OT Operational Technology

SCADA Supervisory Control And Data Acquisition

Chapter 1

Introduction

Industrial Control Systems (ICS) have incorporated a distributed monitoring and control paradigm into the industrial objectives of production and manufacturing, with implementations spanning critical infrastructures, such as nuclear enrichment facilities, power grids, oil and gas, water treatment and distribution, telecommunication, transportation, process manufacturing and discrete manufacturing industries [50, 186]. With the advent of Industry 4.0, there has been a huge impact on global production and supply networks of ICS through large-scale machine-to-machine communication. Unsurprisingly, the integration of the physical processes with embedded components over a layered plant-wide network, further connected to an enterprise network through IT/OT[1] vertical integration support, has magnified the ICS threat-surface, where an adversary, by exploiting the digital security vulnerabilities, can disrupt the normal operations of the physical processes. The consequences of these disruptive events are fatal, leading to economic crisis, damage of critical equipment, loss of human life, etc. [106, 136, 196]. Exploration and examination of the infamous ICS attacks are necessary in comprehending a "complete kill chain" approach to the strategies taken towards executing such influential disruptive events to safety-critical systems, such that solutions could be formulated and implemented for defending against these attacks.

1.1 Proprietary Industrial Control Framework

In this section, we discuss the limitations that restrict industrial control applications from achieving the security goals required to defend against influential cyber

[1] OT comprises of software and hardware modules that control physical processes in industrial infrastructures.

attacks.

The distributed nature of industrial control is enabled through legacy controllers, that regulate real-world processes through pre-configured control logic on top of proprietary ICS networks [186]. The embedded engineers have attributed availability and time-criticality as central to the functioning of the safety-critical systems, disregarding the need for implementing the security essentials [33]. Though the traditional "air-gap" principle, along with poorly documented technologies have protected ICS by a security-through-obscurity approach, the requirement of monitoring and advanced control via increased connectivity to corporate and IT networks, along with the Internet, have exposed the insecure control framework to the risk of cyber-threats.

The industrial controllers are legacy devices with no provisions for built-in-security [156, 77]. In particular, an adversary can use the documented features to change the control behavior, leading to unpredictable and unstable responses in safety-critical applications. Moreover, majority of the controllers have limited supply of memory and computational capabilities for accommodating security implementations [173]. Finally, the vendors do not incorporate security essentials during design and development of the industrial equipment [77], and restrict access to the memory and implementation details of the proprietary hardware architectures [167].

Furthering on the insecurities in industrial control, the ICS specific network protocols specialize in features, such as real-time control, noise immunity, and reliability in harsh environments, and do not include provisions for security [33, 97, 194]. Interestingly, industrial communication protocols such as Modbus [189], DNP3 [84], EtherNet/IP [82], etc. carry critical control and configuration information in clear text [97]. More specifically, these application-specific protocols do not have provisions for imparting confidentiality, integrity and authentication to the critical control communication, and as such, modifications done to control and configuration parameters via network access cannot be detected by the monitoring units.

Given that the ICS products are not inherently resilient towards cyber-attacks, the ICS research community and industrial security practitioners recommend predictive passive intrusion detection and prevention techniques or authentication/encryption implementations on commercial-off-the-shelf (COTS) products,

using bump-in-the-wire techniques [28, 48, 194, 212]. However, the predictive techniques for detection and defence are inaccurate, and the cryptographic recommendations are too ambitious in complying with the real-time requirements of generic time-invariant critical applications (depending on the sampling and response rates for ICS field devices [3]). Finally, efforts from ICS vendors to integrate cyber-security directly into ICS equipment is an expensive alternative for customers who have a number of ICS devices deployed at their production site [216].

1.2 Evolving Threat Surface in ICS

In this section, we investigate the reported high-profile cyber-attacks in ICS space to bring forward their distinguishing aspects. In particular, we aim to analyze adversary's intent, pre-requisites for achieving attack objectives, tools and techniques employed to deliver the attacks, and severity of the attacks. We argue that such explorations are necessary in both understanding future attacks trends against industrial infrastructures, as well as designing strategies and roadmaps to safeguarding the infrastructures.

1.2.1 Taxonomy of ICS Attacks

While the digital transformation of industrial infrastructures is appealing, analysing the cyber threats of growing complexity and severity is important in identifying tools, techniques and procedures employed for delivering and executing ICS disruptive events.

The threats in ICS space could be basic, such as generic phishing scams against industrial infrastructures with no security implementations, to Advanced Persistent Threats (APTs)[2] such as data extraction or extortion using custom tools and techniques [216]. The "Industrial Control System Cyber Kill Chain" [9] categorizes *targeted attacks* to be the most sophisticated and impactful attacks against an ICS [121, 115], and as such our study focuses on analyzing and strengthening defence against this class of attacks. In what follows, we provide an abstraction of the generic attributes of reported targeted attacks against industrial facilities.

[2]NIST defines APTs as threats where adversaries possess sophisticated levels of expertise and significant resources which allow to create opportunities for achieving their objectives using multiple attack vectors [174].

Targeted Attacks. Attacks against industrial control infrastructures, characterized by the following features:

- **Purpose of Attack.** Adversaries are motivated by the objective of performing either of the following:
  - **Process Manipulation**: ICS security community identifies them as *false-data injection*, where an adversary overrides the desired control values leading to unstable responses in the control application, with the most impactful consequences [159].
  - **Cyber Espionage**: This class of attacks target theft of intellectual property or monetary gain through extortion [216].
- **Evidence of Target Selection.** Attacks show a clear evidence of selection of target control parameters within the control infrastructures, exploiting which can cause predictive outcomes, catered to the requirements of the adversaries [192].
- **Knowledge of Process Control.** Adversaries perform a campaign of efforts to gain pre-requisite knowledge of process control and engineering designs of the target control application.
- **Attack Methodologies.** Attack methodologies are unique, advanced and specific for the victim ICS. Adversaries spend considerable amount of budget and time in developing zero-day tools for the specified targets.

1.2.2 Infamous ICS Attacks

In this section we explore the high-profile targeted attacks reported against critical infrastructures in order to understand the evolving trend in adversary tradecraft and its implications.

Maroochy Shire Sewage Spill Event [201] in 2000 at Queensland in Australia demonstrated the capability of a knowledgeable adversary in releasing millions of gallons of untreated sewage in local parks and rivers by modifying control commands via unauthorized access to the insecure control network between controllers and monitoring units.

The Stuxnet malware [106] uncovered by the Kaspersky lab demonstrates the power of a resourceful and knowledgeable adversary in sabotaging 1000 centrifuges of a Natanz nuclear-enrichment facility of Iran in 2010. Manifesting

sophisticated capabilities of four zero-day exploits, Windows rootkits, first discovered PLC rootkits, peer-to-peer communication with infected devices, Stuxnet is considered to be the first targeted cyber weapon against an industrial facility [190].

Havex/Dragonfly [22] targeted utilities, transport and manufacturing systems, extracting sensitive information using ICS network protocols. As per Kaspersky report [171], the motivation behind installing the malware was solely industrial espionage. In 2011, Duqu [206], a malware similar to Stuxnet, infected the control applications to gather intelligence on the process control design.

German Steel Mill Event [116] in 2014, indicates the massive damage caused to a steel plant by modifying control parameters, that prevented a blast furnace from shutting down. The attackers initially gained access to the enterprise network, and worked their way through the production network, causing multiple failures of control systems.

BlackEnergy malware [65] caused massive outages in three regional electricity distribution management systems of Ukraine in 2015. The adversaries gained knowledge on the operational semantics of the process, along with the engineering design of the control framework by infecting the corporate enterprise network. The malware is also reported to have infected other critical infrastructures of Ukraine.

Crashoverride malware [184] took down an electrical distribution equipment at Ukraine in 2016, where the adversary gathered knowledge of the victim ICS through network protocols, codified them in software and, finally enabled it to disrupt the process control. The malware execution occurred in multiple stages, starting with gathering knowledge on the control behavior of the application, identifying the control parameters to be modified, initiating unauthorized connection to controllers to manipulate the identified parameters. Unsurprisingly, subsequent detection and defence suffered due to limitations on deployment of security services within the victim ICS environment [184].

Trisis [105], the first known malware targeting safety instrumentation systems, caused a massive shutdown of a petrochemical plant in Saudi Arabia in 2017. The malware framework included a rootkit designed to change control parameters in

Schneider Electric's Triconex safety instrumented system (SIS), with abilities to codify the potential unsafe states, design multiple conditions on safety tolerances for causing massive shutdowns, and to death of individuals [184].

The industrial infrastructure of a water treatment and distribution system in Florida was hacked in 2021, where the adversaries modified the level of sodium peroxide in drinking water by exploiting the poor digital security controls [80].

Chernovite's Pipedream [71], the seventh-known ICS-specific malware identified and analysed by Dragos in 2022 and yet to be employed, is a modular attack framework designed to cause disruption, degradation and destruction to target control environment. The malware is a collection of utilities, including tools for reconnaissance, manipulation and disruption of wide-variety of controllers and ubiquitous industrial technologies. Pipedream enumerates an industrial environment, and utilizes controller implants to execute untrusted code, with the capability to hide its existence for years.

Along the line of events, other targeted attacks, such as [144, 222] contribute to our strong inference on the advances in ICS-specific capabilities of adversaries in codifying attack methodologies for causing disruption and destruction of critical utilities. Reports by Dragos show an alarming growth in ICS targeted attacks, with vulnerabilities doubling in 2021 compared to 2020 [72]. As per the 2022 X-Force Threat Intelligence Report [80] and other sources [145], *critical infrastructures are the primary target* of cyber-adversaries, who are more inclined towards targeted attacks, compared to low-budget script-kiddie attacks. Gartner predicts that 30% the mission-critical control systems will experience massive shutdowns by 2025, with cyber-adversaries weaponizing OT environments to successfully harm and even kill individuals [61].

1.2.3 Analysing Attack Methodologies

Though there has been a noticeable shift towards commodified technologies for intrusion techniques in industrial infrastructures, breaking down the components of the modular attack framework enables an expanded view of the attack progression.

Modular Attack Framework. A targeted attack in an ICS is a *process with inter-linked modules*, executing all of which renders an adversary in achieving

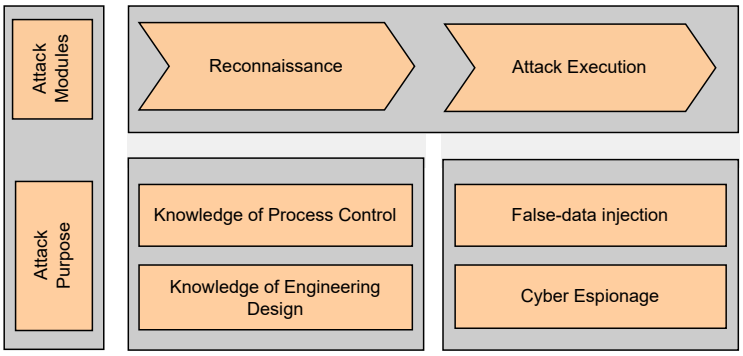


Figure 1.1: Modular attack framework for Targeted Attacks.

predictive and desired outcomes. Understanding how these modules work is vital to designing solutions that increase adversary's cost of attack. Such an attack framework incorporates an adversary who has an understanding of the elements of a generic process control paradigm, and has tools and techniques for interfacing and interacting with ICS equipment and/or protocols. The attack framework consists of *two distinct* modules:

(a) **Reconnaissance**: By executing this module, an adversary extracts operational semantics of the target control application, and corresponding engineering designs and practices employed at the plant site. This helps in learning the control parameters central to the safe functionality of the application, such that tactics and techniques could be designed for disruption and destruction of the facility, or for gaining an in-depth knowledge of process control for cyber-espionage campaigns.

(b) **Attack Execution**: By executing this module, an adversary delivers his capability, which could be Intellectual Property theft for political or economic reasons, or manipulation of critical control parameters (*false-data injection*). This module may include sub-modules for enabling other support functionalities that trigger necessary conditions needed for modifying the parameters.

The efforts of an adversary in gaining accurate and in-depth knowledge of process control and delivering custom-made exploitation are simplified due to insecure nature of the legacy controllers and proprietary protocols (see Section 1.1). Also,the adversarial attempts to eavesdrop or modify critical control data over the insecure

network cannot be detected by the computers employed to monitor the control activities.

From the above discussions, it can be safely conjectured that security controls in industrial infrastructures need to address the following:

(i) Prevent an adversary from extracting the operational semantics of target control application.

(ii) Prevent an adversary from manipulating process control parameters of target control application.

(iii) Prevent an adversary from evading detection by the monitoring units deployed at the target control facilities.

1.3 Implementing Security Controls

As discussed in Section 1.1, securing industrial infrastructures from targeted attacks bears its challenges from the use of legacy ICS products and limitations in employing cryptographic implementations due to time-invariant nature of the applications. Furthermore, limited system resources and diversity in configurations of specialized industrial systems restrict designing all-in-one solutions. Finally, the differences in how similar technologies are employed in ICS and enterprise IT environments, and the corresponding complexity penalties restrict the wholesale import of IT security technologies [184].

The high demand of availability and being uptime have pushed the ICS security community towards a "detection-over-prevention" approach for minimizing complexity penalties [136]. In particular, emphasis is given upon detecting adversarial techniques for enumeration, data-gathering and process manipulation using IDS and machine-learning models with signature and anomaly based predictive techniques [90]. However, these methodologies are inaccurate and inadequate against zero-day exploits and other custom-made techniques of adversaries.

In addition, the state-of-art recommendations for detection and defence with cryptographic implementations are not suitable for complying with the latency requirements of the target control application.

The other important concern is that the existing solutions for imparting integrity, authentication and confidentiality are not suitable for achieving application-specific security required at the target control environment. More specifically, the current recommendations rely upon the security services offered at the *transport, network and data-link* layers. A typical control framework might have application-specific security requirements such as *'provide confidentiality/integrity/authentication if read/write request about valve v is being communicated '*. Implementing protocols such as TLS 1.3, IPSec introduce unacceptable overhead, with mandatory encryption of all control parameters, and unwanted features such as AEAD, perfect secrecy, etc. which are not a defacto requirement in generic ICS facilities.

Finally, ICS security community has *no* implementations for preventing adversarial attempts of extracting operational semantics of target application. The risk-assessment matrices calculated by the individual organizations categorize targeted attacks as disruptive events with *lowest probability* and *highest damage potential*, prioritizing designing strategies for mitigating high probability low damage potential events [121].

SANS 2022 survey indicates the importance attributed to increased cyber-security consultation for designing security policies for OT environments, regular security assessment within control infrastructures and increased cyber-security awareness program for employees [101, 153]. With an aspiring ambition of Industry 5.0, which visions both societal and industrial progress, countermeasures to defend against targeted attacks should not be limited to approaches that address people and policies. Emphasis should also be upon reconsidering security essentials with custom-made implementations that rely upon *lightweight cryptographic primitives*[3]. Finally, ICS security practitioners and research community need to be mindful of the fact that solutions for defending against targeted attacks should be aimed at a holistic end-to-end approach, one that addresses the diverse security goals (see Section 1.2.3) essential for effectively dealing with identification, prevention and mitigation of malicious actions of ICS adversaries.

[3]Cryptographic primitives that incur low computational complexity, and are particularly well-suited for implementations limited by size, power-consumption and processing-speed [76].

1.4 Objectives of the Research

This research identifies a comprehensive approach towards defending against targeted attacks, with methodologies that rely upon rigorous cryptographic primitives. The overall objective of this study is given as follows:

(1) Design tools and techniques for **accurate detection of false-data injection attacks** within ICS, such that the proposed solutions comply with the latency requirements of the target control environment.

(2) Design tools and techniques for **achieving application-specific security in detecting and preventing false-data injection attacks**, while incurring low computational complexity.

(3) Design tools and techniques for **preventing extraction of operational semantics** of target control application.

1.5 Main Contributions

To achieve the objectives discussed in Section 1.4, we make the following contributions:

- We design an efficient tamper-detection framework that accurately identifies manipulation of process control parameters without compromising the network bandwidth required for critical control communication.
- We design a general-purpose modular framework that considers security implementations at the application-level. In particular, we propose a technique that prevents false-data injection attacks by encrypting a set of ICS payloads that are critical to the control behavior of the application, while incurring minimal overhead compared to the state-of-art methodologies. Our framework helps envision how minimal application-level access to the vendor-specific controller devices could enable the customers define their respective benchmarks for security. Our methodology applies to achieving *integrity*/*authentication* in communicating critical ICS payloads.
- We identify an adversary who aims at extracting the operational semantics of the target control application through static analysis of control program. Our goal is to prevent such efforts of the adversary by making use of cryptographic obfuscation, a tool that transforms a program to its semantically

equivalent counterpart, such that access to the transformed version does not give away the assets (secret values within a program). To this end, we formalize the abstraction of control programs, and define its *assets*, the secret values in the program that give away the operational semantics of the process. Since control programs function as the "decision-making layer" of the process control framework, we employ binary decision trees to develop a structured representation. Our formalism is generic, and applies to all types of control programs implemented in industrial environment. Following this, we design an *efficient virtual-black box (VBB) obfuscator for binary decision trees*, and use the random oracle paradigm to analyze the security of our construction.

- We design a general purpose platform that makes use of cryptographic obfuscation to prevent reverse-engineering of control programs. We take inspiration from our construction of VBB obfuscator for binary decision trees, which we leverage to design a lightweight practical solution for obfuscating control programs that prevents extraction of process semantics.
- We discuss the theoretical possibility of a malicious obfuscator that causes an obfuscated program to have some undesirable behaviour. Obfuscating control programs with a malicious obfuscator could trigger conditions to manipulate a critical control parameter, leading to fatal consequences. We show that some of the published obfuscation techniques in the theoretical literature can have this malicious functionality. Next, we propose a new definitional framework for *verifiable obfuscation*, which aims to provide security against malicious obfuscators, and follow this with proposals of verifiable schemes for a number of existing cryptographic constructions, that *either apply to control programs or are subjects of independent interest.*

1.6 Organization of the Thesis

The thesis is organized into *nine* chapters, an overview of which given as follows:

Chapter 1 provides an introduction to the research problem addressed in this thesis, and the motivation behind this work. In particular, it highlights the legacy design of ICS that is inextricably linked to insecure equipment and protocols, modular attack framework of high-profile targeted attacks reported against industrial infrastructures, preparedness of industry in defending against such attacks

and, finally the objectives of this research, along with an outline of the main and specific contributions made towards addressing the research problem.

Chapter 2 provides a general description of the key components of an industrial process control framework, an in-depth analysis of the insecurities in industrial equipment and protocols, and the threat surfaces exploited to cause targeted attacks against industrial infrastructures. Following this, the chapter gives a detailed overview on the security essentials and controls adopted to prevent, detect and deter targeted attacks. Finally, the chapter discusses the cryptographic ciphers used in our implementations, with a special focus on program obfuscation.

Chapter 3 looks into the efforts by ICS security research community in detecting and preventing targeted attacks against industrial infrastructures. This is followed by a broad-level discussion on the state-of-art in cryptographic obfuscation.

Chapter 4 presents the first of our contributions TaDeT, a general-purpose tamper detection framework that accurately detects tampering of control data without compromising the bandwidth for critical control communication.

Chapter 5 introduces SelEnc, a selective encryption framework that aims to achieve security at application-level within a process control system.

Chapter 6 introduces our efficient VBB obfuscator for encoding evasive binary decision trees, which also includes the description of our new cryptographic primitive for encoding parameters in an interval-membership problem.

Chapter 7 presents a formalization of abstraction of control programs and its assets, introduces a new threat model (to the best of the authors' knowledge) and finally proposes ObfCP, a legacy-compliant platform for preventing reconnaissance of process control. As far as we can tell, this is the first attempt to prevent extraction of process semantics in industrial control applications.

Chapter 8 introduces the notion of *malicious obfuscation*, and proves the existence of undetectable malicious obfuscators for a number of obfuscation schemes in the theoretical literature. This is followed by formulating *verifiable obfuscation*, along with a proof-of-concept of the developed notions.

Chapter 9 presents an overall summary of the key findings and limitations of this research, and details on the possible future directions that emerge from this context, to be undertaken as a future initiative by the authors of this work.

1.7 List of Publications

Some of the work detailed in this thesis contributed to publications that have been accepted or, are currently in the submission process:

1. Banerjee, Shalini, Tariq Khan, John H. Castellanos, and Giovanni Russello. "Selective Encryption Framework for Securing Communication in Industrial Control Systems." In ICC 2023-IEEE International Conference on Communications, pp. 4125-4130. IEEE, 2023. - *The details are included in* **Chapter 5**.
2. Banerjee, Shalini, Steven D. Galbraith, and Giovanni Russello. "Obfuscating Evasive Decision Trees." In International Conference on Cryptology in India, pp. 3-20. Cham: Springer International Publishing, 2018. - *The details are included in* **Chapter 6**.
3. Banerjee, Shalini, Steven D. Galbraith, Tariq Khan, John H. Castellanos, and Giovanni Russello. "Preventing Reverse Engineering of Control Programs in Industrial Control Systems." In Proceedings of the 9th ACM Cyber-Physical System Security Workshop, pp. 48-59. 2023. - *The details are included in* **Chapter 7**.
4. Banerjee, Shalini, and Steven D. Galbraith. "Auditable Obfuscation." Cryptology ePrint Archive (2023). - *The details are included in* **Chapter 8**.

Chapter 2

Preliminaries

This chapter provides a detailed description of control flow execution in generic ICS facilities, along with the key components necessary for implementing a process control framework. Following this, we discuss the various threat surfaces exploited by a cyber adversary in bringing about targeted attacks. We study in detail the insecurities of industrial controllers and communication protocols, and the defense and policy responses adopted by the industry for combating targeted attacks. We take a deeper look into the security goals that the ICS practitioners prioritize while implementing security controls, keeping in mind the real-time constraints of control applications. Finally, we survey lightweight cryptographic primitives, with a special focus on cryptographic obfuscation, a good understanding of which is necessary to get introduced to our constructions.

Without loss of generality, we use the term 'process' to refer to industrial processes, that are a systematic series of physical, chemical, mechanical, or operations of similar kind, to produce a result [182]. *By 'state' of a process, we mean the various control values defined for the actuators within a process* [92].

2.1 Control System Architecture

ICS is a collective term used to refer to diverse control system types and associated instrumentation, including devices, networks and controls required to operate and/or automate processes. Modeling a process control framework is guided by the aim of achieving a distributed nature of monitoring and control over the *state* of a process.

The key elements of industrial control are: (a) control loop (b) SCADA monitoring units, remote diagnostics and engineering workstations, and (c) plant communication networks [186].

Control loop comprises of PLC and the field devices (sensors, actuators) connected to the process. Sensors sense analog or digital values such as distance, proximity, level, pressure, temperature, flow, magnetic sensors, LED lights, push button signals, etc., and actuators operate on control elements, such as pumps, valves, switches or motors. PLCs are the programmable industrial computers containing pre-configured control programs that act as the decision-making layer for the target control application. PLCs collect process measurements from the sensors in form of process variables, execute control program, and output manipulated variables for controlling the state of the process [4]. This operation is carried out in repeating cycles at discrete time intervals to preserve the real-time requirements of the process, and is referred to as *scan cycle* of the control application. A summary description of the process variables and manipulated variables is given in Section 2.2.1.

Most basic control systems define four main components featuring a control loop [4]:

(i) A measurement of the state of the process which is collected through sensors

(ii) A PLC computing an action based on comparing measured value against a preset or desired value, usually called a *setpoint* (embedded within a control program)

(iii) An output signal resulting from the computations performed by PLC, which manipulates the process action through some actuators

(iv) The process itself reacting to this signal, and changing its state or condition

PLCs connect to SCADA units for advanced control and monitoring, where Human Machine Interface (HMI) programs allow a process engineer to make adjustments to the physical processes by overriding the manipulated variables output by PLC. The PLCs are programmed by engineering workstations, which provides IEC 61131-3 [99] compliant Integrated Development Environments (IDEs) for developing control programs, which are transferred to the PLCs using a procedure called *program download*.

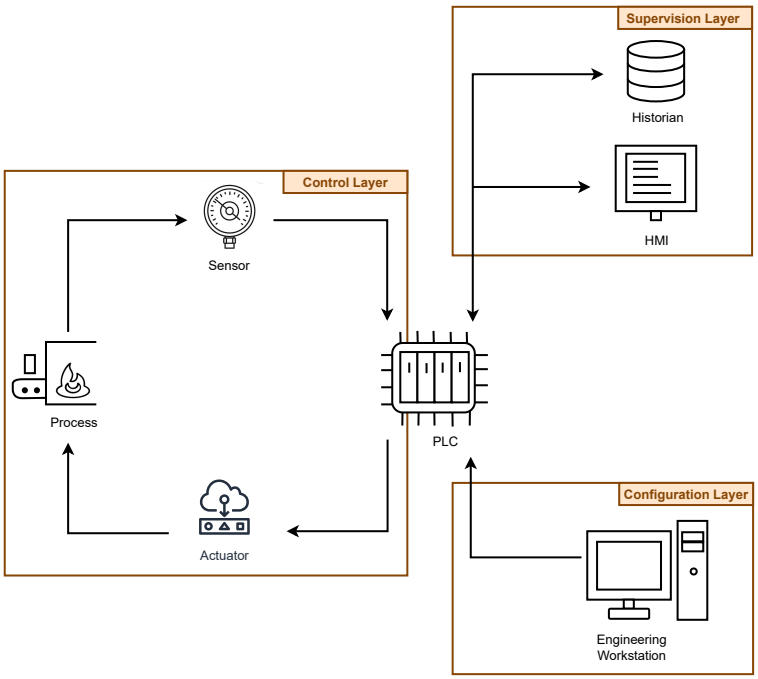


Figure 2.1: High level schematics of ICS distributed process control framework.

The complex nature of distributed control is accomplished through integrating computational units and physical processes, and deploying them over industrial communication protocols. Based on functionality and control, *we identify three distinct layers* constituting a process control framework: (a) *control layer* constituting the control loop, (b) *supervision layer* consisting of SCADA command, monitoring and maintenance facilities, and (c) *configuration layer* involving software development for PLC. The high-level schematics of control, configuration and monitoring in an ICS framework is given in Figure 2.1.

2.2 Overview of Key Components

In this section, we present an elaborate discussion on the key components of a generic process control framework.

2.2.1 Field Devices

Field devices are the industrial equipment that communicate control I/O signals between a process and PLC. Two of the most important signals used in process control are as follows:

(a) **Process Variables** - In industrial process control, a process variable is measured by an industrial equipment called *sensors*, and acts as input to PLC. Precisely, a sensor converts physical, biological and chemical quantity into electrical signals that can be measured or interpreted, and sends them to the PLC.

(b) **Manipulated Variables** - The variable that needs to be manipulated in order to have control over a process variables is called a manipulated variable. *Actuators* are transducers that convert electrical signals acquired from the PLC to mechanical motion, and communicate them to the process to manipulate its state.

The field devices can communicate both analog and discrete signals. Analog signals are continuous values, such as pressure or temperature. Analog sensors produce voltage signals or current signals, which is proportional to the quantity being measured, and the value perceived by the PLC depends on its configuration. For example, some Allen Bradley SLC PLC models return any integer between 0 to 8191 for an input of 0 to 5 volts. Examples are flow transducers, humidity sensors, load cell sensors, potentiometers, pressure transducers, vibration transducers and temperature sensors. Analog output interfaces translates the discrete values of the PLC into continuous analog output signals. Examples include modulating control valves, chart recorders, motor drives, analog meters, etc. For instance, say a PLC decides to send a 50% open command to a control modulating valve, this command signal is transmitted in form of a 4-20 mA DC electric current.

Discrete sensors send high/low signals to PLC corresponding to some physical process parameter. Examples include limit switches, proximity switches, photoelectric sensors, etc. Discrete actuators are usually used to turn on/off motors, wipers, pistons, heating elements, etc. and examples include magnetic valves, solenoids, relays and contactors, etc.

2.2.2 Programmable Logic Controllers

PLCs act as the nucleus of industrial automation controlling a multitude of processes of varying levels of complexity. PLCs are specialized computers preferred in industrial environment due to their multiple input-output arrangements, resistance to high temperature and electrical noise, capability of executing real-time tasks with high efficiency, electromagnetic interference and mechanical vibrations.

Some of the major advantages of using PLCs over PCs and COTS devices are as follows:

- Easy to connect to different hardware modules.
- Easy replacement of malfunctioning circuits.
- Easy to program and re-program.
- Easy to maintain and repair.
- Shelf life of at least 20 years.
- Real-time control software.
- Capable of acting as an edge device, i.e. PLC connects to field devices over the control layer, SCADA/HMI devices over the supervision layer and engineering workstations over the configuration layer.

PLCs are microprocessor-based computer units that controls multi-variable processes through the coordination of the following modules: (a) processor, (b) I/O modules, and (c) power supply. The processor contains CPU and memory. A typical processor implements the logic and controls the communication among different modules. In addition, it contains at least one interface to the engineering workstation and many interfaces to remote I/O and other communication modules for connecting with HMIs. The controller memory is divided into *program* and *data* memory. The program memory contains the control program, and accounts for most of the memory of the device. A typical Allen Bradley ControlLogix 1756-L74 assigns 16 MB for housing the control program, while as little as 0.98 MB is allocated for storing the rest. The data memory stores I/O signals acquired through the field devices in *process image tables*, status of timers, counters, data storage, firmware, system administration modules, etc. The I/O modules condition the signals from the sensors and actuators and does the interfacing with the process image tables. As programmable devices, PLCs offer the flexibility for re-engineering of control programs when there is a change in mechanical setup of the manufacturing system or some other requirement that mandates on-the-fly changes to program code.

During operation in RUN mode, a PLC repeatedly executes a scan, during which input channels from from all input modules are copied to the process image tables, the control program is scanned, and the outputs held in the internal memory are

updated and finally copied into the actual output modules. The scan cycle time, depending upon the size of control program and the number of I/O channels, is of the order of milliseconds to minutes [186].

For serving today's expanding demands of industrial controllers, leading automation companies have come up with new set of industrial controllers called Programmable Automation Controllers (PACs) with advanced features in terms of communication, data logging, process control in a single programming environment. Allen Bradley series of PACs are one of the most featured set of controllers [136] and include various models such as ControlLogix, Compactlogix, FlexLogix, MicroLogix. These controllers follow a tag-based approach for memory addressing where tags represent data and identify the areas in the memory where they are stored, which makes it flexible to program data as per the needs of the application. The 1756-ENBT and 1756-EWEB modules provide ethernet connectivity to the controllers, supporting a wide variety of control system protocols such as EtherNet/IP, Modbus TCP, etc. Other popular PLC manufacturers such as Siemens, ABB, Omron, Delta, Honeywell, Schneider Electric follow a similar architecture and protocol space.

2.2.3 Control Programs

Control Programs incorporate the *logic* for controlling the factory processes, and in essence act as the 'decision-making layer' for the process control framework.

The programs consist of a sequence of instructions, that constitute high-level abstraction of the process descriptions specific to the industrial application. Control programs employ digital sequential logic circuits to make decisions and calculate the value of manipulated variables, based on values of process variables. Different PLC vendors offer dedicated tools for development and compilation of control programs, such as Simatic STEP 7 by Siemens, Studio 5000 by Rockwell Automation, Software Unity Pro by Schneider Electric, etc. Based on control narratives by operations engineers, control programs are developed at engineering workstations by process engineers in programming languages compliant with IEC 61131-3 standards [99] on software architecture and PLC programming guidelines, and are downloaded to the PLC over industrial communication protocols.

- **Structured Text (ST):** This is a text-based high-level language that mimics classical programming languages C, PASCAL with the flexibility to perform

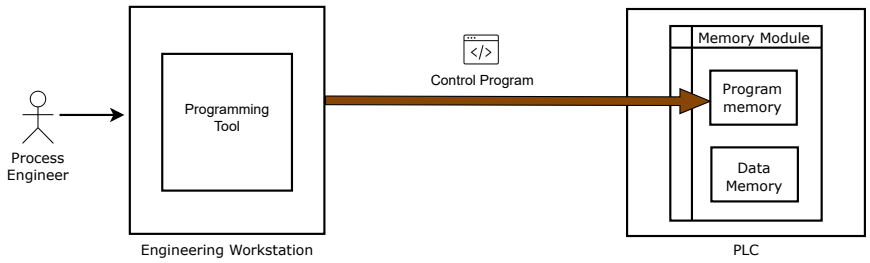


Figure 2.2: Control Program Engineering: Control programs are downloaded to the PLC using dedicated industrial buses on top of ICS network protocols.

complex arithmetic calculations, table manipulations that can be performed over a single command line.

- **Function Block Diagram (FBD):** This is a graphical programming language that uses a set of elementary blocks as functions, with multiple input-output arrangements.
- **Ladder Diagram (LD):** Based upon an industrial evaluation, this is the most popular language among ICS engineers and technicians. This is a graphical language resembling traditional electrical wiring diagrams that uses fundamental components such as rungs, contacts and coils to execute basic control functions, such as logic, time-control, counting, etc.
- **Instruction List (IL):** It is a low-level textual language that resembles assembly language and most of IEC compatible development tools translate codes written in other languages to IL before generating compiled binaries.
- **Sequential Function Chart (SFC):** This is mostly a graphical approach for structuring the program using a flow-chart based construct.

2.2.4 Industrial Communication Protocols

The complex nature of distributed control in industrial infrastructures requires a set of communication schemes to tie the various system modules together. Some distinct features that set it apart from IT networks include: (a) real-time or nearly real-time control, (b) data integrity, (c) high noise immunity, (d) reliability in harsh industrial environments [33]. Initially the PLC manufacturers designed their own control networks, called data highways, which were proprietary in every sense, which made it difficult to integrate PLCs, I/O devices, HMIs that are designed by different manufacturers. Today, most manufacturers of PLC and

control equipment support open communication networks, based on intermediate standards developed through industry associations.

Industrial communication networks can be categorized into *three types*, of which we discuss the first two, as the third one is out of the scope of this study.

- **I/O Networks** - I/O networks work at the lowest level of the process control framework, and provide communication links between the field devices and PLCs at the *control layer*. The industrial communication protocols at this layer are typically characterized by relatively small data packet sizes, ring topologies, high degree of reliability, fault-tolerance, real-time and deterministic operation [33]. These networks carry device-dependent messages following non-standard formats of fixed lengths, which is defined by vendors and process engineers of the control application [196].
- **Distributed Control Networks** - Distributed control networks work at the middle layer of the process control framework, providing communication link between PLC and HMIs, data historians and engineering workstations constituting both *supervision* and *configuration layers*. In a nutshell, this network allows exchanging messages that helps control message requests, monitor and manipulate tags and device parameter specifics and device configurations. The messages contain rich semantics about the information being exchanged, with characteristics such as high speed, star topologies, high degree of reliability, etc.

For wired communication networks, the industrial control framework adheres mostly to the standards: Fieldbus and Industrial Ethernet [160]. A careful analysis of these standards is essential for designing the control framework of an industrial application.

Fieldbus, standardized as IEC 61158 [59], provides deterministic, reliable, real-time networking solutions [19] but at a maximum of 10-12 Mbps, fails to provide the fast connectivity that Industrial Ethernet offers. Again, Ethernet connectivity provides high bandwidth and marks a significant increase in the speed of data-transfer (sometimes, at the speed of 100 Mbps), however it lacks intrinsic deterministic control due to collision domain (intrinsic property of ethernet networks). The size of data packets are larger with ethernet, and architectural variants allow easier integration of cyber security.

Due to availability and high performance of the ethernet products, unlike Fieldbus systems, with a significantly bigger consumer electronics and IT market, the price of ethernet devices are low, with high interoperability and performance. From a user perspective, the pool of talent on ethernet technologies, along with the easily available tools, make it easier to deploy than fieldbus technologies.

Ethernet connectivity also restricts the designer to limit the distance between various system modules, which Fieldbus systems can avoid. Fieldbus has multiple physical layers, such as RS-485, CAN, RS-232, whereas, the Ethernet networks use the same physical layer based on the standard 802.3, with the flexibility to operate using cat5e cables, making it simpler for implementing. Ethernet supports vertical integration, making the OT and IT networks integrate for overall control and monitoring in industrial automation. Ethernet requires no extra hardware, typically integrated into the CPU itself, whereas, Fieldbus requires an additional network interface card.

Examples of fieldbus includes Profibus [210], Modbus [209], DeviceNet [205], CANopen [203], ControlNet [204], etc. Examples of Industrial Ethernet include Profinet [211], EtherNet/IP [208], Ethernet CAT [207], Modbus TCP [16].

The 2022 annual report by ISSUU (see Figure 2.3) estimates Industrial Ethernet standards occupying around 66% of the global market of newly installed nodes compared to fieldbus technologies at 27% [103]. EtherNet/IP is the most popular and best in-class ethernet communication protocol [78], and the adoption is increasing at a fast rate globally. There are a set of arguments behind EtherNet/IP emerging as the most popular protocol among the industrial control systems. EtherNet/IP follows the CIP standards, where all devices are represented as a series of objects that decrease the training and start-up required when new devices are brought online. Also, Ethernet/IP provides improved response time and greater data throughput compared to other CIP implementations, such as DeviceNet and ControlNet [15]. EtherNet/IP can also connect devices in all the three layers of a model ICS, along with a consistent application layer interface. EtherNet/IP provides the best vendor support, flexibility and total architecture support as compared to all its competitors in the market, including Modbus/TCP from Groupe Schneider, Profinet from Siemens, and EtherCAT from Beckhoff [15].

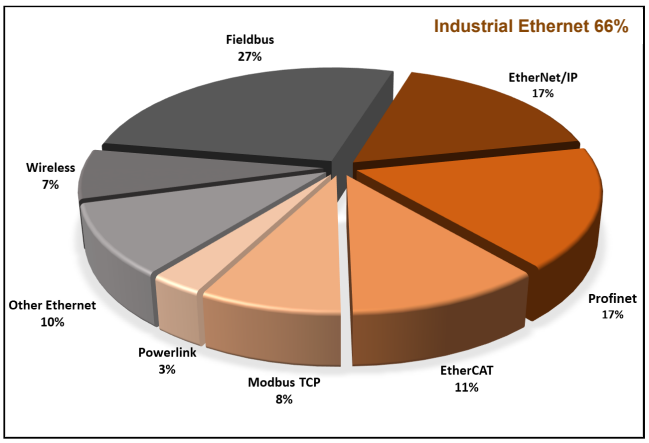


Figure 2.3: Market shares of industrial communication protocols based on 2022 report by ISSUU.

2.3 Analyzing the Threat Surfaces

In this section, we present an in-depth analysis of the threat surfaces for targeted attacks against industrial infrastructures, a brief overview of which has already been presented in Chapter 1 (see Section 1.1). Our discussion follows from the large-scale adversary tradecraft undertaken in infamous targeted attacks reported against critical infrastructures. The significance of this survey is in identifying and designing specific and accurate countermeasures targeted towards a secure ICS framework.

2.3.1 ICS design flaws

Infamous cyber-attacks plaguing the ICS domain mostly stem from the insecure-by-design industrial controllers and proprietary communication protocols that carry critical control and configuration information in cleartext.

The recent years have marked a significant increase in the number of ICS vulnerabilities [155]. CISA disclosed 681 vulnerabilities, with more than 22% assigned critical and 42% rated with high severity [114]. Nevertheless, it is essential to realize that *even if the industrial control establishments eliminate all these vulnerabilities, an adversary can still exploit the design flaws to cause targeted attacks.*

PLCs constitute one of the weakest links in ICS security [199]. Being an edge-device operating at the control, supervision and configuration layers, PLCs serve

as the most preferred attack point towards targeted attacks. The blueprint of a generic PLC does not include any modules for security; in particular, the embedded engineers did not incorporate security provisions during its design phase as they were developed with the sole purpose of achieving automation. PLCs are insecure-by-design, i.e. an adversary can use the documented features to stop a valve by writing commands to the PLC and need not exploit any vulnerabilities. Of additional note, the PLC vendors limit access to its memory to its protocol address space, and do not share the implementation details of the proprietary hardware architectures, which makes it harder for the customers, who do not have the provisions for installing security modules within the controller.

The ICS communication protocols do not include inherent provisions for security. Interestingly, application-specific protocols, such as Modbus , DNP3, EtherNet/IP carry critical control/configuration information cleartext. In particular, the ICS protocols are not designed with achieving the security goals of integrity, authentication and encryption. This indicates that any unauthorized network access (control plane) for eavesdropping/manipulating the process parameters or logic is not visible to engineering and security staff at the monitoring units.

An important takeaway is that till security controls are put to effect against the design flaws of ICS, no amount of defence and policy responses to prevent, detect or deter targeted attacks would be effective against industrial infrastructures. Correcting ICS protocol vulnerabilities is an ambitious goal as they are open communication standards, working perfectly for what they are designed and not specific to the ICS vendors. A convenient and feasible approach would be the ICS vendors taking the following initiatives:

- Integrate cyber security features directly into the PLC platform, and address the corresponding threat model to be shared with the customers.
- Allow some application-level access to the PLCs, so that the customers can define and deploy the necessary security features specific to the target control application.
- Integrate cyber security in the development life cycle, such that the customers are able to do regular audits for the SDL at FAT/SAT.

Viewed along these lines, long lifetimes of the legacy controllers imply living with the security risks for decades/years.

2.3.2 Heterogeneous ICS Networks

In what follows, we study the heterogeneous ICS networks to bring to light the most vulnerable layer in ICS framework that account for bringing about majority of the targeted attacks.

The control layer exchanges real-time signals, and the commands are preset in the hardware configuration of the devices. The messages at this layer follow non-standard application-level semantics, partially defined by the vendors and process engineers [196]. An adversary who has unauthorized access to the network at this layer, requires detailed understanding of the process design and implementation specifications for parsing and manipulating the packets. More specifically, the adversary needs access to device specifications, electrical and installation layouts.

However, packets at the supervision and configuration layers contain rich semantic application-specific information, making it less arduous for an adversary interpret the payloads. To see this, we refer to [196], where the authors discuss the challenges that come with parsing device-dependent implicit messages at the control layer, compared to the explicit messages at the supervision and configuration layers of Secure Water Treatment (SWaT) testbed [129].

2.3.3 Threat Surface

In this section, we analyze the various attack points in the ICS supervision and configuration layers, assuming that an adversary does not have to the control layer. *This is admissible, as the attack vectors with lower relative likelihood could be ignored* [185]. Several attacks are possible, if the adversary gets physical access to the cyber system, while a wide range of other attacks are possible by adding networked cyber-dimension, increasing the scope of what could be attacked.

We begin the section by introducing an *adversary, who can be a person, program or a computing system (running in time polynomial in the security parameter of the system).* The adversary has a basic understanding of how ICS control, configuration, and diagnostic data moves between the PLCs and HMI/engineering workstations. The ultimate goal of the adversary is to intercept the information exchanged between these devices and cause false data injection attacks or industrial espionage. We note that it might also be necessary to protect an ICS from threats such as

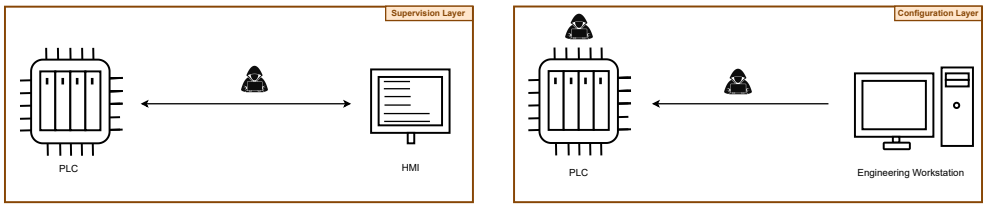


Figure 2.4: Attack points of an adversary at the *supervision* and *configuration* layers of an ICS.

phishing, denial-of-service attacks, etc., which is out of the scope of this study. The adversary is deployed in one of the following settings:

1. The adversary has unauthorized access to plant communication network at the *supervision* layer of the process control framework. We consider that the adversary can set up a physical device at any point in this network and has tools to interact with the ICS protocols and interpret the semantics of the ICS payloads. Such an adversary can leverage the access to perform interception and modification of critical control parameters exchanged between PLC and HMI.
2. The adversary has unauthorized access to the PLC and the plant communication network at the *configuration* layer of the process control framework. This gives him an advantage to recover the clear text control program implementation for the target control application and reverse engineer to perform reconnaissance of process control.

Of note, *we do not consider the adversary to have access to HMI/engineering workstations, as these devices are usually sophisticated and high-end reliable computing platforms, and thus could be deployed with device-level security.*

2.4 Recommended Security Controls

With no application-level security support provided by the PLC vendors and insecure nature of open communication standards, securing process control framework is solely dependent on bump-in-the-wire techniques using COTS hardware and software modules, that provides cryptographic implementations at the transport, network and data-link layers of the OSI model.

Nevertheless, it is important to consider the real-time nature of the industrial control applications. A control application with unacceptable delay is no longer time-invariant, and the maximum permissible delay is directly proportional to the response rate of the field devices [3]. While applications such as food processing could allow delay of up to 8 milliseconds, safety-critical applications, such as turbo-machinery, magnetic suspension systems allow up to 1 millisecond delay [96]. Lightweight cryptographic primitives as discussed in [107] could be implemented in real time running environments, provided their running time is within the latency requirements of the application, to avoid disrupting the normal physical operations of the industrial processes. Simulation experiments have been performed on PLC networks [194] with symmetric ciphers, and Rijndael AES has shown the best results in terms of speed of encryption and size of input data. However, using AES in a real PLC network can be very challenging as AES requires cryptographic hardware support and binary fields for Sbox and Mixcolumns computations with look-up tables for efficiency. With stream ciphers, simplicity in design and speed in hardware is a major advantage, however the initialization phase before the protocol starts is mostly time consuming. All these factors make it important to select cryptographic primitives by considering a careful trade-off between performance, cost and security. Considering safe optimizations of existing cryptographic algorithms is discussed in [173] has risks, as the embedded designer needs to extremely familiar with the algorithm he chooses to optimize with even a small mistake leading to expensive security breaches. Less expensive techniques, such as hashing can be used to provide authentication between PLC devices, however confidentiality cannot be implemented with such an arrangement. While the domain of cyber threats in the existing PLC networks is huge, a proper risk-analysis can lead to identification and development of a sophisticated security framework for integration into the ICS networks.

2.4.1 Defense and Policy Responses

In this section, we discuss the various defense and policy responses prepared by the industry in combating targeted attacks against critical infrastructures [101, 153].

- **Making a Clear Distinction between ICS and IT Security.** Visualizing how the security requirements of ICS deviate from that of an IT, is important in designing security paradigms, catered to the requirements of a control

application. ICS deals with real-time data, tampering which could lead to fatal consequences such as economic crisis or loss of human life. A high demand of availability and responsiveness of these time-invariant applications require cryptographic implementations that incur significantly low time and space overheads, and thus coping with the latency requirements imply lower security guarantees. In particular, IT security protocols, such as TLS, IPSec etc. fail to meet the aforementioned objectives. Furthermore, features mandated by IT security protocols might not be a defacto requirement for generic ICS facilities. What is more significant is that the IT protocols fail to provide application specific security in ICS-centric settings. As opposed to the CIA triad that bolster IT security, ICS focuses on *availability* to be the most significant aspect, followed by *integrity*, while attributing the lowest priority to *confidentiality*.

- **Creating Demilitarized Zones.** ANSI/ISA - 99 standard [5] suggests partitioning the ICS network into various logical DeMilitarized Zones (DMZ) according to the Purdue Enterprise Reference Architecture (PERA) model [125]. Within a zone, all the devices have similar level of security, based on parameters such as criticality and potential implications. Such segmentations help in deploying additional layer of security, preventing a threat to propagate once it infiltrates and infects one of the zones [177].
- **Implementing Anomaly and Intrusion Detection Tools on ICS Networks.** Prioritization of availability and being uptime has led to understating active or preventive solutions, and shifting towards a detection and response paradigm. ICS security practitioners recommend implementing signature and anomaly based IDS in the industrial communication networks, a detailed discussion on which is presented in Chapter 3 (see Section 3.1.1).
- **Investing in Cyber Security Education and Training for IT, OT, and Hybrid IT/OT Personnel.** All employees authorized to access OT/IT assets associated with a target control application are recommended to receive appropriate cyber security training to lower the risk of security breaches. Furthermore, management authorities at the enterprise level are given awareness training to make secure transactions at the IT-OT convergence level.

2.5 Cryptographic Primitives

In this section, we discuss about cryptographic primitives, which we make use of in the proposed constructions (covered in the upcoming chapters).

2.5.1 Cryptographic Implementations

The scope of our cryptographic implementations is limited to ciphers that incur low computational overhead, and in this section we present a brief overview of the cryptographic primitives used in our proposals. We have aimed to implement the proposed methodologies using cryptographic ciphers that are ideal for deploying in environment with tight cost and implementation constraints: (a) limited memory (registers, RAM, ROM), (b) reduced computing power, (c) low battery power, and (d) real-time response [191].

Block Cipher. Among the popular lightweight block ciphers, we have implemented one of our constructions using AES (considered in ISO/IEC 29192), with a 128 bit key and block size, throughput of 12.40 and 3400 GEs, encrypting a plaintext within 1032 clock cycles. Though AES could be challenging to implement in real PLC networks, *our justification behind using AES-128, is that it is considered as one of the most suitable variants in NIST-approved cryptographic primitives for constrained environments* [131] *and its speed in software implementations makes it a suitable choice for our simulation setups.* We note that the results of our experiments are expected to improve when implemented with lightweight ciphers that are faster than AES-128. For information on how AES works, interested readers can refer to [95].

Stream Cipher. We have implemented two of our constructions with ChaCha-20, a lightweight stream cipher [49] developed by DJ Bernstein, and compared their performances with AES. With stream ciphers, simplicity in design and speed in hardware is a major advantage, however the initialization phase before the algorithm starts is mostly time consuming. ChaCha-20 uses a 256-bit key, with block size of 64 bytes, and a throughput of 111.3. A detailed description of the algorithm can be found in [130].

2.5.2 Program Obfuscation

In this section, we present a survey on program obfuscation [142], a tool to protect intellectual property of software programs. The topic is extensively studied

by the research community under two broad categories, *code obfuscation* and *cryptographic obfuscation*. We explore the obfuscation techniques employed by real-world applications (broadly classified as code obfuscation) and follow this with a discussion on black hat obfuscation scenarios, where malicious obfuscators propagate malwares in computer software. Finally, we study the standard notions used in cryptographic obfuscation, *the last of which is the highlight of this discussion.*

Obfuscation is a powerful tool that aims to achieve security against reverse engineering of classified information in a program from a Man-At-The-End (MATE) adversary who has complete control over the program and its execution environment [8].

We give a brief intuition to the security guarantees provided by program obfuscation towards software protection through the following motivating context: Suppose a software developer designs an application program that solves an important problem and wishes to commercialize his work by selling his program. However, he is now challenged by the fact that his customers can extract the valuable information proprietary to the software and gain some economic advantage through it. Such an extreme threat environment, where the software implementation of a program is available to an untrusted host, standard cryptographic solutions with encryption algorithms do not apply, as the program needs to be decrypted and then executed. *This leads to a compelling yet general question: how would the software developer ensure that the intellectual property of the application program in not extracted by his customers?*

To answer the above, the software developer could obfuscate the program before distributing it to his customers, such that the secrets within the program are protected (heuristically or provably), yet facilitating correct functionality. Though an ambitious goal, as we will see in the upcoming sections, this could be achieved for a broad class of programs.

Program obfuscation protects the *assets*, which are the secrets within a software program. The assets are defined by the owner of a program, which could be proprietary algorithms, strings (cryptographic keys), designs (data structures, modules), regular expressions, etc. and needs to be kept hidden from anyone who has access to the source code or binaries. An obfuscator transforms a program

to its semantically equivalent counterpart, such that access to the obfuscated version does not give away assets, yet preserves the functionality. Depending upon the security properties a program desires to achieve, obfuscation can be a powerful tool to defend against both static analysis (analysing the code) and dynamic analysis (learning the assets from the input/output behaviour of the program) by PPT algorithms.

In the following, we look into the various obfuscating code transformations implemented in real-world applications [1], and analyze the security provided by these techniques.

Code Obfuscation

Code obfuscation refers to the modification of the source code or compiled code such that comprehension and interpretation of the assets through static and dynamic analysis are "hard" to perform [142]. Though there are no rigorous definitions as to security guarantees provided by such modifications, the authors in [57] identify *four* valuable criteria for evaluating the strength of the obfuscating code transformations:

- **Potency** - The obfuscating transformations should introduce enough confusion to the original code by hiding its intent. To achieve this, the following methodologies are suggested: introducing new methods and functions, increasing nesting level of conditional and looping constructs [55, 57, 58, 162], method arguments, variable dependencies [94], etc. such that overall program size increases, leading to deliberate addition of *ambiguous* information.
- **Cost** - The obfuscating transformations should not introduce *much* overhead in time and space.
- **Stealth** - The obfuscated code should be indistinguishable from the original code, i.e. it should be *hard* for an adversary to determine whether an obfuscating transformation has been applied to the original code.
- **Resilience** - An automated deobfuscator should expend *more resources* to extract the assets from the obfuscated code, compared to the effort put in for constructing the deobfuscator.

[1] Real-world applications of code obfuscation include *Digital Rights Management*, *Mobile Agent Computing*, *Grid Computing*, *Artificial Diversity*, *Malicious Reverse Engineering, etc.* [142]

In [57], the authors identify *four* main classes of obfuscating code transformations:

(a) **Abstraction Transformations** - These transformations break down the structural representation of the program. Obfuscation techniques include expression equivalence [187], reordering code and data [142], function inlining and outlining [56], identifier renaming [52], etc.

(b) **Data Transformations** - These transformations modify the form in which data is stored in a program. Encoding integers and strings [142], arrays (array permutation [74], array restructuring [73] are very prevalent in high performance code), etc. are fairly common approaches towards obfuscating hard-coded data in a software program.

(c) **Control Transformations** - These transformations modify the control structures (if and while statements), such that an adversary finds it difficult to reconstruct the original control flow graph. The most stealthy, low-cost and thus prevalent control flow obfuscation method is opaque predicate obfuscation [215]. Conceptually, opaque predicates are Boolean valued expressions, whose values are either true or false for all possible input values, and are fixed while applying the obfuscating transformations. The intuition is to introduce superfluous branches that will never be taken at runtime (for example, an opaquely true predicate forces the control flow execution towards the true path). However an attacker finds it difficult to distinguish them from predicates that lead to original path conditions. Due to its popularity in a wide array of applications (software diversification [62], metamorphic malware mutation [38, 39], software watermarking [6], Android Apps obfuscation [113]), opaque predicates have been an area of active research. Other control flow obfuscation techniques are control-flow flattening [198], call-stack tampering [175], loop transformations [20], etc.

(d) **Dynamic Transformations** - These transformations cause the program to continuously transform at the runtime. Packing and encryption [45], hardware-assisted code obfuscation [227], etc. are some example methodologies.

An important takeaway from the above discussion is that *the notion of code obfuscation does not quantify (a) the theoretical limits of time and resources to be expended by an adversary in performing deobfuscation, (b) success of an adversary in*

attempting to deobfuscate an obfuscated program. The threat models for these transformations are vague and based upon the goal that an adversary should use more resources to reveal the assets in the obfuscated program, as compared to the assets used in the original program. This provides sufficient reasons to move to cryptographic obfuscation, where the assets are protected by some cryptographically hard problems, which however is rarely used in practical implementations.

We now review black hat obfuscation, a scenario where obfuscators hide the malicious functionality within programs. Our sole purpose behind this discussion is to bring to attention the significance of designing *verifiable obfuscators*, which we introduce in Chapter 8.

Unsurprisingly, the powerful essence of program obfuscation has found spectacular success in black hat scenarios [143], where the obfuscators hide undesirable functionality within software programs. Obfuscation is used by malware writers to protect their viruses, worms, Trojans and rootkits from detection.

Obfuscation conceals characterizing features of malware, undermine anti-malware software, and thwarts malware analysis [150]. Polymorphic and metamorphic malwares make use of various obfuscation transformations such as dead-code insertion, register reassignment, subroutine ordering, instruction substitution, code transposition and integration to evade detection [219]. However, the most widespread and popular of the existing approaches incorporate trigger-based initiation of malicious payloads based on conditions satisfied by a specific set of inputs [141] (for example, malicious actions are made to hide behind some conditional expressions such as a particular day, presence of a certain file, keyloggers, bot-command inputs, etc), followed by obfuscating the conditional expressions. In [180], the authors design a conditional code obfuscation technique that relies upon one-way hash functions, such that it is hard to identify the set of inputs for which the conditions are satisfied. In particular, the authors obfuscate the equality conditions to inject some malicious functionality and develop a compiler level tool written in C/C++.

The most important line of defense against such malicious codes are *scanners* (static analysis), that rely upon a database of signatures characterizing malware instances, and *debuggers* (dynamic analysis), that study the behavior of a program

by executing the binaries on restricted environment. Of note, the static analysis methods are inaccurate in detecting the zero-day exploitations using techniques such as structure and pattern identification [141], and the dynamic analysis methods, such as fuzzy testing schemes lead to high false positive rates due to low coverage of inputs [215]. Interestingly, the cryptographic community has not been much attentive towards dealing with malicious obfuscators that inject undesirable functionality in programs.

Cryptographic Obfuscation

Program obfuscation has received considerable recognition in the cryptographic community over the recent years. An obfuscator $\mathcal{O}$ is a probabilistic polynomial-time algorithm that transforms a program C to its semantically equivalent counterpart $\tilde{C}$, such that a secret, that is efficiently computable from C, is hard to extract, given $\tilde{C}$.

The definitional framework of program obfuscation was established by Barak *et al.* in their seminal work [24] using a simulation-based security paradigm. They established the notion of *virtual black-box* (VBB) obfuscation, where a polynomial adversary $\mathcal{A}$ having access to $\tilde{C}$ has but a negligible advantage in extracting a desirable property over a polynomial simulator $\mathcal{S}$, who only has oracle access to C; in short, anything that is efficiently computable from $\tilde{C}$, can also be computed efficiently from the input-output access of the program. Their main results rule out the possibility of designing efficient obfuscators for all class of programs. However, obfuscators for *specific* families of programs may be achievable.

We recall some basic concepts used in theoretical cryptography so that the readers can refer to them while going through the standard notions used in cryptographic obfuscation (defined in the subsequent segments).

Probabilistic Polynomial Time (PPT) Adversaries [108]. They are efficient randomized algorithms running in time, *polynomial* in the security parameter $\lambda \in \mathbb{N}$ of the system. We equate the notion of "small probability of success" with probabilities smaller than inverse polynomial in λ. We say that a scheme is secure, if a PPT adversary cannot break the scheme, except with negligible probability in λ. For real-world efficiency, we require the honest parties to run in polynomial time in λ, while the adversaries are allowed to be much more powerful.

Definition 2.5.1 (Distributional Virtual Black-Box Obfuscator (DVBB) [24, 23])**.** *Let $\lambda \in \mathbb{N}$ be the security parameter. Let $\mathcal{C} = \{\mathcal{C}_\lambda\}$ be the family of polynomial-size programs parameterized by inputs of length $n(\lambda)$, and let $\mathcal{D} = \{\mathcal{D}_\lambda\}$ be the class of distribution ensembles, where $\mathcal{D}_\lambda$ is a distribution over $\mathcal{C}_\lambda$. A PPT algorithm $\mathcal{O}$ is a VBB obfuscator for the family $\mathcal{C}$ and the distribution $\mathcal{D}$, if it satisfies the following conditions:*

- *Functionality Preservation : For every $\lambda \in \mathbb{N}$ and for every $C \in \mathcal{C}_\lambda$, there exists a negligible function $\mu(\lambda)$, such that:*

$$\Pr_{\mathcal{O}}\ [\,\forall x \in \{0,1\}^{n(\lambda)} :\ \ \mathcal{O}(C)(x) = C(x)\,] > 1 - \mu(\lambda)$$

  where the probability is over the coin tosses of $\mathcal{O}$.

- *Polynomial Slowdown : For every $\lambda \in \mathbb{N}$ and for every $C \in \mathcal{C}_\lambda$, there exists a polynomial q such that the running time of $\mathcal{O}(C)$ is bounded by $q\,(|C|)$, where $|C|$ is the average running time of the program.*

- *Virtual Black-box : For every (non-uniform) polynomial size adversary $\mathcal{A}$, there exists a (non-uniform) polynomial size simulator $\mathcal{S}$ with oracle access to C, such that for every distribution $D \in \mathcal{D}_\lambda$:*

$$\Big|\ \Pr_{C \leftarrow \mathcal{D}_\lambda, \mathcal{O}, \mathcal{A}}[\mathcal{A}(\mathcal{O}(C)) = 1] - \Pr_{C \leftarrow \mathcal{D}_\lambda, \mathcal{S}}[\mathcal{S}^C(1^\lambda) = 1]\ \Big| \leq\ \mu(\lambda)$$

  where $\mu(\lambda)$ is a negligible function.

To summarize, if there exists a non-uniform PPT algorithm that computes a predicate from an obfuscated function, then there exists a non-uniform PPT algorithm that computes the same predicate from oracle access to the function with almost same probability.

In [41], Canetti shows the construction of an efficient obfuscator for *point functions* (a point function evaluates to 1, if and only if the input equals to a particular value, e.g. password check) that achieves a relaxed notion of virtual black-box using a probabilistic hashing algorithm $\mathcal{R}$ that imitates the 'useful' properties of a random oracle. The obfuscated program stores $\mathcal{R}(x)$, where x is sampled uniformly from a superlogarithmic min-entropy distribution, such that on input y, outputs 1, if

$\mathcal{R}(x) = \mathcal{R}(y)$. Access to the obfuscated program does not give away the hard coded hashed secret, yet facilitates in correlating it with a given input value. Additionally, it does not allow a PPT adversary to identify the secret, except with negligible probability, as *it is computationally hard to find an accepting input*, a property defined for *evasive functions* (see Definition 2.5.2).

Since then, there has been a multitude of research works towards constructing special-purpose obfuscators that restrict to a special class of functions for which it is hard to find an accepting input from the black-box access to the program. Obfuscating these functions gives a strong intuition that learning assets of a program by identifying the accepting inputs within modeled time-complexities is computationally hard.

In what follows, we define evasive functions, which we will restrict to in our proposed constructions. They are a special class of Boolean functions with the condition that for a random circuit from the collection, a PPT algorithm finds it hard to map an accepting input. For these functions, it makes sense to consider an *input-hiding* obfuscator (introduced by Barak *et al.* in [41]), the intuition behind which is that given an obfuscated program produced by an input-hiding obfuscator, it should be hard to find an accepting input for the program.

Definition 2.5.2 (Evasive Circuit Collection [41])**.** *A collection of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ parameterized by inputs of length $n(\lambda)$ is called evasive, if there exists a negligible function $\mu(\lambda)$, such that for every $\lambda \in \mathbb{N}$ and every input $x \in \{0,1\}^{n(\lambda)}$:*

$$\Pr_{C\leftarrow\mathcal{C}_\lambda}[C(x) = 1] \leq \mu(\lambda)$$

where the oracle access to the circuit allows at most $p(n)$ queries.

Definition 2.5.3 (Input-Hiding Obfuscator [41])**.** *An obfuscator $\mathcal{O}$ for a collection of evasive programs $\mathcal{C}$ is input-hiding, if for every PPT adversary $\mathcal{A}$, every $\lambda \in \mathbb{N}$ and every auxiliary input $z \in \{0,1\}^{poly(n)}$ to $\mathcal{A}$, there exists a negligible function μ, such that:*

$$\Pr_{C\leftarrow\mathcal{C}_\lambda}[C(\mathcal{A}(z, \mathcal{O}(C)) = 1] \leq \mu(\lambda)$$

where the probability is also over the randomness of $\mathcal{O}$.

To summarize, the above definition signifies that, given an obfuscated program $\mathcal{O}(C)$, it is hard to find an input that evaluates to 1.

A weaker security notion introduced by authors in [24], known as *indistinguishability obfuscation* (iO), states that obfuscations of two functionally equivalent programs should be computationally indistinguishable from each other.

Definition 2.5.4 (Indistinguishability Obfuscator (iO) [24])**.** *A uniform PPT algorithm $\mathcal{O}$ is called an indistinguishability obfuscator (iO) for a class of programs $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$, if it satisfies the following properties:*

- *Functionality Preservation : For every $\lambda \in \mathbb{N}$ and for every $C \in \mathcal{C}$, there exists a negligible function $\mu(\lambda)$, such that:*

$$\Pr_{\mathcal{O}}\ [\,\forall x \in \{0,1\}^{n(\lambda)} :\ \ \mathcal{O}(C)(x) = C(x)\,] > 1 - \mu(\lambda)$$

  where the probability is over the coin tosses of $\mathcal{O}$.

- *For any (not necessarily uniform) PPT distinguisher $\mathcal{D}$, there exists a negligible function $\mu(\lambda)$, such that the following holds: For every security parameter $\lambda \in \mathbb{N}$, and for all pairs of programs $C, F \in C_\lambda$, if $C(x) = F(x)$ for all inputs $x \in \{0,1\}^{n(\lambda}$, then*

$$\Big|\, Pr[\mathcal{D}(\mathcal{O}(C)) = 1] - Pr[\mathcal{D}(\mathcal{O}(F)) = 1] \,\Big| \leq \mu(\lambda)$$

In short, the above definition states that any two programs that are "functionally equivalent", their obfuscation should be computationally indistinguishable.

Chapter 3

Literature Review

This chapter explores the existing state-of-art on addressing the security goals identified in Chapter 1. The oncoming discussion is divided into *two* distinct parts.

The *first* part looks into the efforts by the ICS security research community in detecting and preventing false-data injection attacks. In addition, it highlights the importance attributed to the knowledge of process behavior in bringing about influential and sophisticated attacks on ICS facilities, and analyzes the existing literature that explores the efforts of an adversary, who uses various tools and techniques to learn the operational semantics of target control application. Finally, it presents an investigation on the existing attempts towards formalizing control program abstraction.

The *second* part gives a detailed overview on the advances in program obfuscation. In particular, it highlights upon the general and special purpose obfuscators in the literature. Our intention behind this broad-level discussion on state-of-art in cryptographic obfuscation is that *not only does it address one of the security objectives (not addressed prior to this work, to the best of our knowledge) in defending ICS against targeted attacks, but also the contributions are of independent interest, and have wider applicability in software program obfuscation.*

3.1 Defending Against Targeted Attacks

In this section, we elaborate on the methodologies employed by the state-of-art in achieving the security goals discussed in Section 1.2.3. Before we review the

related work in this domain, we would like to draw the attention of the readers to the fact that *performing academic research in ICS security is largely hindered by limitations in accessing a fully functional ICS facility, and majority of the published algorithms rely upon simulations and emulations to test the effectiveness of the designed approaches.*

3.1.1 Detecting False-Data Injection Attacks

The existing methodologies for detecting false-data injection attacks either involve inaccurate passive predictive modeling techniques using Intrusion Detection Systems (IDS) [90, 104] or bump-in-the-wire approaches [48, 194] with expensive cryptographic implementations that introduce heavy overhead in the critical control path.

IDSs are categorized with *signature* and *anomaly* based detection techniques [139]. Signature-based IDS contains a large database of attack signatures and correlates them with the data being monitored, nevertheless are ineffective against custom-made and zero-day exploits prevalent in targeted attacks [139].

Anomaly-based IDS models the intricate knowledge of target control application into a set of rules that specify ideal behaviour, and flag any deviations [137]. Along the same line, authors of [91] use auto-regression modelling techniques to design a variable-specific forecasting model by monitoring the process variables, and distinguish unobserved values as potential false-data injection attack.

In [137], Mitchel *et al.* design an IDS defining three states as secure, warning and insecure, based on the behaviour of sensors and actuators. They model system behaviour as a probabilistic finite automaton and use compliance degree to distinguish between normal and attack scenarios.

However, anomaly-based predictive models are mostly inaccurate with high false-positive rates [98], and designing them require in-depth knowledge of the process control, which a process engineer might be unwilling to disclose to a security personnel. As per SANS 2022 survey [153], only 7% of the ICS facilities allow an external security provider to be acquainted with the control framework, while rest of the organizations do not allow anyone but the owner of the operator or the engineering manager.

IDS can further be categorized as *host* and *network* based systems, depending on the data source [98]. Host IDS (HIDS) such as Alienvault [66] monitors individual systems by analyzing OS files, software logs and network connections, and can distinguish based on privilege escalation, modification of critical files, known malware rootkits, critical services that may have been stopped, and used access to the system and applications [128]. However, HIDS are not suitable to be deployed in ICS environment due to its nature of interfering with the processes [197]. On the contrary, Network IDS (NIDS) such as Snort [172] uses adapters to examine network capture, defines desired stages with parameters such as traffic alert, packet size, etc. and flags warnings if it detects any changes in the baseline [168].

Several approaches towards designing NIDS are based upon analyzing industrial communication protocols using syntax and grammatical verification rules [51, 119, 140]. Morris *et al.* design an IDS based on Modbus protocol specification, that uses the rules of Snort [140]. Similarly, Lin *et al.* design an IDS with a protocol parser that supports DNP3 and analyze legal values of different fields in the message using Zeek [154] policies. In [51], the authors explore the ICS protocol specifications and analyse communication patterns in diverse control applications to identify patterns for tamper detection. In [120], Linda *et al.* mirror the network traffic under stable conditions, and correlate them with ongoing packets to flag discrepancies. Caselli *et al.* in [47] discuss the limitations of such network based IDS in detecting covert Stuxnet-like custom-made attacks and zero-day attacks.

Other approaches [28, 48, 196] implement cryptographic primitives using COTS modules, deployed in the critical control path, to perform accurate detection of false-data injection attacks.

The authors in [177] discuss how an ICS adversary causes false-data injection attacks by exploiting the existing vulnerabilities in ICS networks. They discuss the significance of modifying the communication protocols, and employing SHA2 and a public-key/private key primitive for imparting data integrity and device authentication. However, deploying such constructions would incur unacceptable overhead for time-critical applications.

In [28], Batke *et al.* discuss an *integrity profile* for integrity assessment of ICS payloads using TLS, which is an expensive solution, in terms of complexities

involved, and thus not suitable for time-critical applications. In [48], John *et al.* suggest adding a signature to the ICS payloads using ECDSA and HMAC, which introduces significant overhead in the network bandwidth.

In [212], the authors use additional hardware modules on top of serial links to encrypt control messages and append a Message Authentication Code (MAC) to it. Such solutions are not custom-made for time-invariant critical applications and thus not feasible to be deployed in control environment with high sampling frequencies.

In [64], the authors deploy a low-cost device parallel to the controllers, which transparently intercepts the control commands and correlates them with the status of the physical I/O, such that potential inconsistencies due to false-data injection are correctly detected. However, the authors do not discuss the threat surface introduced due to the passive channel, i.e. how likely it is for an adversary to compromise the communication in the supplementary pathway to render the approach ineffective.

We aim to design solutions for detecting false-data injection attacks, that are neither based on inaccurate and predictive intrusion detection methodologies, nor do they impact the bandwidth used for critical control communication.

3.1.2 Preventing False-Data Injection Attacks

In this section, we review the state-of-art methodologies to prevent false-data injection by *encrypting* ICS control and configuration messages. We also look into the recommendations by ICS security practitioners for implementing *security at the application-level* in industrial control applications. We start the discussion by noting that existing solutions to impart confidentiality come in form of bump-in-the-wire approaches that rely upon security offered by *transport* [28], *network* [194] and *data-link layers* [194] .

Batke *et al.* [28] were the first to suggest wrapping the ICS application layer payload in TLS/DTLS protocols by using a *confidentiality profile*. This was followed by standardizing TLS in ICS network communication by IEC 62351-3 [157]. CISA critical infrastructure security requires all ICS facilities to employ end-to-end encryption using TLS [53].

Though TLS is effective in securing IT communications, its features are not optimal for use in ICS settings [14]. A typical control framework might have requirements such as *'provide confidentiality/integrity/authentication if read/write request about valve v is being communicated'*. TLS 1.3 imposes mandatory encryption, and as such cannot be used for addressing such customized security requirements. Also, features mandatory in TLS, such as AEAD and perfect secrecy [149] are not defacto requirements in a generic ICS facility. Finally, TLS is an expensive solution, in terms of the complexities involved, and thus not suitable for deploying in time-critical applications [14].

In [194], the authors suggest a bump-in-the-wire solution that provides confidentiality using AES_CTR to encrypt frames (link-layer implementation), along with a CRC-based detection method, which is mainly for serial-based communication systems and does not provide customized application-level security.

Secure DNP3 [127], a sophisticated ICS protocol that provides optional encryption support with TLS, incurring noticeable overhead, does not take into account the customized security requirements of an ICS facility.

A recent noteworthy development SSP-21 [14] provides optional encryption support by allowing encrypted sessions using an authenticated encryption mode. Such expensive AEAD implementations are adequate for violating the real-time requirements of a safety-critical application, and thus we claim are not suitable for generic ICS facilities. Furthermore, the authors include a timestamp in session messages (a feature not provided by TLS) and claim to impart purpose-built security. However, we argue that such a solution, though beneficial in thwarting replay attacks, does not capture the custom-made security essentials of a control application.

We aim to design solutions that provide defence against false-data injection attacks within control infrastructures, that not only incur minimal overhead, but also aim to achieve application-level security, in line with the demands of an ICS application environment.

3.1.3 Towards Reconnaissance of Process Control

This section highlights the importance attributed to the knowledge of process behaviour in bringing about influential and sophisticated attacks, and analyses the existing literature that explores the efforts of an adversary, who uses various tools and techniques to learn the operational semantics of a target control system.

In [121], Line *et. al.* explain the role of motivation, resources and competencies of well-organized adversaries in causing catastrophic consequences to the process control, and argue that targeted attacks are more powerful than low-budget kiddie scripts, that can be mitigated using off-the-shelf security products.

The "Industrial Control System Cyber Kill Chain" [9] emphasizes adversary's intimate familiarization of the target process in causing reliable and predictable harm on systems. In [13], the author mentions different tools and techniques to perform reconnaissance over ICS network protocols.

In [132] McLaughlin shows that process analysis could be done by obtaining the control program which could further be used for process manipulation, resulting into targeted attacks on the victim ICS.

In [91], the authors describe how *awareness* information containing regular and critical updates to the supervisory infrastructure by the PLCs, collected from the network traces, could be leveraged to design variable-specific forecasting model corresponding to the process variables. They indicate that by parsing the header and data segment of the application-layer protocol data unit (PDU) and using techniques such as *auto regression modelling* and *control limits*, a behavioural model for each process variable can be constructed, which eventually leads to extracting the process control design. However, their threat model does not include an adversary who leverages unauthorized network/PLC access to acquire control programs, followed by reverse-engineering the program to extract ICS process semantics.

Keliris and Maniatakos [109] propose a framework (ICSREF) to extract PLC control logic by reverse engineering compiled control programs, followed by building knowledge databases and reconstructing control flow graphs (CFGs). They discuss how this design can facilitate an adversary in dynamic process-aware payload generation.

Authors in [161] show how to perform dynamic analysis of the reactive behaviour of PLC control framework by recording the trace logs of the control program run in the original execution environment and re-running the values collected from the trace log in the host simulation environment, followed by building transition graphs and identifying recurring patterns.

In [165], the authors develop a tool (Similo) for automated recovery of control logic from ICS network traffic dump of control program downloads. They make use of decompiler within ICS engineering software and reconstruct the high-level source code.

The authors of [133] present a tool (SABOT) that (a) recovers the semantics of the PLC memory locations mapped to the physical devices by comparing adversary's specification on process control design with unauthorized copy of control programs downloaded from PLC, and (b) generates malicious payloads for the target ICS. The authors impose a strong requirement by allowing powerful adversaries with accurate knowledge specification on the target plant behaviour obtained from sources such as vulnerable HMIs, control plane protocols, etc. However, the authors are imprecise about how HMIs help in identification of process design with full accuracy, as it is but a reflection of the high level abstraction of the status of process variables or control logic found in the PLC. It is also unclear how the network traces aid in accurate target specification as predictive models can be only be partially accurate, and also SABOT allows adversaries to be unfamiliar with communication protocols. Finally, the authors suggest countermeasures against such attacks using control logic obfuscation with no details whatsoever in how to achieve the same.

In [179], the authors specify that a hard-coded value (for example, a program snippet comparing the value of a sensor against a hard-coded threshold) in the control program could be dangerous in view of contextual code manipulations. However, they do not provide any detailed solutions to contain such attempts of the adversary.

On further explorations, we observe that most of the existing works in ICS security research focus on analysing false data injection attacks [46, 112, 123, 214] between ICS equipment and corresponding propositions on 'detecting and preventing' such

attacks by adding cryptographic solutions with goals of integrity, authentication and confidentiality [28, 48, 136, 196], while a very little attention has been given on constraining the adversary in extracting the process control design. We argue that the latter deserves significant attention by the research community as the consequences of false data injection attacks could be escalated severely when preceded by reconnaissance of process control. To the best of our knowledge, none of the existing works propose detailed methodologies for containing reconnaissance of process control in ICS environments.

We aim to design solutions that prevent the adversarial efforts of extracting the operational semantics of a process control system from a recovered implementation of a control program for the target environment.

3.1.4 Formalization of Control Programs

The existing literature on formalizing control programs derives its motivation from the need for software re-engineering and formal verification of PLC codes [40, 87].

Falcoine and Krogh, in [79], model control programs as functions of the form $\mathcal{B} : U \times Q \rightarrow Q$, where U and Q denote the set of input and output variables respectively. Nevertheless, the formalization does not specify the details on control flow graph of the application. Adding to that, the authors do not consider PLC analog control engineering by restricting the variables to be boolean.

In [124], the authors present Reusable Automation Component (RAC) containing implementation details and formal specification of control programs. However, not only that the formalization is complicated in terms of the specifications used within RAC, but also it does not include the formal description of how the control function reacts with the process.

In [220], Younis *et al.* develop an abstraction of low-level IL program into a high-level form by line-wise conversion of the existing code into the form $IF <$ expression $> THEN <$ statement $1 > ELSE <$ statement $2 >$ and convert it into state automaton using SVG.

A careful analysis of the state-of-art methodologies for developing formal models of control program shows that such formalization captures the abstraction of various components of the program, but does not capture the control flow, i.e. how the control function interacts with the target system. This is admissible, as the inspiration behind formalizing the control function is to identify the erroneous behaviour of the program and check whether the program follows the structured representation as defined by the process engineer [81].

To conclude, a methodical approach, that not only details the interaction of input and output variables with the system, but also captures accurate control flow, which is an area of active research [79], will be a significant step in designing solutions for efficient software recovery, verification, as well as increasing adversary's cost of attack in extracting process semantics, the last of which is the focus of this work.

3.2 Advances in Program Obfuscation

In this section, we study the state-of-art general and special purpose obfuscators.

3.2.1 General and Special Purpose Obfuscation

The ideal notion of obfuscation requires "perfect" correctness, where the obfuscated program $\tilde{C}$ computes the exact same function as C. A weaker, yet practicable variation allows $\tilde{C}$ to approximate C with overwhelming probability *over the coin tosses of the obfuscator*. Barak *et al.* [24] show that VBB obfuscation is impossible for generic function family with exact functionality, and extend the impossibility results for obfuscators preserving approximate functionality. In particular, they show that given one-way functions exist, there exists a family of inherently unobfuscatable functions $\mathcal{F}$ and a predicate $\varphi : \mathcal{F} \to \{0, 1\}$, such that

(i) Given any program that computes a function $f \in \mathcal{F}$, the value $\varphi(f)$ can be efficiently calculated.

(ii) Given oracle access to $f \in \mathcal{F}$, no efficient algorithm can compute $\varphi(f)$.

Nevertheless, the authors leave open the possibility that large classes of programs could still be obfuscated, referring to [41] as a form of special purpose obfuscation. Such favoring assertions were followed by designing efficient obfuscators for

evasive functions such as compute-and-compare programs [88, 202], pattern-matching with wildcards [27, 30], fuzzy matching for Hamming distance [85], etc. We remind the readers that evasive functions are the programs for which it is hard to find an accepting input from the oracle access to the program (see Definition 2.5.2).

In [41], Canetti shows the construction of an efficient obfuscator for point functions (point function evaluates to 1, if and only if the input equals to a particular value, e.g. password check) that achieves a relaxed notion of virtual black-box using a probabilistic hashing algorithm. In [200], Wee designed a VBB obfuscator for point functions with multi-bit outputs.

In [30], Bishop *et al.* design an efficient DVBB obfuscator (see Definition 2.5.1) for evasive conjunction functions using Lagrange's interpolation, while proving its security in the generic group model. Their security goal roughly states that, a PPT adversary cannot distinguish the obfuscation of C from obfuscation of a function that always outputs 0. The construction satisfies "approximate" functionality preservation for an ensemble of uniform distributions. A formal description of their obfuscator is given in Algorithm 8.1. The evaluation procedure (see Algorithm 8.2) takes $x \in \{0,1\}^n$ as input and outputs 1, if it matches the pattern pat and 0 otherwise.

In [88, 202], the authors propose a DVBB obfuscator for compute-and-compare programs under the learning-with-errors (LWE) assumption, parameterized by a polynomial time evasive function f and a target value y, such that on input x the function outputs 1, if $f(x) = y$. As a building block, they encode branching programs based upon directed-encoding scheme inspired by [89].

In [42], authors design a DVBB obfuscator that checks for membership in a hyperplane of constant dimension and analyze its security under a strong variant of decisional Diffie-Hellman problem.

In [63], Crescenzo describes an obfuscator for functions of the form $C_{\phi,s_1,\ldots,s_m}$, where $s_1, \ldots, s_m$ are secret strings and ϕ denotes a monotone formula. Along with hiding the formula gates, the obfuscator hides the secret strings and checks whether an input $x_1, \ldots, x_m$ satisfies $\phi((x_1 = s_1), \ldots, (x_m = s_m))$. The construction depends on indistinguishability notion and a newly introduced *formula indistinguishability* notion for formalizing security.

In [85], Zobernig *et al.* design an efficient DVBB and input-hiding obfuscator for fuzzy matching for Hamming distance, based on a new computational assumption "decisional distributional modular subset product problem". In particular, they obfuscate evasive hamming ball membership programs parameterized by $x \in \{0,1\}^n$, such that on input $y \in \{0,1\}^n$, the obfuscator outputs 1 if and only if y has a hamming distance at most r, where $r < \frac{n}{2}$ determines the actual security level of the obfuscator.

We aim to design special-purpose obfuscators for encoding branching programs, that would be applicable to obfuscating ICS control programs.

Chapter 4

TaDeT: Towards Efficient Tamper Detection

As discussed in Chapter 1, false-data injection is the primary objective of ICS targeted attacks. We consider a threat model, where an MiTM adversary with unauthorized access to ICS network at the supervision layer modifies control parameters exchanged between PLCs and monitoring units/HMIs. Such adversarial attempts are not detected by the HMIs due to lack of built-in security provisions in ICS equipment and protocols. Existing passive IDS techniques are mostly inaccurate (see discussion in Chapter 3), and bump-in-the-wire cryptographic implementations violate the latency requirements of the time-invariant control application. We propose a novel general-purpose framework TaDeT, that exploits the analytical redundancies of the control environment to detect false-data injection attempts of the modeled adversary. Our construction relies upon lightweight cryptographic primitives, and does not utilize the network bandwidth employed for communicating critical control information. Furthermore, TaDeT framework does not require a security personnel to be acquainted with the control design of the target application. We provide a prototype implementation of the proposed construction over an example testbed, launch a series of false-data injection attacks against sensor/actuator readings communicated over the ICS network, and record the delay in attack-detection to determine the efficacy of the proposed construction.

4.1 Rationale

An in-depth analysis of the most powerful and sophisticated cyber-attacks in ICS domain shows that process manipulation or false-data injection is the most dominant objective in targeted attacks, with prominent and impactful consequences [159]. False-data injection attacks are mostly delivered by exploiting the insecurities of the industrial communication protocols [9].

As discussed in Chapter 3, the state-of-art methodologies for detecting control data tampering either involve inaccurate predictive modelling techniques with Intrusion Detection Systems (IDS) [90, 104] or expensive bump-in-the-wire [194, 48] approaches that violate real-time requirements. We are motivated to *find solutions that accurately identify data tampering in control applications without compromising the network bandwidth used for communicating control information*, which is a scarce resource for constrained time-critical applications. This indeed is our problem statement.

Our Contributions. Our emphasis is upon methodologies that perform accurate tamper-detection, such that sophisticated adversarial techniques to evade detection can be effectively captured. To address this concern, we exploit the analytical redundancies of industrial components that are usually deployed in ICS facilities to reduce the impact of potential failures [146]. We leverage the use of customary back-up PLCs [1, 129] (fail-over devices used for the purpose of availability and resilience) and heterogeneous ICS networks with separate threat surfaces [177], to design a tamper-detection framework suitable for constrained control applications. Our framework is (a) *non-intrusive*, requiring minimal changes to the existing architecture, (b) *passive*, not interrupting the critical control communication, (c) *low-cost*, involving minimal additional cost, (d) *accurate* under some assumptions, and (e) *flexible*, independent of vendor specifications, protocol implementations and the nature of the application. Furthermore, *we take into account the scenario, where a process engineer is reluctant in providing a security personnel with the access to control design of the application*; this is admissible and plausible, as control and configuration information are proprietary to an ICS facility, and hence should be secured from industrial espionage. As per [153], 93% of the ICS facilities do not allow a security personnel to be acquainted with the control framework, and our tamper-detection framework is in line with such requirements. In particular, we make the following contributions:

- We introduce a general-purpose framework TaDeT that accurately detects of false-data injection attacks, without compromising the bandwidth used for control data communication.
- We study the performance restrictions necessary for handling errors (false-positives/negatives) in TaDeT. Our proof-of-concept implementation indicates adding a correction parameter δ evaluated empirically in the experimental setup.
- We propose an alternate solution to remove evaluation errors keeping in mind the ICS facilities that do not comply with the stringent performance restrictions imposed by TaDeT.

To the best of our knowledge, this is the first attempt to provide accurate detection of false-data injection attacks without introducing delay in critical control communication.

Organization. This chapter is organized into the following sections. Section 4.2 introduces our threat model, followed by a brief overview of our example testbed. Section 4.3 details the proposed construction. Section 4.4 analyzes the scenarios that lead to false-positives and false-negatives, and presents a broad-level discussion on the stringent performance requirements of the proposed framework. An overall conclusion is given in Section 4.6.

4.2 Preliminaries

In this section, we discuss the motivation behind this research and introduce the threat model that we consider for this study. Furthermore, we give a brief description of our example testbed, which we use as a running example to demonstrate our approach.

4.2.1 Problem Formulation

In this section, we analyze the threat surfaces of heterogeneous ICS networks and bring to light the most vulnerable network segment for false-data injection attacks. We also discuss the acceptable sampling rates in diverse control applications, and the incapacities of the existing recommendations to fit into such time-constrained paradigm.

The Purdue model is used for segmenting the ICS networks based on functionality and nature of control [125]. We remind the readers that control layers exchange real-time signals, and the control commands are preset in the hardware configuration of the devices. For an adversary who has unauthorized access to the network at this layer, require detailed understanding of the process design and implementation specifications for parsing and manipulating the packets. However, packets at the supervision layer contain rich semantic application-specific information, making it less arduous for an adversary interpret the payloads. Thus, it is safe to establish that ICS network at the supervision layer is more vulnerable to being compromised, and hence we focus on a threat model where adversary has unauthorized access to the network at this layer.

A control application with unacceptable delay is no longer time-invariant. While applications such as food processing could allow delay of up to 8 milliseconds, safety-critical applications, such as turbo-machinery, magnetic suspension systems allow up to 1 millisecond delay [96]. Thus, an interesting question to ask is how feasible are the existing accurate tamper-detection techniques with cryptographic implementations in complying with the aforementioned constraints?

This encourages us to investigate into *practical and achievable solutions for generic ICS applications, who have their supervision layer susceptible to false-data injection attacks.*

4.2.2 Threat Model

Our threat model incorporates a Man-in-the-Middle (MitM) adversary who has a basic understanding of how ICS control, configuration, and diagnostic data moves between the PLCs and SCADA systems. The adversary is present at the industrial site and has access to the plant communication network at the *supervision layer*. The adversary can set up a physical device at any point in this network and has tools to interact with the ICS protocols and interpret the semantics of the ICS payloads. The ultimate goal of the adversary is to intercept the information exchanged between these devices and cause false data injection attacks. Furthermore, we assume that the adversary does not have access to the control layer; this is admissible, as attack vectors with a lower relative likelihood (see Section 4.2.1) could be ignored [185].

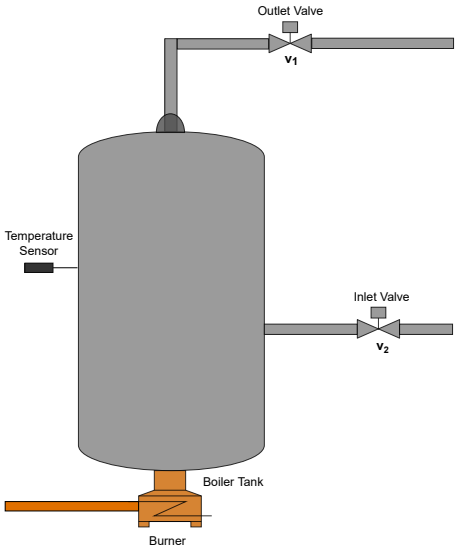


Figure 4.1: Experimental Testbed: Schematics of an Industrial Steam Boiler Application

4.2.3 A Plausible Scenario

In this section, we introduce a simple experimental steam boiler application, which we use as a running example to evaluate our approach.

Industrial Steam Boiler Application. We design a simple realistic steam boiler application, which is key to critical industrial utilities such as thermal power plants, chemical manufacturing, etc. As evident from Figure 4.1, a steam boiling tank is connected to valve v_2, which supplies cool water to the tank, and a burner which heats the water in the tank to its boiling point. When water vaporizes, the outlet valve v_1 releases the steam from the tank. A temperature sensor with a transmitter measures the temperature inside the tank. When the temperature (*temp*) reaches the level t_1, valve v_1 is opened to allow the steam to be released. Again when *temp* reaches t_2, valve v_2 closes immediately to prevent inflow of water, otherwise the cold air can lead to the boiler flashing into steam, followed by an explosion.

Composing Attack Vectors. We discuss an example attack scenario, where the modeled adversary explodes the steam boiler with fatal consequences. SCADA unit explicitly requests for the value of the temperature sensor, and based on the status, sends command to the PLC to stop inflow of water through valve v_2 ($v_2 = 0$). The adversary intercepts the packet, overrides the value ($v_2 = 1$) and

sends the modified command to the PLC. This eventually causes the steam boiler to explode causing severe consequences.

4.3 Proposed Framework for Detecting False-Data Injection Attacks

In this section, we introduce TaDeT, a novel general-purpose framework for accurate tamper detection at the *supervision layer* of a process control framework.

4.3.1 Brief Overview

TaDeT is designed for addressing the challenges that come with implementing state-of-art recommendations on accurate tamper detection (see Section 3.1.2). The starting point of this construction is the customary back-up PLCs that get periodic updates from primary PLCs. For ICS facilities with limitations on redundant PLCs, an identical functionality can be achieved using commercial off-the-shelf hardware modules, such as Raspberry Pi [169], Arduino [163], etc. The other important consideration that we take advantage of, is the fast computations supported by SCADA units, which are usually deployed using sophisticated high-end reliable computing platform.

TaDeT is built on top of the basic process control framework described in Section 2.1. TaDeT provides a parallel computing platform due to an added network module between the SCADA unit and back-up PLC, supporting cryptographic operations. This enables the possibility of designing a tamper detection system, which gets its accuracy and efficiency from the lightweight cryptographic primitives introduced over a parallel computing environment, while incurring no overhead in the critical control path. The high-level schematics of TaDeT architecture is given in Figure 4.2. The proposed framework is completely agnostic to the underlying ICS protocols, vendor implementations of ICS equipment, and the nature of control applications. Furthermore, TaDeT requires minimum changes to the existing architecture, restricting a security personnel to be oblivious to the functional behaviour of the process control.

4.3.2 System Architecture

Consider a process control framework Π, where sensors are time-driven, and actuators, controllers and SCADA units are event-driven. Let the control layer

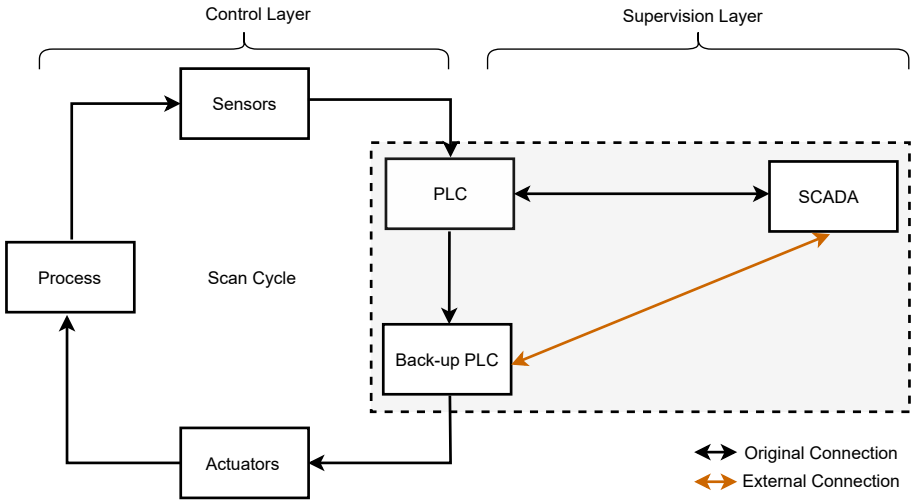


Figure 4.2: Block diagram of TaDeT architecture.

of Π contains n control elements (sensors and actuators), a primary PLC and a back-up PLC, denoted by $\mathcal{P}_a$ and $\mathcal{P}_b$ respectively, all connected in a ring topology. At cycle κ, $\mathcal{P}_a$ stores the sequence $\mathcal{X}^\kappa = (x_1^\kappa, \ldots, x_n^\kappa)$, where x_i^κ denotes the value of ith control element at the κth cycle, and forwards $\mathcal{X}^\kappa$ to $\mathcal{P}_b$. Let $\mathcal{S}$ denote the SCADA unit, which is connected to $\mathcal{P}_a$ over the supervision layer of Π. $\mathcal{S}$ queries $\mathcal{P}_a$ with Read and Write requests on $\mathcal{X}^\kappa$ over the critical control path, denoted by $\mathcal{P}_a \Leftrightarrow \mathcal{S}$. This topology is inspired by the network segmentation of SWaT testbed [129], an experimental facility that imitates a real ICS.

TaDeT extends the basic control architecture by adding a separate network module between $\mathcal{P}_b$ and $\mathcal{S}$, denoted by $\mathcal{P}_b \overset{e}{\Leftrightarrow} \mathcal{S}$, that allows implementing cryptographic operations. Let λ be the security parameter of the system. Let (Enc, Dec) be a symmetric encryption scheme, and let $\mathsf{k} \leftarrow \mathsf{Gen}(1^\lambda)$ where Gen is a key generation algorithm to generate the secret key shared between $\mathcal{P}_b$ and $\mathcal{S}$.

TaDeT implements the protocols TaDeT^{read} and TaDeT^{write} for efficient and accurate tamper detection during Read and Write requests on $\mathcal{X}^\kappa$. We first present a high-level survey of the protocols, followed by their formal description. Note that, any run of the protocols starts with $\mathcal{S}$. We assume that, at the start of the protocols, $\mathcal{P}_a$ and $\mathcal{P}_b$ are in the κth cycle, storing a copy of $\mathcal{X}^\kappa$.

TaDeT^{read}: For *reading* a specific control element, say x_i^κ, $\mathcal{S}$ makes two simultaneous Read requests to $\mathcal{P}_a$ and $\mathcal{P}_b$. While $\mathcal{P}_a$ replies with x_i^κ over $\mathcal{P}_a \Leftrightarrow \mathcal{S}$, $\mathcal{P}_b$

returns $\mathsf{Enc}_{\mathsf{k}}(x_i^{\kappa})$ over $\mathcal{P}_b \overset{e}{\Leftrightarrow} \mathcal{S}$. $\mathcal{S}$ issues an alert signal if there is a data mismatch. The formal description is given in Protocol 4.1.

Protocol 4.1 TaDeT^{read}

1: $\mathcal{S}$ queries $\mathcal{P}_a$ and $\mathcal{P}_b$ simultaneously for *reading* the ith control element.
$\mathcal{S} \xrightarrow{\mathsf{Read}(i)} \mathcal{P}_a$
$\mathcal{S} \xrightarrow{\mathsf{Read}(i)} \mathcal{P}_b$

2: $\mathcal{P}_a$ and $\mathcal{P}_b$ reply to $\mathcal{S}$.
$\mathcal{S} \xleftarrow{\delta_a = x_i^{\kappa}} \mathcal{P}_a$
$\mathcal{S} \xleftarrow{\delta_b = \mathsf{Enc}_{\mathsf{k}}(x_i^{\kappa})} \mathcal{P}_b$

3: $\mathcal{S}$ correlates the control values.
(1) $\mathcal{S} : \delta_b' \leftarrow \mathsf{Dec}_{\mathsf{k}}(\delta_b)$
(2) $\mathcal{S}$: **if** $(\delta_a == \delta_b')$ **return** $\perp$
else issue SYSTEM.ALERT

TaDeT^{write}: $\mathcal{S}$ sends Write requests to $\mathcal{P}_a$ and $\mathcal{P}_b$ simultaneously, by sending $\tilde{x_i}^{\kappa}$ and $\mathsf{Enc}_{\mathsf{k}}(\tilde{x_i}^{\kappa})$ respectively. $\mathcal{P}_a$ updates $\mathcal{X}^{\kappa}$ with $\tilde{x_i}^{\kappa}$, call it $\tilde{\mathcal{X}}^{\kappa}$, and forwards it to $\mathcal{P}_b$. $\mathcal{P}_b$ decrypts $\mathsf{Enc}(\tilde{x_i}^{\kappa})$ received over $\mathcal{P}_b \overset{e}{\Leftrightarrow} \mathcal{S}$, and correlates it with the ith entry of $\tilde{\mathcal{X}}^{\kappa}$. We formally describe the procedure in Protocol 4.2.

Protocol 4.2 TaDeT^{write}

1: $\mathcal{S}$ sends $\tilde{x_i}^{\kappa}$ to $\mathcal{P}_a$ and $\mathsf{Enc}_{\mathsf{k}}(\tilde{x_i}^{\kappa})$ to $\mathcal{P}_b$ simultaneously for *writing* the ith control element.
$\mathcal{S} \xrightarrow{\mathsf{Write}(\tilde{x_i}^{\kappa})} \mathcal{P}_a$
$\mathcal{S} \xrightarrow{\mathsf{Write}(\mathsf{Enc}_{\mathsf{k}}(\tilde{x_i}^{\kappa}))} \mathcal{P}_b$

2: $\mathcal{P}_a$ updates $\mathcal{X}^{\kappa}$ to $\tilde{\mathcal{X}}^{\kappa}$ and forwards it to $\mathcal{P}_b$.
$\mathcal{P}_a \xrightarrow{\tilde{\mathcal{X}}^{\kappa}} \mathcal{P}_b$

3: $\mathcal{P}_b$ decrypts $\mathsf{Enc}_{\mathsf{k}}(\tilde{x_i}^{\kappa})$ and correlates it with the ith element in $\tilde{\mathcal{X}}^{\kappa}$.
(1) $\mathcal{P}_b : \gamma_a \leftarrow \tilde{x_i}^{\kappa} \in \tilde{\mathcal{X}}^{\kappa}$
(2) $\mathcal{P}_b : \gamma_b \leftarrow \mathsf{Dec}_{\mathsf{k}}(\mathsf{Enc}_{\mathsf{k}}(\tilde{x_i}^{\kappa}))$
(3) $\mathcal{P}_b$: **if** $(\gamma_a == \gamma_b)$ **return** $\perp$
else issue SYSTEM.ALERT

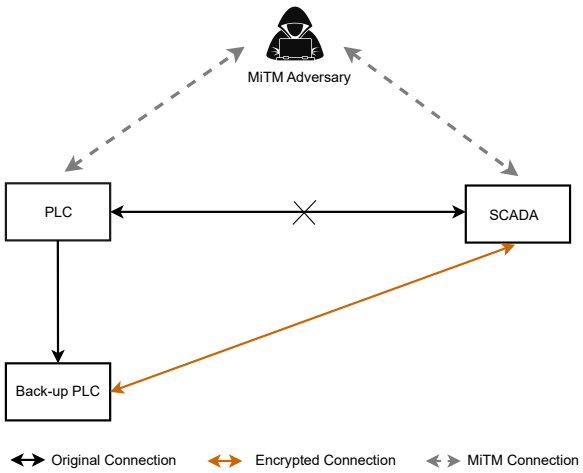


Figure 4.3: High level schematics of the MiTM adversary who manipulates read/write requests via unauthorized network access to the *supervision* layer.

Our key management paradigm is loosely based on the Industrial Key Infrastructure (IKI) concept considered in [14]. We assume the existence of a Trust Center, an authority completely controlled by the control infrastructure. The Trust Center configures a secret key in the PLC and SCADA devices using a secure network, where an adversary does not have access.

4.3.3 Add-on Software Modules

TaDeT requires incorporating additional software components in back-up PLC and SCADA units to facilitate proper execution of TaDeT^{read} and TaDeT^{write} protocols. Furthermore, the add-on modules need to coordinate with the basic software components discussed in Section 2.1, without introducing delay in critical control communication. We discuss the extension modules for back-up PLCs, though a similar design needs to be incorporated in SCADA units.

Back-up PLCs are *central* to the design of proposed construction, acting as edge-devices that connect with primary PLCs over the control layer and SCADA units over the supervision layer. These devices have limited storage and computational capabilities due to vendor restrictions, and hence proper care needs to be taken while configuring them. In what follows, we discuss the extended software architecture of back-up PLCs.

As evident from Figure 4.4, the extended architecture of back-up PLCs comprises of *five* add-on software components. The *encryption* module encrypts control

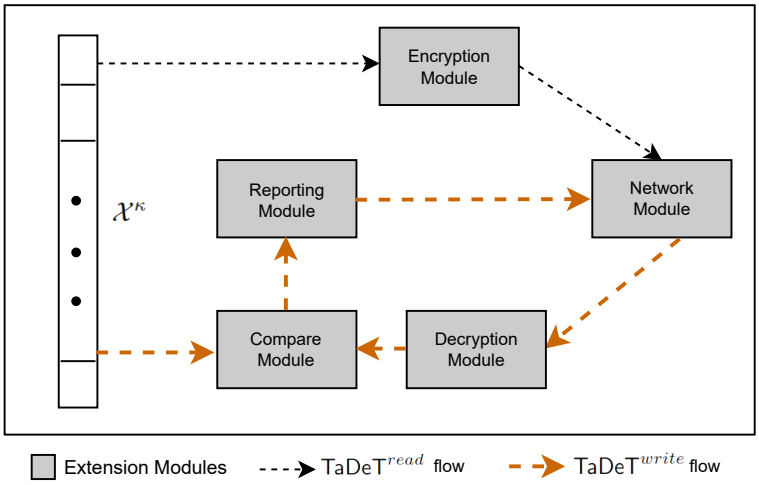


Figure 4.4: Add-on software modules executed in the back-up PLC during the run of TaDeT^{read} and TaDeT^{write} protocols. $\mathcal{X}^{\kappa}$ denotes the process image table updated from primary PLC during scan-cycle.

elements requested by SCADA unit during the run of TaDeT^{read} protocol and sends them to *network* module, that acts as an interface between back-up PLC and the SCADA unit. During the execution of TaDeT^{write} protocol, the *decryption* module decrypts the encrypted control elements received from SCADA unit through *network* module, and forwards it to the *compare* module, which then correlates the values with the corresponding control elements in process image table. The *reporting* module generates system alert, if data mismatch occurs.

4.4 Attack Detection

As discussed in Section 4.2.2, the modeled adversary focuses on destabilizing critical control applications by falsification of control commands. Our construction detects such efforts of an adversary, who modifies commands that control the state of the physical system. In an *ideal* scenario, the attack detection should take place *instantaneously*. A serious hurdle that prevents this strategy is the delay introduced due to the computations in the extension modules. The other important concern to keep in mind is that an MiTM attack involves *change in response time due to delay added by the adversary who intercepts, processes and re-transmits each packet* communicated between the two parties [221].

Let $\mathcal{A}$ be an instance of the modeled adversary, who has unauthorized access to

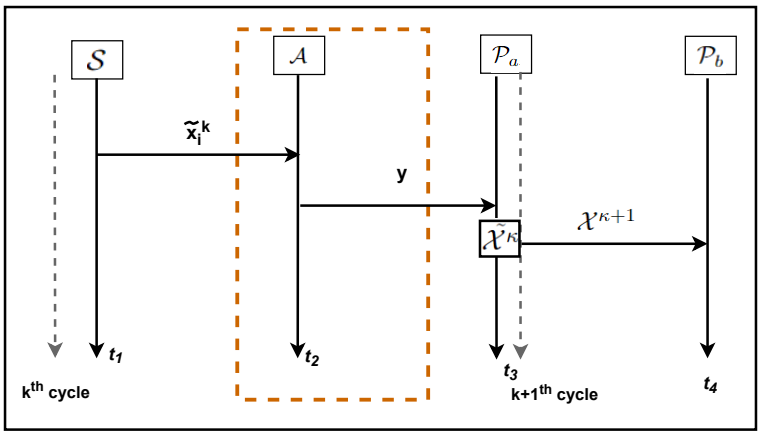


Figure 4.5: Scenario that leads to *false positive* during execution of TaDeT^{write}.

$\mathcal{P}_a \Leftrightarrow \mathcal{S}$. Note that, during the run of the Protocols 4.1 and 4.2, $\mathcal{P}_a$ and $\mathcal{P}_b$ are in the κth cycle and contains a copy of $\mathcal{X}^\kappa$.

Remark 4.1 *Consider Protocol 4.1. Suppose $\mathcal{A}$ modifies δ_a at Step 2 of the protocol i.e. $\mathcal{A} \xrightarrow{\delta_a = y} \mathcal{S}$. Let Δ_a be the delay introduced by $\mathcal{A}$ due to the modification, and let Δ_b be the time taken for executing for $\mathcal{S} \xleftarrow{\delta_b = \mathsf{Enc_k}(x_i^\kappa)} \mathcal{P}_b$ and Step 3 of Protocol 4.1. Then* TaDeT^{read} *accurately detects an attack with an overall delay of $\Delta = |\Delta_b - \Delta_a|$.*

Remark 4.2 *Consider Protocol 4.2. Suppose $\mathcal{A}$ modifies $\tilde{x_i^\kappa}$ over $\mathcal{P}_a \Leftrightarrow \mathcal{S}$, and let σ_a be the delay introduced by $\mathcal{A}$. Let σ_b be the time taken by $\mathcal{P}_a$ and $\mathcal{P}_b$ to update $\tilde{\mathcal{X}}$, and let σ_c be the time for Step 2 and Step 3 of Protocol 4.2 respectively. Then* TaDeT^{write} *detects an attack with an overall delay of $\sigma = |\sigma_c - (\sigma_a + \sigma_b)|$.*

4.4.1 Necessary Conditions for Accurate Attack Detection

We avoid delay introduced due to physical components and electrical/electronic noise in our calculation. In what follows, we discuss the scenario that leads to false positives.

At the start of Protocol 4.2, consider $\mathcal{P}_a$ and $\mathcal{P}_b$ to be at the κth cycle, as demonstrated in Figure 4.5. Let $\mathcal{A}$ modifies $\tilde{x_i^\kappa}$ at time instant t_2. Let $(\kappa+1)$th cycle starts at time instant t_3. $\tilde{\mathcal{X}}^\kappa$ is overwritten by $\mathcal{X}^{\kappa+1}$ and $\mathcal{P}_a$ forwards $\mathcal{X}^{\kappa+1}$ to $\mathcal{P}_b$.

- If $\tilde{x_i^\kappa} = x_i^{\kappa+1}$, TaDeT^{write} does not detect an attack. This can be disregarded as $\mathcal{A}$ does not succeed in modifying the control value at the control layer.

- If $\tilde{x_i^\kappa} \neq x_i^{\kappa+1}$, TaDeT^{write} correctly detects an attack. However, this is not due to the spoofed control value by the adversary, but due to the acquired periodic update during the scan-cycle.

Consider there is no attack, i.e. $\mathcal{A}$ does not modify $\tilde{x_i^\kappa}$ and let $(\kappa+1)$th cycle starts at time instant t_3.

- If $\tilde{x_i^\kappa} = x_i^{\kappa+1}$, TaDeT^{write} correctly issues no alert.
- If $\tilde{x_i^\kappa} \neq x_i^{\kappa+1}$, TaDeT^{write} incorrectly detects an attack, which is a *false positive*.

Thus for TaDeT *to accurately detect an MiTM attack, we need to avoid the situation where $\tilde{\mathcal{X}}^\kappa$ is overwritten by $\mathcal{X}^{\kappa+1}$.* Throughout the rest of the chapter, we assume that during the entire run of TaDeT and the MiTM adversary, the control application is in its κth scan-cycle.

4.5 Prototype Implementation

In this section, we present a prototype implementation of the proposed approach over an experimental testbed (introduced in Section 4.2.3) and an in-depth analysis of our empirical evaluation.

4.5.1 Experimentation Setup

We implement the proposed TaDeT framework using a simulation setup in *three* Raspberry Pi 4 Model B [170] devices, whose specifications are listed in Table 4.1. The Raspberry Pi's are used to model the PLC, back-up PLC and the modeled MiTM adversary (see Section 4.2.2). We implement the SCADA monitoring unit in a computer with an Intel(R) Core(TM) i7-9750H CPU rated at 2.60 GHz, 8 GB RAM (see Table 7.2 for configuration details). We setup TCP and UDP socket programming [134] (following the network configurations used in SWaT testbed [129], which imitates a real-world ICS facility) in Python 3.8.2 [164] (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AMD64)] on win32 between the devices listed above, following the design in Figure 4.3, in *wireless* mode. We use ChaCha-20 [147] implementation from PyCryptodome cryptographic library to establish the encrypted communication between back-up PLC and SCADA monitoring unit.

Table 4.1: Implementation Details of the Raspberry Pi's used as PLC, back-up PLC, and MiTM adversary

Artifact	**Configuration**
Platform	Raspberry Pi 4 Model B
Hardware	Broadcom BCM2711
Connection	2.4 GHz and 5.0 GHz IEEE 802.11ac wireless
Processor	Quad core Cortex-A72 (ARM v8)
Operating System	Raspbian GNU Linux 10 (buster)

4.5.2 Methodologies

We make use of the example application introduced in Section 4.4 to measure the effectiveness of the proposed TaDeT framework. In particular, we aim to determine if TaDeT accurately detects an MiTM attack. However, there could be subtleties involved: the delay added by the adversary due to processing, parsing and modifying the packets could lead to false-negatives, where the *compare* module compares out-of-sync data with the freshly decrypted packets forwarded from the *network* module. In our experiments, we execute TaDeT^{write} over the simulated environment and deploy the add-on extension modules in the back-up PLC. We design *three* test-cases to determine if there are false-negatives due to additional delay introduced by the MiTM adversary. Note that, we assume that the entire run of TaDeT^{write}is within $t_{\kappa+1} - t_{\kappa}$, where t_{κ} and $t_{\kappa+1}$ be the time instances at the start of κth and $(\kappa + 1)$th cycle respectively. This ensures that there are no false-positives due to $\tilde{\mathcal{X}}^{\kappa}$ being overwritten by $\mathcal{X}^{\kappa+1}$ (see discussions in Section 4.4.1). Empirical evaluations with TaDeT^{read} require a similar procedure with the add-on extension modules deployed in the SCADA unit.

We start with describing the functional requirements of our experimental testbed. The industrial steam boiler application introduced in Section 4.2.3 comprises of *two* control elements: a temperature sensor *temp*, and an actuating valve v_1. The functional requirement of the application is given as follows:

```
IF temp > 120 THEN v1 := 0 ELSE v1 := 1;
```

We execute the TaDeT^{write} protocol and launch MiTM attacks during the run of the protocol in the following settings:

- Case **A** : $\mathcal{S}$ sends control data to $\mathcal{P}_a$ over $\mathcal{P}_a \Leftrightarrow \mathcal{S}$ and $\mathcal{P}_b$ over $\mathcal{P}_b \overset{e}{\Leftrightarrow} \mathcal{S}$ *simultaneously.*
- Case **B** : $\mathcal{S}$ sends control data to $\mathcal{P}_a$ over $\mathcal{P}_a \Leftrightarrow \mathcal{S}$, adds $\mathsf{delay} = 0.1$ seconds delay, and then sends data to $\mathcal{P}_b$ over $\mathcal{P}_b \overset{e}{\Leftrightarrow} \mathcal{S}$.
- Case **C** : $\mathcal{S}$ sends control data to $\mathcal{P}_a$ over $\mathcal{P}_a \Leftrightarrow \mathcal{S}$, adds $\mathsf{delay} = 0.2$ seconds delay, and then sends data to $\mathcal{P}_b$ over $\mathcal{P}_b \overset{e}{\Leftrightarrow} \mathcal{S}$.

In Case **A**, we do not add any delay in sending control data to the $\mathcal{P}_a$ and $\mathcal{P}_b$, i.e. $\mathcal{S}$ sends the control data $\tilde{x_i}$ and $\mathsf{Enc}_{\mathsf{k}}(\tilde{x_i})$ simultaneously. For Case **B** and Case **C**, we add 0.1 and 0.2 seconds of delay. Our objective is to identify if TaDeT^{write} accurately detects whether there is an MiTM attack for all the test-cases.

For the timing experiment, we calculate the delay added by the MiTM adversary and the extension modules, while we do not include the time taken for communicating the control updates between the devices. Table 4.2 shows the specific delays we have considered for generating our results.

Table 4.2: Components for Delay Calculation

Delay	**Description**
σ_0	$\mathcal{A}$ processes, parses and modifies control data in a packet
σ_1	$\mathcal{P}_a$ processes a packet and updates $\tilde{\mathcal{X}}^{\kappa}$.
σ_2	$\mathcal{P}_b$ processes the updated $\tilde{\mathcal{X}}^{\kappa}$ forwarded by $\mathcal{P}_a$.
σ_3	$\mathcal{S}$ executes *encryption* module
σ_4	$\mathcal{P}_b$ executes *decryption* module

We consider a simple scenario (depicted in Figure 4.6), where $\mathcal{S}$ sends the request to update $v_1 = 0$ to $\mathcal{P}_a$ and $\mathcal{P}_b$. We launch an MiTM attack over $\mathcal{P}_a \Leftrightarrow \mathcal{S}$, where $\mathcal{A}$ modifies v_1 to 1. $\mathcal{P}_a$ receives $v_1 = 1$, updates $\mathcal{X}^{\kappa}$ to $\tilde{\mathcal{X}}^{\kappa}$ and forwards it to $\mathcal{P}_b$. $\mathcal{P}_b$ decrypts $\mathsf{Enc}_{\mathsf{k}}(0)$ sent by $\mathcal{S}$, processes $\tilde{\mathcal{X}}^{\kappa}$ sent by $\mathcal{P}_a$, correlates the values.

We conduct 14 individual trials (approximated by Experiment ID) for each of the above test-cases and calculate the timing components described in Table 4.2 to empirically evaluate the suitable conditions for accurate attack detection by the TaDeT^{write} protocol.

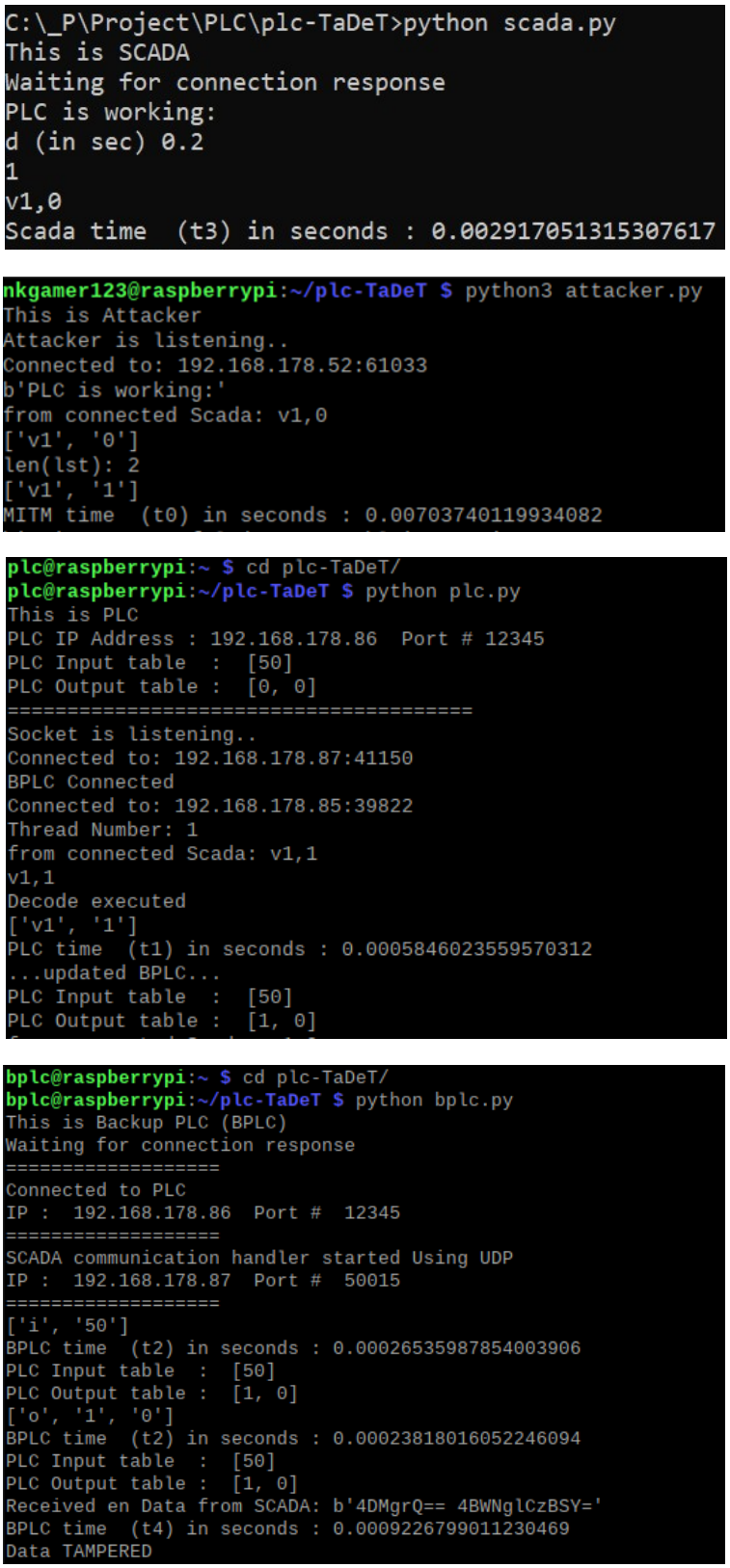


Figure 4.6: Recorded screenshots for each of the devices during the run of an experiment for Case **C**.

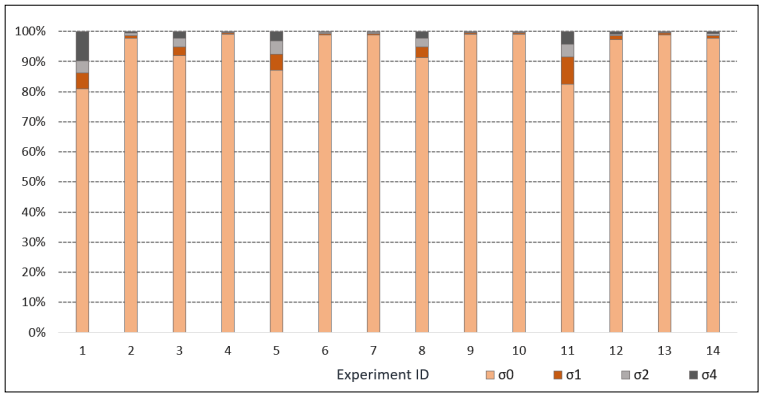


Figure 4.7: Time components calculated for Case **A**. We do not show σ_3 as the time taken to execute the *encryption* module by SCADA unit is approximated to *zero* for all the 14 trials.

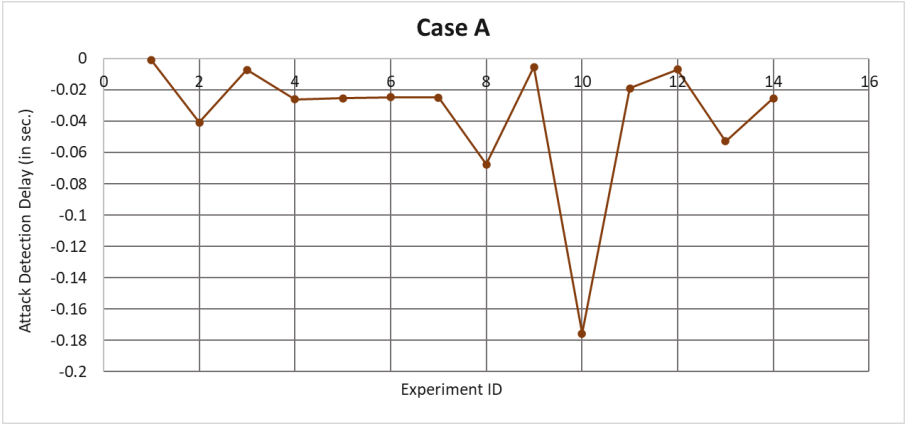
4.5.3 Results

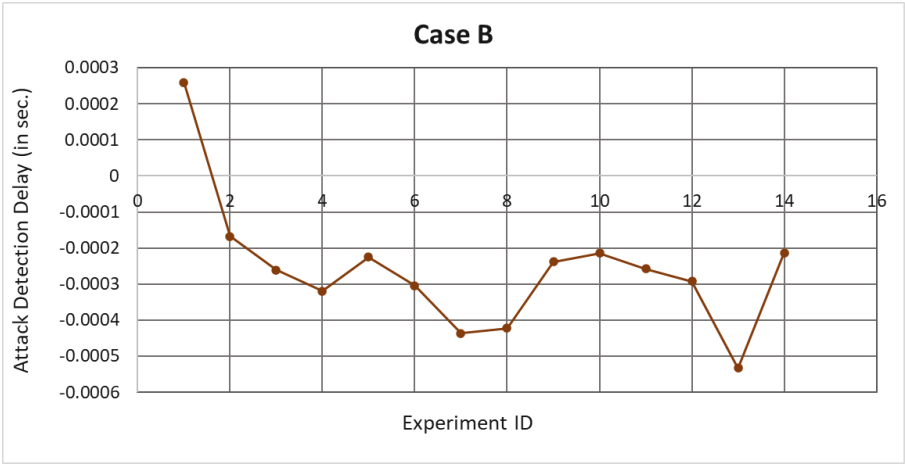
We now present the experimental outputs, along with an in-depth analysis of our evaluations.

Our results indicate that TaDeT^{write} accurately detects an MiTM attack for Case **C**, 0.07% of the attacks for Case **B**, while all the test trials for Case **A** generate inaccurate results. Note that when $(\sigma_0+\sigma_1+\sigma_2 > \sigma_3+\sigma_4)$, TaDeT^{write} produces inaccurate results. To see why, $\mathcal{P}_b$ compares $\mathsf{Dec}_\mathsf{k}(\mathsf{Enc}_\mathsf{k}(0))$ with the corresponding value in $\mathcal{X}^\kappa$, rather than the updated table $\tilde{\mathcal{X}}^\kappa$, generating a *false-negative* if $v_1 = 1$ in $\mathcal{X}^\kappa$.

As can be seen from Figure 4.8, the delays corresponding to the 14 test trials in Case **A** are represented along the negative x-axis, indicating inaccurate results. For Case **B**, one of the trials generate inaccurate output. Finally, for Case **C**, the delays corresponding to all the trials are plotted along the positive x-axis, indicating that the generated outputs are accurate.

We do a closer inspection on the results generated by Case **A** to identify the time component that primarily contributes to the inconsistencies. As can be seen from Figure 4.7, σ_0 accounts for more than 98% of $\sigma_0+\sigma_1+\sigma_2$. Furthermore, σ_0 is around 80% of the overall time for executing *encryption* and *decryption* modules for Case **A**, 98.8% for Case **B** and 97% for Case **C**. This invariably indicates the significance of adding delay in generating accurate results by the proposed framework. We





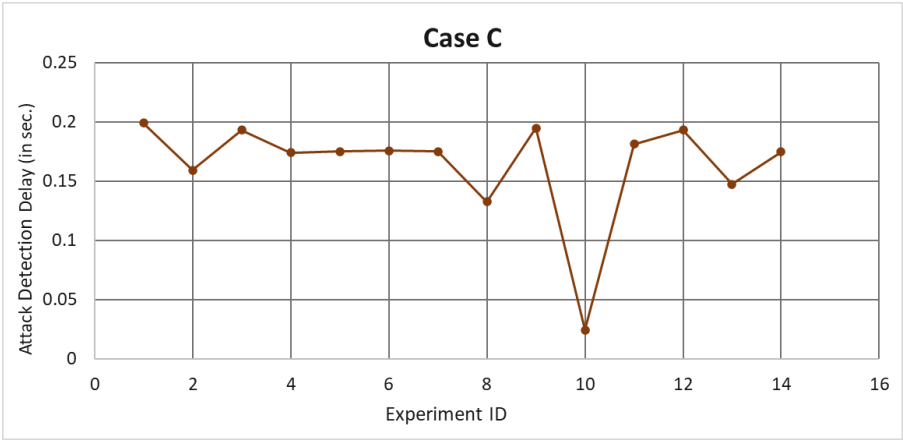


Figure 4.8: Delay in attack detection for Case **A** **B** and **C**.

note that on implementing the TaDeT framework in a real-world ICS setup with PLCs and back-up PLCs, the presented numbers are expected to vary.

Of note, that for accurate tamper detection, a suitable value for delay would depend upon the specific choice of lightweight ciphers and their implementations, scan-cycle time of the control application, and configuration of PLC and back-up PLC deployed in the control environment.

4.5.4 Discussions

As evident from our experimental results, the proposed TaDeT framework imposes stringent performance restrictions for accurate detection of false-data injection attacks. In particular, for handling the scenarios that lead to false-positives/negatives, we essentially require the back-up PLC / SCADA to correlate correct values during the execution of TaDeT^{Write} / TaDeT^{Read} protocol. This is our intuition behind adding delay parameter σ/δ evaluated empirically in the experimental setup, conditional to the availability of the following:

- Lightweight cryptographic ciphers
- SCADA's ability to perform fast computations
- Back-up PLC's ability to perform fast computations

A number of free and open-source cryptographic libraries offering lightweight implementations (e.g. Crypto $++$, libsodium) are available. Additionally, SCADA is generally deployed with a high-end reliable fast computing platform. For our SCADA implementation with an Intel(R) Core(TM) i7-9750H, the encryption/decryption time is approximated to be zero for all the 14 trials.

The only caveat is the availability of a back-up PLC with fast computational capabilities (see Section 1.2). Allowing the back-up PLC to be flexible in terms of performance requirements, we propose an alternate solution for accurate detection of the false-data injection attacks : given the back-up PLC does not host the control program, the program memory (accounting for over 95% of the PLC memory) is free, and hence registers could be allocated to store the correct values, such that correlation is effectively performed by the back-up PLC.

4.6 Conclusion

In this chapter, we have presented a framework TaDeT that identifies control data tampering at the supervision layer of an industrial control infrastructure. Our solution is generic, in the sense that it is agnostic to the specific implementations of ICS protocols, equipment, and nature of control applications. Our construction employs lightweight cryptographic primitives, and does not compromise the bandwidth used for critical control communication. Our experimental evaluations indicate that the proposed framework gives accurate results based on a careful selection of the delay parameter. As our future initiative, we aim to extend our framework in detecting interception of control data exchanged over the ICS network at the supervision layer.

Chapter 5

SelEnc: A Selective Encryption Framework for Securing ICS Payloads

ICS implement a distributed process control framework with legacy controllers and proprietary protocols, enabling a wide range of cyber-attacks. The ICS research community and industrial security practitioners recommend implementing TLS/DTLS or bump-in-the-wire techniques for communicating confidential information. In this chapter, we discuss how such techniques fail to provide application-level security required in control applications. We examine the proprietary application-layer protocols and how they access controller memory for performing read/write operations on the process control parameters, and claim that custom-made ICS security solutions require application-level access to the controller. To this end, we propose SelEnc, a general-purpose modular framework that provides application-specific security while incurring significantly low overhead. We provide a proof-of-concept implementation of the proposed framework over an example testbed and evaluate our construction with *two* use cases and *five* different datasets. Our micro-benchmarks indicate a significant reduction in computational overhead (less than 1.5% of the overhead incurred due to TLS and other state-of-art approaches), with guarantees of application-level security acknowledged at the target control environment.

5.1 Rationale

The efforts by the ICS research community in imparting cryptographic solutions come in the form of bump-in-the-wire techniques that rely upon the security services offered at transport, network, and data-link layers [97, 48]. IEC 62351-3 standardizes using TLS/DTLS in ICS network communication [157]. Though TLS/DTLS is effective in securing IT communications, its features are not optimal for use in a generic ICS settings [14]. A typical control framework might have customized application-level security requirements such as *'perform encryption only if read/write request about valve v is being communicated'*. TLS 1.3 imposes mandatory encryption for all application-layer payloads, and hence not applicable in this scenario. Also, features such as AEAD and perfect secrecy are not a defacto requirement in a generic ICS facility [149]. Finally, TLS is an expensive solution, in terms of the complexities involved, and thus not suitable for deploying in time-critical applications [14].

Our Contributions. We emphasize upon the importance of ICS security implementations at the application-level of the OSI model. Given that the vendors do not provide application-level access in the PLC, there are no solutions for implementing customized security controls in ICS environment. Through our work, we imply that if vendors provide some minimal controller-level access, then not only application-level security could be achieved, but also would incur an overhead which is less than 1.5% of the complexity introduced due to TLS, IPSec and other existing solutions. A summary of our contributions is given as follows:

- We discuss the significance of application-specific security in ICS facilities, and the downsides of the existing solutions in achieving the same.
- We perform an in-depth analysis of the message format of EtherNet/IP [102], one of the most popular ICS protocols[1]. [151]
- We study the memory structure and communication interfaces in Allen Bradley controllers [17] (one of the most popular controllers [136]) to understand how EtherNet/IP accesses its memory to perform read/write operations on the process [1].

[1]We note that though there are documents on how the EtherNet/IP protocol works and how Allen Bradley controllers are accessed, they have not been explored before in the context of securing ICS communication, to the best of our knowledge.

- We introduce SelEnc, a general-purpose modular framework that filters critical information at the application level, bypasses the critical information over an unused TCP port, and encrypts the traffic at the network layer.
- We evaluate our approach over an example chemical mixing facility with *two* use cases and five different datasets (demonstrating the scalability of our construction).

Our solution can be *extended to other ICS protocols*, with little modifications. Our exclusive goal behind this work is to *indicate the significance of the proposed solution to the PLC vendors, to imply that minimal application-level filtering capabilities could not only allow the customers to implement purpose-built security solutions for their target control applications, but also incur around* 1.5% *of the overheads caused due to TLS/IPSec and other existing technologies.*

Organization. This chapter is organized into the following sections. Section 5.2 provides a brief description of the example testbed, followed by an introduction to the threat model. Section 5.3 provides a general overview of message format of EtherNet/IP and protocol address space of Allen-Bradley controllers. Section 5.4 details the proposed construction. Section 5.5 provides a proof-of-concept implementation of our design. An overall conclusion is given in Section 5.6.

5.2 Preliminaries

We start the section with a brief description of a chemical mixing facility, which we use as a running example to demonstrate our approach. We follow this with an overview of the problem statement and the threat model that we consider for this study.

5.2.1 Example Testbed

Figure 1 illustrates the process for mixing two chemicals, A and B. When the *START* button is pressed, valve v_1 opens and chemical A fills the tank till the level reaches the L_2. Thereafter, valve v_2 opens and chemical B fills the tank till it reaches level L_1. This is followed by an agitator v_3 mixing the two chemicals for 60 seconds, after which a mixture outlet valve v_4 drains out the mixture for a duration of 120 seconds. When an error occurs, an Emergency Stop button stops the entire process.

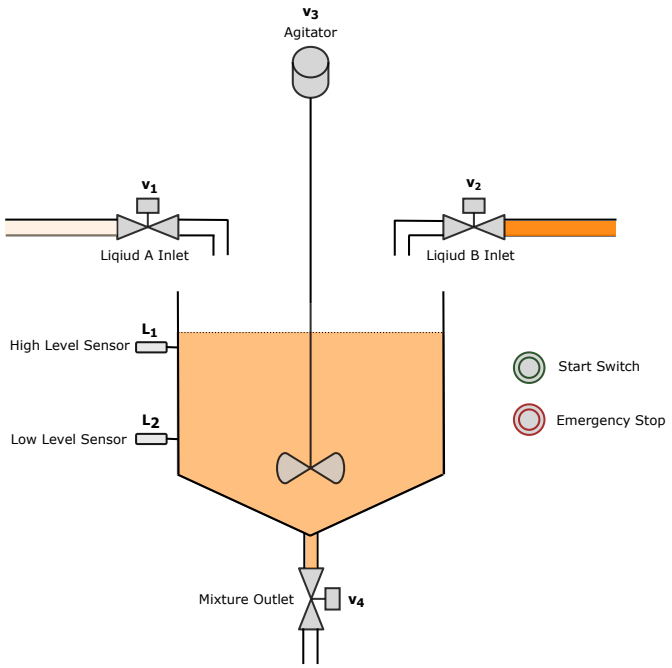


Figure 5.1: Experimental Testbed: Schematics of a Chemical Mixing Facility

5.2.2 Problem Formulation

As discussed in Section 2.1, PLC communicates the I/O values to the HMI systems at the supervision layer of process control framework for advanced control and monitoring of the target control application. The packets exchanged at this layer contain rich semantic information on the control parameters, whereas the packets at the control layer follow non-standard application-level semantics defined by vendors and process engineers that is preset in the hardware configuration of the devices. Thus, a MitM adversary finds it more challenging to recover the semantics of the ICS payload in the control layer, as it requires a detailed understanding of the process design and implementation specifications [196]. The aim of this study is to design legacy-compliant solutions for achieving application-specific security in exchanging confidential messages at the *supervision* layer of a generic ICS framework.

5.2.3 Threat Model

Our threat model incorporates a MitM adversary who has a basic understanding of how ICS control, configuration, and diagnostic data moves between the PLCs and SCADA systems. The adversary is present at the industrial site and has access to the

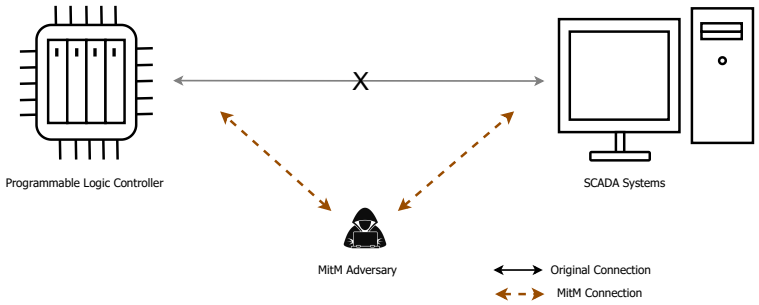


Figure 5.2: High level schematics of an MiTM adversary who eavesdrops the traffic via unauthorized network access to the *supervision* layer.

wired plant communication network at the supervision layer. The adversary can set up a physical device at any point in this network and has tools to interact with the ICS protocols and interpret the semantics of the ICS payloads. The ultimate goal of the adversary is to intercept the information exchanged between these devices and cause false data injection attacks or industrial espionage. Furthermore, we assume that the adversary does not have access to the control layer; this is admissible, as attack vectors with a lower relative likelihood (see Section 5.2.2) could be ignored [185].

5.3 Proprietary Products in Industrial Automation

In this section, we present a brief discussion on how EtherNet/IP protocol accesses the memory of the Logix family of controllers for performing read/write operations.

5.3.1 EtherNet/IP Message Format

EtherNet/IP stands for Industrial Ethernet and is considered one of the most widely deployed wired communication protocols for the industrial environment. Developed by Rockwell Automation, its adoption rate is the highest of all industrial ethernet protocols [7], serving critical infrastructures such as chemical processing, power generation, oil and gas, water distribution, etc. with a multitude of advantages over other industrial protocols [135]. EtherNet/IP is an application layer protocol and integrates CIP object libraries and device profiles [11] with standard TCP/IP and 803.2 technologies, allowing integration with the enterprise network.

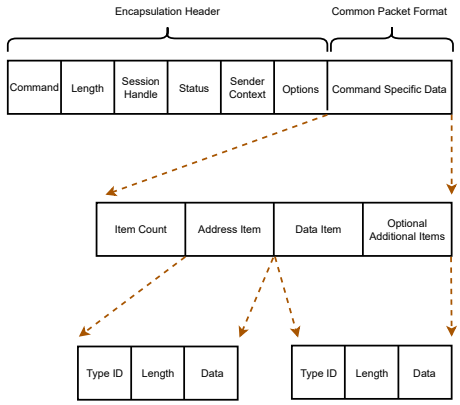


Figure 5.3: Encapsulation packet format in EtherNet/IP.

CIP follows an abstract object modelling, i.e. the network devices, such as PLCs and SCADA units, represent information in form of a series of objects and accessing their memory for performing read/write operations, is equivalent to accessing specific object instances for required attribute vales using defined services codes, as per the standard specification [102]. Each class of object has a defined set of attributes and services and the EtherNet/IP explicit messaging protocol mandates the use of the required Class ID, Instance ID, Attribute ID and Service Code to be mentioned explicitly in the encapsulation packet, while trying to access information in any connected network component.

The EtherNet/IP specification defines a number of objects [75] with the associated codes for their respective instances, attributes and services. Along with that, an integer ID is assigned to each of the network components in the Ethernet/IP network. For example, reading the vendor ID in a ControlLogix device (Allen Bradley PLC) using EtherNet/IP protocol, the ICS payload follows the standards defined in the specification: *Class ID* = `[0x01]`, *Instance ID* = `[0x01]`, *Attribute ID* = `[0x01]`, and *Service Code* = `[0x0E]`. Each Ethernet/IP device is modeled as a collection of objects with the standard addressing format *Class/Instance/Attribute/Service*, and the objects not found in the profile for a device class are vendor-specific objects.

EtherNet/IP Encapsulation Packet. EtherNet/IP uses a standard encapsulation packet format to send CIP packets over TCP/IP network and specifies reserved TCP and UDP ports to transfer data that should be supported by all the EtherNet/IP devices [10]. The encapsulation packet structure can be broadly divided into two parts: (a) Encapsulation Header, and (b) Command Specific Data. Figure 5.3 illustrates the

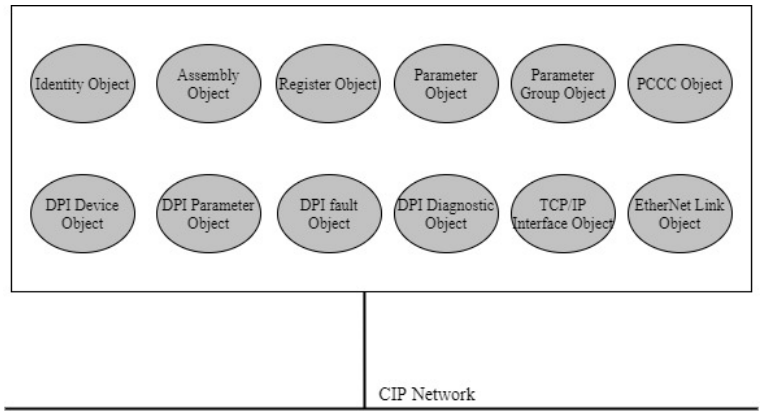


Figure 5.4: EtherNet/IP device modelled as a collection of objects

encapsulation packet format with a fixed-length header of 24 bytes that contains six fields specifying commands and other session-specific information required for the packet identification and transfer between two EtherNet/IP devices. We mainly focus on the command specific data portion that uses Common Packet Format to give a standard structure to the data exchanged between the two EtherNet/IP devices.

EtherNet/IP Messaging. EtherNet/IP supports two types of messaging for exchange of data between the network components. Implicit messaging refers to the time-critical control information that is exchanged at a specified periodic rate between the EtherNet/IP devices using the services of the UDP/IP/MAC protocol stack. With Explicit messaging, the client requests the server for some information which is explicitly stated in the message request using the TCP/IP/MAC protocol stack and the object model framework of the CIP standards. At the supervision layer of an ICS, EtherNet/IP follows explicit messaging standard, where the application layer PDU includes Class ID, Instance ID, Attribute ID, Service Code in the Command Specific Data of the encapsulation packet [11].

5.3.2 Protocol Address Space of Logix Controllers

It is important to note that vendors limit access to the PLC memory to its protocol address space and do not share the implementation details of the proprietary hardware architectures [167]. A typical ControlLogix controller organizes accessible memory in form of tags [158], and the client application interacts with the objects associated with the tags [18]. Tags are *application objects*, used for assigning and referencing memory in Allen Bradley controllers. For example, an object corresponding to the profile of a valve is essential for controlling the valve. A closer

look into the flagship products of Allen Bradley controllers shows that creating a tag creates an instance of Symbol Class in the controller memory, with a unique Instance ID assigned to it, with Rockwell-specific services such as Read Tag Service `[0x4c]` and Write Tag Service `[0x4d]` for performing read/write operations.

Table 5.1: Table Showing mapping of Tags and Instance IDs

Alias Tag	Base Tag	Instance ID
Low_Level_Sensor	<*Local:2:I.Data.1*>	0x0E
Liquid_A_Inlet	<*Local:2:I.Data.3*>	0x10
Mixture_Outlet	Internal Tag	0x16

To illustrate this, we code the operational semantics of the use-case in a ControlLogix (Logix 5571) using Ladder Diagram in Studio 5000. Table 5.1 shows the tag names and memory locations, along with the Instance IDs (extracted using pycomm3) of an input, output and internal tag.

5.4 Proposed Framework for Securing Communication at ICS Supervision Layer

In this section, we present a general description of SelEnc, a selective encryption framework that aims to provide purpose-built security in communicating confidential control/configuration information exchanged at the *supervision* layer of a control application.

System Architecture. SelEnc constitutes three distinct modules that provide a clear separation between functions, based on three layers of Open Systems Interconnection (OSI) model: a *filter module* that filters critical ICS payload, a *bypass module* that assigns a dynamic TCP port to the critical payloads, and a *security module* that performs encryption and decryption of critical payloads. We assume that, within a process, sensors are time-driven, and actuators, controllers, and SCADA systems are event-driven. Furthermore, we assume that the controller and SCADA systems are EtherNet/IP-enabled devices. Note that the scope could be extended towards other ICS products, with little modifications.

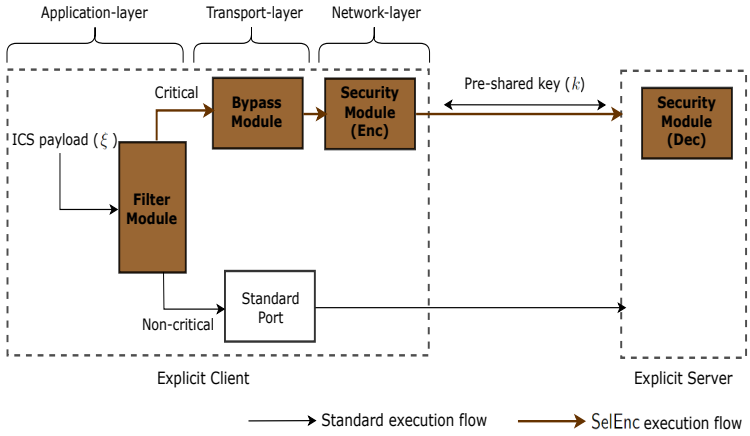


Figure 5.5: Schematics of SelEnc execution flow in the supervision layer.

5.4.1 Filter Module

This layer operates at the application layer and is responsible for filtering critical information to be exchanged at the supervision layer. Based on risk analysis, where threats range from basic, advanced to Advanced Persistent Threats (APTs), we claim that I/O values corresponding to the sensor and actuator signals could be safely established as *critical, as identifying or modifying these values could lead to industrial espionage* [216] or *false data injection* [159]. However, a process engineer can use his own discretion to identify any subset of I/O values as critical to the functionality of the application.

Let $\xi = (\xi_a, \xi_b, \xi_c)$ be an ICS payload, where ξ_a, ξ_b and ξ_c denote the Class ID, Instance ID, and Service Code of ξ. The filter module implements a function FilMod that takes as input ξ and a *filtering specification* $\mathcal{F}$ and outputs 1, if ξ agrees with the specification. The formal description of the function algorithm is given in Algorithm 5.1.

Definition 5.1 (Filtering Specification). *Let $\mathcal{X} = (x_1, \ldots, x_n)$ be the sequence of n I/O tags and $\mathcal{I} = (id_1, \ldots, id_n)$ be the corresponding Instance IDs of Class ID: q assigned to the tags. Let $\mathcal{Y} = (y_1, \ldots, y_m)$ be the sequence of m services defined on the class instances. Let $\mathcal{X}^c \subseteq \mathcal{X}$ be the set of critical tags defined for the target application. Let $\mathcal{Y}^c \subseteq \mathcal{Y}$ be the services required for performing filtering. Then, filtering specification is defined as $\mathcal{F}$ = (Class ID : q, Instance ID : $\mathcal{I}^c = \{id_i : x_i \in \mathcal{X}^c\}$, Service Code : $\mathcal{Y}^c$).*

We illustrate the above with a concrete setting: On implementing the operational semantics of the chemical mixing facility (see Section 5.2.1), all the I/O, internal values, etc. are assigned unique Instance IDs of Symbol Class `[0x6B]`. Let tags *Emergency_Stop* `[0x0F]` and *Liquid_A_Inlet* `[0x16]` be critical, and are filtered only when a *write* operation is performed on them, then $\mathcal{F}$ = (Class ID : `[0x6B]`, Instance ID : `[0x0F, 0x16]`, Service Code : `[0x4d]`). Note that, Service Code `[0xcc]` is the reply service for a Read Tag Service request `[0x4c]`, and `[0x4d]` is the Write Tag Service request, proprietary to Rockwell Automation.

Algorithm 5.1 FilMod (with embedded data $\mathcal{F}$)

Input: ξ
Output: 1 or 0
1: **if** $(\xi_a == q)$ AND $(\xi_b \in \mathcal{I}^c)$ AND $(\xi_c \in \mathcal{Y}^c)$ **then**
2: **return** 1
3: **else**
4: **return** 0
5: **end if**

5.4.2 Bypass Module

This module operates at the transport layer and bypasses the critical payload identified by the filter module through an ephemeral TCP port. Note that, EtherNet/IP uses the standard port `[44818]` to create a TCP connection. A dynamic port can be selected from the ICANN registry by the network engineer of the control application.

5.4.3 Security Module

The security module is responsible for encrypting critical payload and operates at the network layer. To minimize the time delay, we consider lightweight symmetric ciphers for implementing the module. Our key management paradigm is loosely based on the Industrial Key Infrastructure (IKI) concept considered in [14]. We assume the existence of a Trust Center, an authority completely controlled by the control infrastructure. The Trust Center configures a secret key in the PLC and SCADA devices using a secure network, where an adversary does not have access.

Execution Flow. The high-level schematics of the execution flow of SelEnc is given in Figure 5.5. Consider two EtherNet/IP devices, an explicit server $\mathcal{S}$ and an explicit client $\mathcal{C}$, sharing a secret key k. $\mathcal{C}$ executes filter module that triggers the

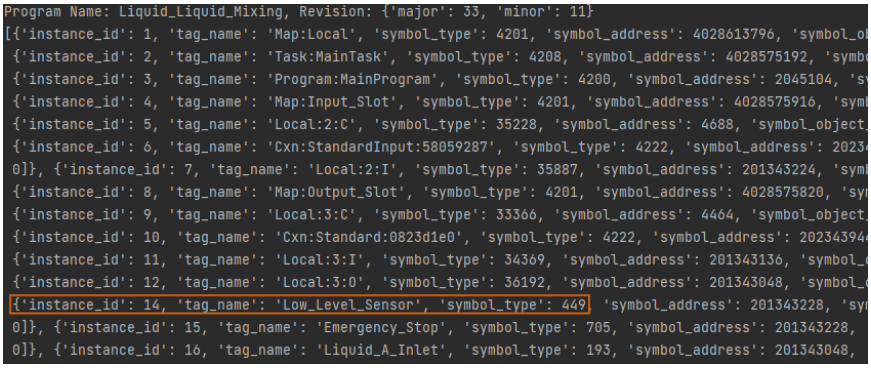


Figure 5.6: Screenshot showing the unique Instance IDs for I/O and internal tags: the highlighted tag *Low_Level_Sensor* has an Instance ID = 14.

execution of bypass module. If a critical payload is identified, the security module implements a function Enc that encrypts the critical payload at $\mathcal{C}$ and an inverse function Dec that decrypts the encrypted packets at $\mathcal{S}$.

Note that a control application with unacceptable delay is no longer time-invariant, and the *maximum permissible delay* is directly proportional to the response rate of the field devices [3]. A control engineer should calculate this value during the design phase to maintain system stability. Let $\Delta_{\mathcal{P}}$ be the maximum permissible delay of control system $\mathcal{P}$. Let $\tau_{\mathcal{P}}$ and $\delta_{\mathcal{P}}$ be the computation delay introduced due to the proposed framework at $\mathcal{C}$ and $\mathcal{S}$ respectively, then $\tau_{\mathcal{P}} + \delta_{\mathcal{P}} \leq \Delta_{\mathcal{P}}$ should hold in order to maintain the stability of $\mathcal{P}$. We ignore the delays due to physical components and electrical/electronic noises, along with communication delays.

5.5 Prototype Implementation

In this section, we present a proof-of-concept implementation of the proposed framework over our example testbed and an in-depth analysis of our empirical evaluation.

5.5.1 Experimentation Setup

We use an Allen-Bradley 1756 ControlLogix 5571 with a 2 MB processor, 32-slot digital I/O, and an EN2T ethernet connectivity module. We use a computer with an

Intel(R) Core(TM) i9-9900 processor, 32 GB RAM, and 64-bit operating system to design the Ladder Logic codes describing the operational semantics of the chemical mixing facility in Studio 5000, and download it in the controller using EtherNet/IP connectivity. We implement a SCADA monitoring/control unit using pylogix, perform read/write operations on the controller values, and capture the network traffic using Wireshark 3.4.6. We collect *five* real datasets from the network capture and design a simulation setup in a Raspberry Pi 3 Model B [169], with an ARMv7 rev 4 (v7l) processor and Raspbian GNU Linux 10 (buster) operating system for implementing the proposed framework using Python 3.7.3. We use two different lightweight symmetric ciphers, AES (block cipher) [67] and ChaCha-20 (stream cipher) [147] implementations from the PyCryptodome cryptographic library.

5.5.2 Methodologies

We study how ControlLogix assigns memory to the I/O tags and extract the unique Instance IDs of the Symbol class assigned to these tags (see Figure 5.6). We use th Raspberry Pi for implementing SelEnc, as access to the controller memory is made restrictive by the vendors, and thus the filter module at the application-level could not be implemented at a PLC. We define *two* use cases to evaluate the performance of our construction. The sets have a different number of critical tags. Case A considers all I/O tags as critical, while case B only takes two tags as critical (see Table 5.2).

Table 5.2: Use Cases

CASE A	**CASE B**
We observe *all* the I/O tags as critical, and implement the *Filter* module with $\mathcal{F}$ defined as follows: Class ID: `[0x6B]` Instance ID: `[0x6B, 0x0E, 0x0F, 0x11, 0x10, 0x12, 0x13, 0x16]` Service Code: `[0xcc, 0x4d]`	We observe *Emergency_Stop* and *Liquid_A_Inlet* tags as critical. $\mathcal{F}$ is defined as follows: Class ID: `[0x6B]` Instance ID: `[0x0F, 0x16]` Service Code: `[0xcc]`

To assess the scalability of our construction, we evaluate the performance of the proposed framework over five different datasets (see Table 5.3). Datasets 1 and 2

capture the packets exchanged while performing a single read/write operation on a critical tag. Dataset 3 constitutes the network traffic generated for reading all the tags (corresponding to the program name, task name, internal variables, I/O values, etc.) Dataset 4 captures the packets while reading all the I/O tags for 30 seconds. Finally, dataset 5 contains the packets generated while exchanging both configuration and control information.

We determine the total number of CIP packets in the captured traffic and the number of CIP packets filtered (using the criteria defined in Case A and Case B). We measure the delay introduced due to the proposed framework (this includes the time required for performing filtering, bypassing, encryption and decryption). We benchmark the performance of AES and ChaCha-20 when implemented individually in SelEnc, as these are lightweight ciphers well-suited for constrained control environments. Furthermore, we encrypt the entire network traffic and compare the overhead with our approach of encrypting filtered traffic. Finally, we evaluate the overhead introduced due to SelEnc over the unencrypted traffic.

Table 5.3: Experimental Datasets

Dataset	**Description**
1	Read *Low_Level_Sensor* tag
2	Write *Liquid_A_Inlet* tag
3	Read all tags
4	Read all I/O tags for 30 seconds
5	Download configuration data and read *all* tags for 30 seconds

5.5.3 Results

Table 5.4 depicts the performance of our construction over the captured datasets in relation to Case A, where the filtering specification specifies *read/write operation on all the I/O tags*.

As clear from Table 5.4, only 1.2% of the entire traffic is encrypted for reading/writing a single I/O tag. Dataset 3 reads all tags (including the I/O tags) from the PLC memory, while the *filter* module considers only 6.66% of the traffic. Reading only the I/O tags for 30 seconds requires only 33.3% of the traffic to be filtered. Table 5.5 shows the summary of results for Case B, where $\mathcal{F}$ describes a *read operation on two I/O tags*. For datasets 4 and 5, an average of 5.9% of the entire traffic corresponds to reading the two specified I/O tags.

Table 5.4: Experimental Results (Case A)

Dataset	Packets			Time (in ms.)		
	CIP	Filtered	Total	Enc.		Unenc.
				AES	ChaCha-20	
1	20	1	79	1.8	1.7	0.5
2	19	1	81	2.2	1.8	0.4
3	38	7	105	5.8	4.7	0.6
4	18765	9373	28194	3778	2614	2342

Table 5.5: Experimental Results (Case B)

Dataset	Packets			Time (in sec.)		
	CIP	Filtered	Total	Enc.		Unenc.
				AES	ChaCha-20	
4	18765	1562	28194	0.99	0.79	0.234
5	24605	2314	38295	1.46	1.16	0.31

5.5.4 Discussions

An interesting observation to make is that ChaCha-20 outperforms AES, introducing less than 44% (Case A) and 20% (Case B) delay. Existing bump-in-the-wire techniques mainly rely upon AES, however using ChaCha-20 for our implementation was inspired due to its simpler design and speed in hardware. Also, the proposed SelEnc framework reflects an average decrease in the delay of 79% and 84% over encrypting the *entire* traffic, which is followed in TLS and existing bump-in-the-wire approaches. Figure 5.7 shows the performance benchmark of the proposed framework over encrypting entire traffic using AES and ChaCha-20 implementations independently. As clear from the figure, with only two I/O tags as critical (Case B), SelEnc introduces 26% decrease in delay over Case A.

5.6 Conclusion

In this chapter, we have discussed the significance of application-level security implementations in industrial control applications and the downsides of the existing approaches in providing the same. Following this, we have introduced a framework that achieves such security in communicating confidential information with significantly lower overhead compared to the current approaches.

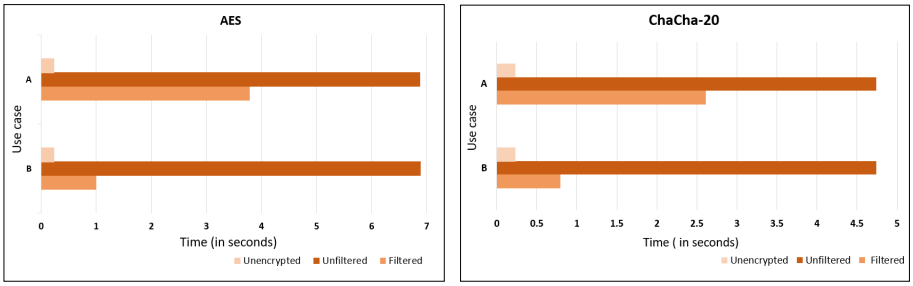


Figure 5.7: Performance benchmark of SelEnc against encrypting entire traffic (for dataset 4) using independent implementations of AES and ChaCha-20.

Our design helps envision how minimal controller-access would allow customers to define their respective benchmarks of security. Our solution can be *easily extended to provide customized integrity and authentication in ICS communication.* As our future work, we would examine other ICS products and protocols, and analyze customer requirements based on user stories such that a standard paradigm of application-specific security for ICS platforms could be designed.

Chapter 6

Privacy-Preserving Classification Using Cryptographic Obfuscation

Based on our discussion in Chapter 1, one of the essential pre-requisites in delivering targeted attacks against industrial infrastructures is reconnaissance, where an adversary learns the operational semantics of the process control framework to develop custom zero-day exploits. We consider a MATE adversary, who retrieves the software implementation of a control program. Our objective is to prevent the adversarial attempts to reverse engineer the recovered implementation of the control program by employing cryptographic obfuscation. This necessitates formalizing control programs and defining its *assets*, the secret values in the program that give away the process semantics.

This chapter focuses on *binary decision trees*, the data structure we employ to formalize control programs in Chapter 7. We give a formal definition of binary decision trees along with their assets. Thereafter, we design an efficient VBB obfuscator for a special family of binary decision trees and use the random oracle model to analyze the security of our construction. Furthermore, we design an encoder for hiding parameters in an *interval membership function*, which we use as the primary building block for developing the proposed VBB obfuscator. We note that, our contribution is of independent interest and has wider applicability in software program obfuscation. *Our exclusive goal behind designing the obfuscator is that, not only will the solution increase the class of functions that has cryptographically*

secure obfuscators, but also address the open problem of non-interactive prediction in privacy-preserving classification using computationally inexpensive cryptographic hash functions, along with preventing adversarial attempts of extracting operational semantics of target control applications, the last of which is the ultimate aim of this work.

6.1 Rationale

The focus of this chapter is in presenting a new technique for *encoding interval membership functions*. We find an interesting and important application in designing an efficient *virtual black-box obfuscator* for *evasive decision trees* (see Definition 6.4.1). In the following, we present our intuition behind obfuscating decision trees.

6.1.1 Privacy-Preserving Classification with Decision Trees

In the interest of establishing the usefulness and significance of obfuscating decision trees, we provide a brief overview on privacy-preserving classification using decision tree classifiers.

Decision tree classifiers are extensively used for prediction and analysis in sensitive applications such as spam detection, medical or genomics, stock investment, etc. [183, 32, 70, 176]. *Consider an example of a medical facility (model-provider) who designs a model from sensitive profiles of patients to diagnose a certain disease. The model is then outsourced to a cloud server to provide classification to a user who wants to make a prediction about her health. If the model is leaked, the sensitive training data will be disclosed [12, 83], breaching the HIPAA*[1] *compliance. What's more, the user does not want to reveal her queries and classification results to the cloud server. This calls for privacy-preserving classification techniques, where the model is hidden from anyone but the model-provider and prediction queries/classification remain private to the user, such that no leakage of useful information happens during the classification phase.*

Related Work on Privacy-Preserving Classification. The existing solutions for safeguarding model and user queries/classification usually follow a paradigm, where the model is encrypted and outsourced to a server, where it processes the encrypted input and forwards encrypted classification to the user [225, 122].

[1] Health Insurance Portability and Accountability Act of 1996

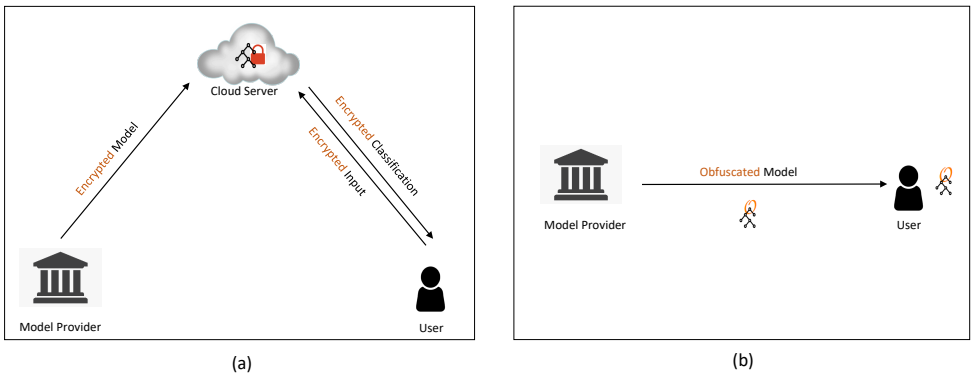


Figure 6.1: Privacy-preserving classification with Decision Tree. (a) Interactive protocol with encrypted model outsourced to cloud server. (b) *Non-interactive protocol with obfuscated model sent directly to the user.*

The state-of-art privacy-preserving classification solutions employ an *interactive* approach: encrypt and outsource the model to cloud server, where it processes encrypted queries and forwards encrypted classification to the users. The constructions involve multiple rounds of communication and rely upon expensive cryptographic computations using fully-homomorphic encryption (FHE) ([34, 35]), garbled circuits ([26, 25]), etc.

Brickell *et al.* [37] suggest an interactive two-party protocol employing additive homomorphic encryption and ml oblivious transfers (where l is the bit-length of each input feature and m is the number of decision nodes), restricting the user from performing multiple queries on the encrypted tree. In their well-known work, Bost *et al.* [35] present a comparison protocol between model-provider and the user for each node in the decision tree using FHE method.

Tai *et al.* [188] make use of multiple communication rounds to transfer the path costs and encrypted labels to the client. Tueno *et al.* in [195] claim designing a non-interactive FHE/SHE based proposal, where entire evaluation takes place at the server. However, we believe that the protocol is interactive as the user needs to communicate with the server to submit encrypted query and receive encrypted classification.

In [118], Liang *et al.* design a method for efficient privacy-preserving decision tree classification by transforming a classifier to a Boolean vector, then using symmetric ciphers, PRFs and pseudo-random permutations to encrypt the vector and generate encrypted tokens. Finally evaluation is an interactive protocol with overall communication cost in the order of $\mathrm{O}\big(m(s + s' + n)\big)$, where m, s, s' and n are the number of leaf noes, pseudo-random functions, pseudo-random permutations and internal nodes respectively.

In order to reduce the high computation and communication costs introduced by the FHE techniques, the authors of [60] design a solution (SortingHat) that secures the prediction queries and classification results, but does not guarantee the privacy of the model. They propose a homomorphic comparison algorithm that takes encrypted prediction queries as input and compares them with the model in plaintext, claiming the practicability of their design in terms of the overheads incurred. However taking into account the potential attack vectors, we argue that deploying such a design is infeasible.

Our motivation behind obfuscating decision trees is to eliminate interaction between user and model-provider/cloud server. In particular, we aim to construct an efficient non-interactive solution to privacy-preserving classification with evasive decision trees. We now explain why we do not consider obfuscating arbitrary decision trees. If a decision tree could be learned from the input-output behaviour of the model, then protecting the privacy of the model would be impossible. Note that, learning a decision tree means identifying the decision nodes and input attributes associated with them, and identifying the accepting nodes. Tramer *et al.* [193] show that a decision tree could be learned through $m \cdot \log_2(b/\epsilon)$ oracle queries, where m is the number of internal nodes, b is the minimum width of an interval in a node, and ϵ is the specified precision value; they call it *model extraction attack*. To prevent such attacks, the existing literature observes API calls to issue warnings [110, 166] or adds perturbations [117, 224]. However, since there are no theoretical restrictions on the number of prediction queries made by a user [152], limiting them is not reasonable approach towards thwarting such attacks. We define a special class of decision trees, for which it is hard to find an accepting input, such that an efficient algorithm cannot extract the model except with negligible probability; we call such decision trees as *evasive*, and claim that if a decision tree is not evasive, then it is impossible to protect the privacy of the model, and hence there is no choice but to restrict to evasive decision trees.

Lockable obfuscation (also called compute-and-compare obfuscation) [88, 202] is a very general tool that encodes a class of branching programs under the learning-with-errors (LWE) assumption. It could be employed to build a decision tree obfuscator, by writing the decision tree as a circuit. Nevertheless, we focus on solutions that are simpler and potentially more practical. In [36], the authors initiate a theoretical investigation on decision tree obfuscation based on indistinguishability obfuscation (see Definition 2.5.4) which is the 'best-possible' obfuscation from the point of view of VBB, but does not guarantee the privacy of the decision tree. We aim to achieve stronger notions of security that allow us to protect the privacy of the model.

6.2 Our Contributions

On the whole, we contribute towards designing a new technique for encoding interval membership functions, and as an application we construct an efficient VBB obfuscator for *evasive decision trees* (see Definition 6.4.4). We focus on trees of bounded number of inputs and depth, but the techniques will apply to more general classes of decision trees. Note that, we do not consider privacy-preserving methods to construct the model and how the model is obtained is out of the scope of this study. A technical briefing of our construction is as follows:

Technical Overview. We consider decision trees that perform binary classification based on the values of n attributes. Attributes are represented as ℓ-bit strings x_i, and are interpreted as integers in $[0, 2^\ell)$. A decision tree is a full binary tree of depth d. Internal node v_j (also called *decision* node) associates threshold t_j, where t_j is an integer between 0 and $2^\ell - 1$. Each decision node tests $x_i \leq t_j$ for some i. The leaf nodes $(s_1, \ldots, s_{2^d})$ are labelled 0 (reject) or 1 (accept). Hence the decision tree is represented by the pairs $[\![t_j, i]\!]$, and the labels on the leaf nodes.

Without loss of generality we may assume that, for any specific path from the root to an accepting leaf, x_i is compared at most twice. Hence each accepting path corresponds to a sequence of interval membership predicates $x_i \in (c_i, c_i + w_i]$. The key observation is that membership $x_i \in (c_i, c_i + w_i]$ can be expressed as a union of distinct predicates $x_i \in [a, a + 2^p)$ for certain pairs (a, p). Each such predicate can be turned into a point function predicate and hence be obfuscated using hashing. We explain the details in the next paragraph.

Let $f_i : \{0,1\}^\ell \to \{0,1\}^{\ell-i}$ such that $f_i(y) = \lfloor \frac{y}{2^i} \rfloor$ for $i \in \{0,1,\ldots,\ell-1\}$. Calculate intersection of sub-intervals $\mathcal{I}^i$ corresponding to $(c_i, c_i+w_i]$ (of the form $[a, a+2^p)$). Encode each entry in $\mathcal{I}^i$ using $H(f_p(a))$, where $H : \{0,1\}^* \to \{0,1\}^\omega$ is a cryptographic hash function; call the set $\mathcal{B}^i$. Note that, this method converts interval membership predicate $x_i \in [a, a+2^p)$ into a point function predicate that determines whether $H(f_p(a))$ is equal to $H(f_q(x_i))$ for some $q \in \{0,\ldots,\ell-1\}$. Finally, for each encoding in $\mathcal{B}^i$, concatenate n entries sorted in the order of i, apply cryptographic hash function $H_c : \{0,1\}^* \to \{0,1\}^q$, and publish the set of hashes. Reordering the nodes in the order of i along each accepting path hides the structure, though the size of the obfuscated program may reveal the number of different accepting paths. To classify input $(x_i)_{i\in[n]}$, compute the set of encodings $\mathcal{E}^i$ by calculating $H(f_q(x_i))$, where $q \in \{0,\ldots,\ell-1\}$ and for each encoding, concatenate n entries sorted in order of i, and apply H_c. For an accepting input, one of the hashes computed by the evaluation procedure will be contained in the set of hashes published by the obfuscator.

Organization. This chapter is organized into the following sections. Section 6.3 briefly discusses the preliminaries. Section 6.4 introduces the formal definitions of decision trees along with conditions for evasiveness. Section 6.5 gives a description of the proposed construction. Section 6.6 provides the proof for VBB security of our proposal. An overall conclusion is presented in Section 6.7.

6.3 Preliminaries

Below are the standard notations and terminologies that will be used throughout the chapter.

6.3.1 Notation

Let S be a set defined by an integer interval as $S = \{x \in \mathbb{N} : 1 \leq x \leq n\}$. We denote by $|S|$, size of the set S. We use the standard notations to denote intervals as (a,b), $(a,b]$, $[a,b)$ and $[a,b]$, for $a, b \in \mathbb{N}$. We denote by $\lfloor x \rfloor$ the integral part of x, where $x \in \mathbb{R}$. We use $\log_2(n)$ to denote the power to which 2 should be raised to obtain the value $n \in \mathbb{N}$.

We denote l-bit *binary encoding* of n as $r_{\ell-1}.2^{\ell-1} + \cdots + r_0.2^0$ for $n \in \mathbb{N}^+$, where $r_i \in \{0,1\}$. We denote hamming weight of n as $wt(n) = \sum_{i=0}^{\ell-1} r_i$. For a

program C, we denote its size by $|C|$. We rely upon the notion of *computational security* and follow the *asymptotic approach* throughout the paper. We provide the honest parties and the adversaries with a security parameter $\lambda \in \mathbb{N}$. We model the adversaries as a family of probabilistic polynomial time (PPT) programs, running in time $a.\lambda^c$, for some constants a, c. A function $\mu : \mathbb{N} \to \mathbb{R}^+$ is called *negligible* in n, if is grows slower than n^{-c}, for every constant c. We measure negligibility with respect to the security parameter λ. We use $\|_{i=1}^{n}\, a_i$ to denote concatenation of a sequence of strings $(a_1, \ldots, a_n)$.

6.3.2 Standard Definitions

We state some standard definitions that will be used for the construction of our obfuscator.

Definition 6.3.1 (Integer Intervals)**.** *Let $a, b \in \mathbb{N}$. A closed integer interval $[a, b]$ is defined by the set $S = \{x \in \mathbb{N} : a \leq x \leq b\}$, whereas an open integer interval (a, b) refers to $S = \{x \in \mathbb{N} : a + 1 \leq x \leq b - 1\}$, where a and b denote lower and upper bounds of the intervals respectively. Width of interval, denoted by $w(S)$ is the cardinal number of the set S defined by the interval.*

Definition 6.3.2 (Interval Membership Function)**.** *Let S be a set defined by an integer interval. An interval membership function $\Pi_S : \mathbb{N} \to \{0, 1\}$ is defined by $\Pi_S(x) = 1$ if $x \in S$ and 0 otherwise.*

Cryptographic Hash Functions. Hash functions are deterministic functions that *compresses* binary strings of arbitrary length and produces fixed-length strings (*digests*), with a property that it is computationally hard to invert the function and retrieve the input strings from the digests.

Definition 6.3.3 (Hash Functions)**.** *A pair of polynomial-time algorithms (key, H) that satisfies the following requirements:*

- key *is a probabilistic algorithm that outputs a key s on a security parameter $\lambda \in \mathbb{N}$.*
- *There exists a polynomial ℓ, such that H is a deterministic polynomial-time algorithm that on input s and $x \in \{0, 1\}^*$, produces a string $H^s(x) \in \{0, 1\}^{\ell(\lambda)}$.*

We are interested in *collision-resistant* hash functions, with a property that it is computationally hard to find a collision in the function H^s on a pair of distinct inputs $x, x' \in \{0, 1\}^*$, where collision happens when $H^s(x) = H^s(x')$.

Definition 6.3.4 (Collision-Resistant Hash Functions). *A hash function (Key, H) is collision-resistant, if for all polynomial-time adversaries $\mathcal{A}$ with $s \leftarrow Key(1^\lambda)$, all $(x, x') \leftarrow \mathcal{A}$ such that $x \neq x'$, there exists a negligible function μ, such that*

$$Pr[H^s(x) = H^s(x')] \leq \mu(\lambda)$$

6.4 Formalizing Decision Trees

In this section, we formalize binary decision trees. Without loss of generality, we define decision trees to be full binary trees and restrict to binary classification. Following this, we introduce *evasive decision trees* and what it means to *learn* a decision tree.

A decision tree classifier exploits the tree structure to build a model where the internal nodes of the tree represent the *decision* nodes, that compares the input set of attributes against some conditions, the edges represent the outcomes of the comparison and finally the leaf nodes represent the final *decision* or *classification* corresponding to the set of input attributes.

Definition 6.4.1 (Decision Trees). *Let $n, d, \ell \in \mathbb{N}$ and $(x_i)_{i=1}^n = (x_1, \ldots x_n) \in \mathbb{N}^n$ be a finite sequence of input elements, where x_i is an integer between 0 and $2^\ell - 1$ that represents the value of some attribute.*

A decision tree is a representation of a function $C : [0, 2^\ell)^n \to \{0, 1\}$. It is a full binary tree of depth d with internal nodes $(v_1, \ldots, v_{2^d-1})$ (where v_1 is the root, v_2, v_3 are nodes at the second level, and so on) and leaf nodes $(s_1, \ldots, s_{2^d})$. Leaf node $s_k \in \{0, 1\}$ gives the value of the function C. Internal nodes v_j are labelled by a pair $[\![t_j, i]\!]$ that defines a predicate g_j as $g_j(x_i) = 1$ if and only if $x_i \leq t_j$. To evaluate the tree on an input $(x_1, \ldots, x_n)$, one follows a path from the root to a leaf, by taking the left child if $g_j = 0$ and the right child if $g_j = 1$. The output is the value of the leaf s_k at the end point of this walk in the tree.

To define model extraction resistance we define the *assets* of a decision tree. Note that a decision tree does not necessarily have a unique representation.

Definition 6.4.2 (Asset of Decision Tree). *Let C be the function represented by a decision tree. We define $\mathsf{asset}(C)$ to be a sequence of pairs $[\![t_j, i]\!]$ and a sequence of leaf nodes $(s_1, \ldots, s_{2^d})$ such that the corresponding decision tree from Definition 6.4.1 defines the same function C.*

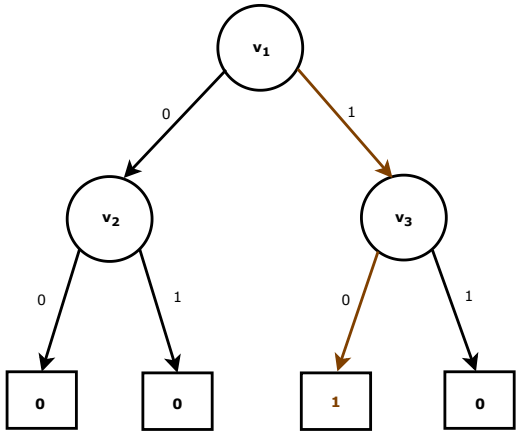


Figure 6.2: Binary classification with a decision tree: the circular nodes represent decision nodes, and the square nodes represent terminal nodes. Decision nodes are numbered in level-order sequence. The path in orange represents the accepting path with terminal node labeled 1.

Without loss of generality, we assume x_i to be compared at most twice along an accepting path. Let $w_{max} \in \mathbb{N}$ and c_i, $c_i + w_i$ be integers between 0 and $2^\ell - 1$ and $w_i \in (0, w_{max}]$. Consider $x_i \leq c_i + w_i$ and $x_i > c_i$ along a path from root at level 0 till terminal nodes at level $d - 1$. The collection of inequalities define an interval $(c_i, c_i + w_i]$, where x_i is true. Finally, $C((x_i)_{i\in[n]}) = 1$ if $x_i \in (c_i, c_i + w_i]$ for every $i \in [n]$. We stress that different accepting paths may arise from different comparisons of x_i. As we now explain, for evasiveness we need the w_i to be not too large, so we introduce an upper bound $w_{max} \in \mathbb{N}$.

Definition 6.4.3 (Decision Region)**.** *Let n, ℓ, $w_{max} \in \mathbb{N}$. Let c_i, $c_i + w_i$ be integers between 0 and $2^\ell - 1$ and $w_i \in (0, w_{max}]$. Define a decision region as the hyper-rectangle formed by n intervals $(c_i, c_i + w_i]$.*

Consider a classification function defined as:

$$C(x_1, x_2) = \begin{cases} 1, & \text{if } x_1 \in (a_1, b_1] \text{ and } x_2 \in (a_2, b_2] \\ 0, & \text{otherwise} \end{cases} \tag{6.1}$$

where $x_1, x_2 \in [0, M]$ for some large integer M.

Figure 6.3 indicates the decision tree that specifies the classification function in Equation 6.1. The decision region h is a rectangular region in a two-dimensional space.

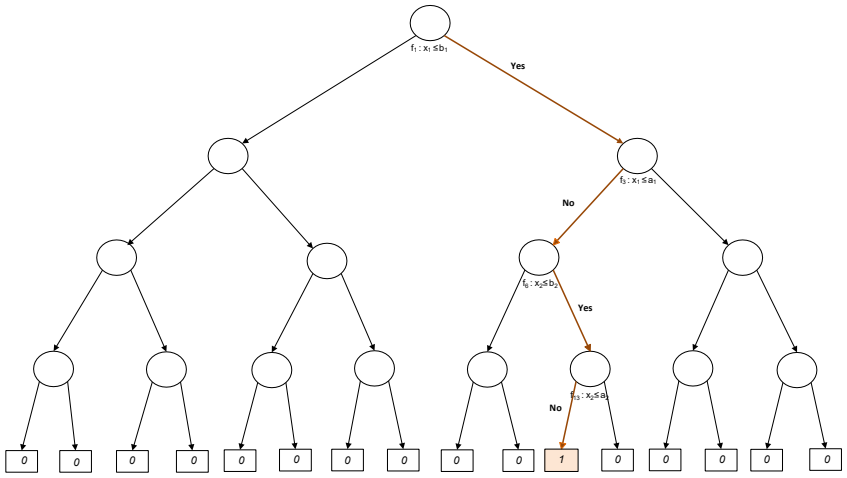


Figure 6.3: Decision tree model specifying the classification function C

Evasive Function Family. As stated in Definition 6.4.1, a binary classification program maps input $(x_i)_{i\in[n]}$ to one of the classes $\{0, 1\}$. From [193], it is evident that an adversary, by identifying the accepting/rejecting inputs, can extract the model (learn the assets in the classification program using binary search) within polynomial attempts. From our discussion in Section 6.1, we can conjecture that it is impossible to protect the privacy of the model from generic model-extraction attacks, and this is our intuition behind restricting to a special class of decision trees (for which it is computationally hard to find an accepting input). We call this *evasive* collection which states that for every input, a random program selected from this collection evaluates to 0 with overwhelming probability. In what follows, we define evasive decision tree collection, with the exclusive goal of obfuscating this class of programs, such that adversary cannot learn the assets from the input/output behavior of the programs. Throughout the paper, we assume that an adversary knows the domain of inputs, but not the accepting inputs.

Definition 6.4.4 (Evasive Decision Tree Distribution)**.** *Let $\mathcal{D} = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ be a distribution of polynomial-size classification functions represented as decision trees of depth $d(\lambda)$ on $n(\lambda)$ variables. We say $\mathcal{D}$ is evasive, if there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$, for every input $(x_i)_{i\in[n(\lambda)]}$*

$$\Pr_{C\leftarrow\mathcal{D}_\lambda}[\, C((x_i)_{i\in[n(\lambda)]}) = 1] \leq \mu(\lambda)$$

In short, Definition 6.4.4 requires that for every $(x_i)_{i\in[n]}$, a program C chosen randomly from the distribution evaluates to 1 with negligible probability.

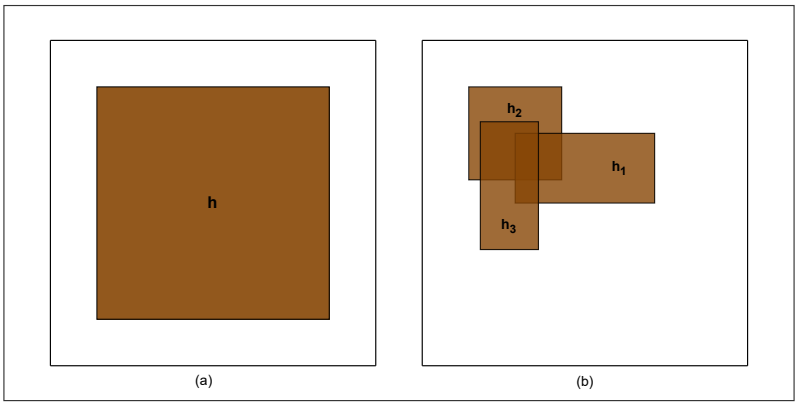


Figure 6.4: Two example cases of distributions which lead towards non-evasiveness: (a) Decision region h_1 is very big. (b) Overlapping decision regions h_1, h_2 and h_3.

A distribution $X_n \in [0, 2^\ell)^n$ defines a distribution $\mathcal{D}_\lambda$, such that $C \leftarrow \mathcal{D}_\lambda$ computes whether an input $(x_i)_{i\in[n]}$ is accepted or not as follows: sample $(c_1, \ldots, c_n) \leftarrow X_n$ and $(w_1, \ldots, w_n) \leftarrow (0, w_{max}]^n$. The accepted inputs satisfy $x_i \in (c_i, c_i + w_i]$ for every $i \in [n]$. For the program collection to be evasive, it is necessary that, for fixed $(x_1, \ldots, x_n)$, the probability is negligible that $x_i \in (c_i, c_i + w_i]$ for every i. Thus we require X_n to have large entropy. As uniform distributions provide the highest entropy, this is the best case. However, real-world applications may have accepting regions that are less uniform. The scenarios that lead to non-evasiveness are: (1) the decision regions are too big (so a random x is likely to be in the set); (2) the number of points in the space $[0, 2^l)^n$ representing $(c_1, \ldots, c_n)$ is too few (not enough entropy); (3) the decision regions overlap each other (so one can choose x from the intersection of the regions). Figure 6.4 shows two example distributions that give non-evasive decision trees.

Hence for an evasive collection, the distribution X_n needs to have a large number of points representing $(c_1, \ldots, c_n)$. Adding to that, the distribution needs to be 'well-spread' (meaning that a set of randomly chosen accepting regions should have empty intersection). We further this discussion with the calculation of the required parameters for identifying an evasive program collection. We start with uniform distribution on $[0, 2^l)^n$, where n, ℓ are polynomials in λ.

Lemma 6.4.1. *Let $n, \ell \in \mathbb{N}$. Let c_i is integer between 0 and $2^\ell - 1$ and w_i is an integer between 0 and w_{max}. The number of elements in the decision region $(c_1, c_1 + w_1] \times \cdots \times (c_n, c_n + w_n]$ is at most $(w_{max})^n$.*

Proof. For a c_i selected uniformly in $[0, 2^\ell)$, w_i has to be selected such that $w_i < w_{max}$, for some $w_{max} \in \mathbb{N}$. It can be readily seen that the number of ways of selecting the elements along an interval is w_{max}. For all the n intervals chosen uniformly and independently of each other, the number of possible ways is at most $(w_{max})^n$. □

Lemma 6.4.2. *Let $\lambda \in \mathbb{N}$ be the security parameter and $n, \ell, w_{max} \in \mathbb{N}$, where $w_{max} \leq 2^{(\ell - \frac{\lambda}{n})}$. Fix an input $(x_i)_{i\in[n]}$. Choose uniformly $c_i \in [0, 2^\ell)$ and $w_i \in (0, w_{max}]$. Then the probability that $(x_i)_{i\in[n]}$ belongs to the decision region defined by $(c_i, c_i + w_i]$, for $i \in [n]$, is not more than $2^{-\lambda}$.*

Proof. The total number of points in the space $[0, 2^\ell)^n$ is given by $2^{\ell n}$. The input $(x_i)_{i\in[n]}$ is contained in the decision region defined by $(c_i, c_i + w_i]$, $i \in [n]$, when $x_i \in (c_i, c_i + w_i]$ for every $i \in [n]$, where c_i and w_i are chosen uniformly from $[0, 2^\ell)$ and $[0, w_{max})$ respectively.

Now, for a fixed input $(x_1, \ldots, x_n)$ to be contained in the decision region, the c_i's need to be selected such that $c_i \in (x_i - w_i, x_i]$, for every $i \in [n]$. It can be readily seen from Lemma 6.4.1 that the number of ways of selecting $(c_1, \ldots, c_n)$ such that the w_i's are less than w_{max} are $(w_{max})^n$, and thus the probability that $(x_i)_{i\in[n]}$ belongs to the decision region defined by $(c_i, c_i + w_i)$ is given by $\frac{(w_{max})^n}{2^{\ell n}}$. For $w_{max} \leq 2^{(\ell - \frac{\lambda}{n})}$, the above probability is at most $2^{-\lambda}$ (negligible function in λ). □

The result shows that if the intervals $(c_i, c_i + w_i]$'s are uniformly chosen with $w_i \leq 2^{(\ell - \frac{\lambda}{n})}$, then the probability that an input (fixed a priori) belongs to the decision region is negligible in λ. We now prove that the class of decision tree functions defined by uniform distributions that follow the above mentioned parameter restrictions, forms an evasive program collection.

Lemma 6.4.3. *Let λ be the security parameter and ℓ, n be polynomials in λ. Let $w_{max} = w_{max}(\lambda)$ be some function such that $w_{max}(\lambda) \leq 2^{\ell(\lambda) - \frac{\lambda}{n(\lambda)}}$. Let $2^{-\lambda}$ be a negligible function. Let X_n be an uniform distribution on $[0, 2^\ell)^n$ and let $\mathcal{D}_\lambda$ be the corresponding distribution on decision trees that checks if $(x_i)_{i\in[n]}$ belongs to the decision region defined by the c_i's and w_i's. Then $\mathcal{D}_\lambda$ is an evasive program distribution.*

Proof. A uniform distribution on $[0, 2^\ell)^n$ defines a $\mathcal{D}_\lambda$, and we need to show that for every $\lambda \in \mathbb{N}$ and every $(x_i)_{i\in[n]}$, $\Pr_{C\leftarrow\mathcal{D}_\lambda}[C((x_i)_{i\in[n]}) = 1] \leq \mu(\lambda)$. For $C \leftarrow \mathcal{D}_\lambda$, the probability that $(x_i)_{i\in[n]}$ is accepted by C is equal to the probability that (x_i) lies in the product of uniformly chosen intervals $(c_i, c_i + w_i]$ as above. It is evident from Lemma 6.4.2 that this probability is at most $2^{-\lambda}$. □

Definition 6.4.5 (Computational Indistinguishabiliy)**.** *Let $\mathcal{X} = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{Y} = \{Y_\lambda\}_{\lambda\in\mathbb{N}}$ be two probability ensembles. We say they are computationally indistinguishable, denoted by $\mathcal{X} \cong \mathcal{Y}$, if for every non-uniform PPT algorithm $\mathcal{A}$, there exists a negligible function $\mu(\lambda)$, such that the following holds:*

$$|\Pr_{x\leftarrow X_\lambda}[\mathcal{A}(x) = 1] - \Pr_{x\leftarrow Y_\lambda}[\mathcal{A}(x) = 1]| \leq \mu(\lambda)$$

Next we explore general distributions. As discussed beforehand, we do not want the support of the distribution to have 'few' points and the decision regions to be clustered along the distribution (clumped distribution) so that we can eliminate the possibilities of non-evasiveness. We require a distribution to have 'sufficient' randomness. An important metric for quantifying the randomness of a distribution is *min-entropy*, which measures the difficulty of correctly guessing a sample from a given distribution.

Definition 6.4.6 (Min-entropy)**.** *A random variable X has a min-entropy, defined by $\mathcal{H}_\infty(X) = -\log[\max_x Pr[X = x]]$ and an average (conditional) min-entropy on a (possibly) correlated random variable Y defined by $\mathcal{H}_\infty(X|Y) = -\log(\mathbb{E}_{y\leftarrow Y}[\max_x Pr[X = x|Y = y]]$.*

Definition 6.4.7 (Decision Tree Min-entropy)**.** *Let ℓ, n are polynomials in the security parameter $\lambda \in \mathbb{N}$. Let $w_{max} \leq 2^{(\ell-\frac{\lambda}{n})}$. Let X be a random variable on $[0, 2^\ell)^n$. Then the decision tree min-entropy of X is defined as:*

$$\mathcal{H}_{D,\infty}(X) = -\log[\max_{(x_i)_{i=1}^n\in\mathbb{N}^n} Pr[X \in \prod_{i=1}^{n}(x_i - w_{max}, x_i]]$$

Lemma 6.4.4. *Let $\lambda \in \mathbb{N}$ be the security parameter and let $\mathcal{D} = D_\lambda$ be an ensemble of distributions over $[0, 2^{\ell(\lambda)})^{n(\lambda)}$. For $w_{max}(\lambda) \leq 2^{\ell(\lambda)-\frac{\lambda}{n(\lambda)}}$, $\mathcal{D}_\lambda$ is decision tree evasive distribution if min-entropy of $\mathcal{D}_\lambda$ is at least λ.*

We next discuss what it means to *learn* an evasive decision tree. If a decision tree is unlearnable, then there is no model extraction attack.

Definition 6.4.8 (Unlearnable Decision Trees)**.** *A collection of classification functions $\mathcal{C}$ is unlearnable, if for every polynomial time algorithm $\mathcal{A}$ with oracle access to C, there exists a negligible function μ, such that for every $\lambda \in \mathbb{N}$:*

$$\Pr_{C \leftarrow \mathcal{D}_\lambda}[\mathcal{A}^C(1^\lambda) = \mathsf{asset}(C)] \leq \mu(\lambda)$$

Remark 6.4.1. *Note that evasiveness implies unlearnability, because evaluating an evasive function always returns 0 with overwhelming probability and hence no information about the function is provided by these queries.*

6.5 Obfuscating Evasive Decision Trees

In this section, we introduce a *new technique for encoding interval membership functions*. We follow this with a description of our decision tree obfuscator.

6.5.1 Setup.

Without loss of generality, we assume decision trees to be full binary trees that perform binary classification on an input $(x_i)_{i\in[n]}$, where $x_i \in \{0,1\}^\ell$. We consider a decision tree function $C \in \mathcal{D}_\lambda$ with a depth d. We denote decision nodes by $(v_1, \ldots, v_{2^d-1})$ and terminal nodes by $\mathcal{S} = (s_1, \ldots, s_{2^d})$. An *accepting path* P_{s_τ} is defined as the sequence of tuples $[\![t_j, i, b]\!]$, such that v_j is an ancestor node of $s_\tau \in \mathcal{S}$ with $s_\tau = 1$, and $b \in \{0,1\}$ denotes the output of Boolean function $g_j(x_i)$. We assume each element in the input sequence to be used at most twice along an accepting path. This assumption in reasonable, since any collection of inequalities in the form $x_i \leq t_j$ and $x_i > t_j$ defines an interval and so is defined by a pair of comparisons. This implies the depth is at most twice the number of input elements. We assume that Evaluation procedure (Algorithm 6.6) is oblivious to the tuples in P_{s_τ} (and is only allowed to know d, and hence n).

6.5.2 Building Blocks

We want the obfuscated tree to be *unlearnable* i.e. we aim to hide $\mathsf{asset}(C)$ from a PPT adversary. To achieve the same, we develop a library of building blocks that enables encoding arbitrary integer intervals, which we leverage to build our obfuscator.

6.5.2.1 Converting Inequality into Intervals

A decision node associates function $g : \{0,1\}^\ell \to \{0,1\}$, such that $g(x) = 1$, if $x \leq t$ and 0 otherwise, where t is any integer between 0 and $2^\ell - 1$. Thus, the Boolean function splits the integer interval $[0, 2^\ell)$ at each node into two distinct partitions, call it $\mathcal{X} = [0, t+1)$ and $\mathcal{X}' = [t+1, 2^\ell)$, where each interval contains ℓ-bit binary encoding of the integers. We further divide intervals $\mathcal{X}$ and $\mathcal{X}'$ into a sequence of disjoint sub-intervals of the form $[a, a + 2^p)$, which is the primary building block of our construction. We present the formal descriptions in Algorithm 6.1 and Algorithm 6.2.

Algorithm 6.1 $\mathsf{GenInt}_{\mathcal{X}}(t)$

Input: $\ell \in \mathbb{N}, t \in [0, 2^\ell)$
Output: $\mathcal{I}_{\mathcal{X}} = \{[a_j, a_j + 2^{p_j})\}_{j \in [k]}$
1: $\mathcal{I}_{\mathcal{X}} = \emptyset$; $\mathsf{temp} = 0$
2: Compute $k = wt(t+1)$
3: Compute $p_1, \ldots, p_k$, such that $t + 1 = \sum_{j=1}^{k} 2^{p_j}$ and $p_j < p_{j-1}$
4: $\mathcal{I}_{\mathcal{X}} = \{[0, 2^{p_1})\}$
5: **for** $j = 2$ to k **do**
6: $\quad a_j = \mathsf{temp} + 2^{p_{j-1}}$
7: $\quad \mathcal{I}_{\mathcal{X}} = \mathcal{I}_{\mathcal{X}} \cup \{[a_j, a_j + 2^{p_j})\}$
8: $\quad \mathsf{temp} = a_j$
9: **end for**
10: **return** $\mathcal{I}_{\mathcal{X}}$

Algorithm 6.2 $\mathsf{GenInt}_{\mathcal{X}'}(t)$

Input: $\ell \in \mathbb{N}, t \in [0, 2^\ell)$
Output: $\mathcal{I}_{\mathcal{X}'} = \{[b_j, b_j + 2^{p'_j})\}_{j \in [k']}$
1: $\mathcal{I}_{\mathcal{X}'} = \emptyset$; $\mathsf{temp} = t + 1$
2: Compute $k' = wt(2^\ell - t - 1)$
3: Compute $p'_1, \ldots, p'_{k'}$, such that $2^\ell - t - 1 = \sum_{j=1}^{k'} 2^{p'_j}$ and $p'_j > p'_{j-1}$
4: $\mathcal{I}_{\mathcal{X}'} = [\mathsf{temp}, 2^{p'_1})$
5: **for** $j = 2$ to k' **do**
6: $\quad b_j = \mathsf{temp} + 2^{p'_{j-1}}$
7: $\quad \mathcal{I}_{\mathcal{X}'} = \mathcal{I}_{\mathcal{X}'} \cup \{[b_j, b_j + 2^{p'_j})\}$
8: $\quad \mathsf{temp} = b_j$
9: **end for**
10: **return** $\mathcal{I}_{\mathcal{X}'}$

Lemma 6.5.1. *Let $\ell \in \mathbb{N}$ and $t \in [0, 2^\ell)$. Consider algorithms $\mathsf{GenInt}_{\mathcal{X}}$ (Algorithm 6.1) and $\mathsf{GenInt}_{\mathcal{X}'}$ (Algorithm 6.2). Let $\mathcal{I}_{\mathcal{X}} \leftarrow \mathsf{GenInt}_{\mathcal{X}}(t)$ and $\mathcal{I}_{\mathcal{X}'} \leftarrow \mathsf{GenInt}_{\mathcal{X}'}$.*

Then $\mathcal{I}_{\mathcal{X}}$ defines sub-intervals of the form $[a, a+2^p)$ for some a and p, whose union is $[0, t+1)$, and $\mathcal{I}_{\mathcal{X}'}$ defines sub-intervals of the form $[b, b+2^{p'})$ for some b and p', whose union is $[t+1, 2^\ell)$.

Proof. $\mathsf{GenInt}_{\mathcal{X}}(t)$ divides $[0, t+1)$ into k disjoint sub-intervals $\{[a_j, a_j + 2^{p_j})\}_{j\in[k]}$ such that $\sum_{j=1}^{k} 2^{p_j} = t+1$ and $p_j < p_{j-1}$. Since $a_1 = 0$, $a_j = a_{j-1} + 2^{p_{j-1}}$, the output is $\mathcal{I}_{\mathcal{X}} = \{[0, 2^{p_1}), [2^{p_1}, 2^{p_1} + 2^{p_2}), \ldots, [\sum_{j=1}^{k-1} 2^{p_j}, \sum_{j=1}^{k} 2^{p_j})\}$ and the union of these intervals is $[0, t+1)$.

$\mathsf{GenInt}_{\mathcal{X}'}(t)$ divides $[t+1, 2^\ell)$ into k' disjoint sub-intervals $\{[b_j, b_j + 2^{p'_j})\}_{j\in[k']}$ such that $\sum_{j=1}^{k'} 2^{p'_j} = 2^\ell - t - 1$ and $p'_j > p'_{j-1}$. Since $b_1 = t+1$, $b_j = b_{j-1} + 2^{p'_{j-1}}$, the output is $\mathcal{I}_{\mathcal{X}'} = \{[t+1, t+1+2^{p'_1}), [t+1+2^{p'_1}, t+1+2^{p'_1}+2^{p'_2}), \ldots, [t+1+\sum_{j=1}^{k'-1} 2^{p'_j}, t+1+\sum_{j=1}^{k'} 2^{p'_j})\}$ and the union of these intervals is $[t+1, 2^\ell)$. □

6.5.2.2 Intersection of Intervals

Let $\mathcal{I}_{\mathcal{X}}$ be a set of sub-intervals, all of the form $[a, a+2^j)$ for some a and j. Let $\mathcal{I}_{\mathcal{X}'}$ be a set of sub-intervals, all of the form $[b, b+2^r)$ for some b and r. Define intersection of $\mathcal{I}_{\mathcal{X}}$ and $\mathcal{I}_{\mathcal{X}'}$ as $\mathcal{I} = \{I \cap J : I \in \mathcal{I}_{\mathcal{X}}, J \in \mathcal{I}_{\mathcal{X}'}\} \setminus \emptyset$.

Lemma 6.5.2. *Let $\ell \in \mathbb{N}$. Consider algorithms $\mathsf{GenInt}_{\mathcal{X}}$ (Algorithm 6.1) and $\mathsf{GenInt}_{\mathcal{X}'}$ (Algorithm 6.2). Let c, $c+w \in \mathbb{Z}$ be such that $0 \le c < c+w < 2^\ell$. Let $\mathcal{I}_{\mathcal{X}} \leftarrow \mathsf{GenInt}_{\mathcal{X}}(c+w)$, $\mathcal{I}_{\mathcal{X}'} \leftarrow \mathsf{GenInt}_{\mathcal{X}'}(c)$. Let $\mathcal{I} = \mathcal{I}_{\mathcal{X}} \cap \mathcal{I}_{\mathcal{X}'} = \{I \cap J : I \in \mathcal{I}_{\mathcal{X}}, J \in \mathcal{I}_{\mathcal{X}'}\} \setminus \emptyset$. Then every interval in $\mathcal{I}$ is of the form $[a, a+2^i)$, for some i, and $|\mathcal{I}| \le 2\ell - 2$.*

Proof. We will prove that if $I \in I_X$ and $J \in I_{X'}$ are such that $IJ \neq$, then IJ or JI. $\mathsf{GenInt}_{\mathcal{X}}(c+w)$ divides $[0, c+w+1)$ into k disjoint sub-intervals $[a_j, a_j + 2^{p_j})$, where k is the Hamming weight of ℓ-bit binary encoding of $c + w + 1$. Since $p_j < p_{j-1}$, we can conclude that $2^{p_1} \le c+w+1 < 2^{p_1+1}$ and $\mathcal{I}_{\mathcal{X}} = \{[0, 2^{p_1}), \ldots, [2^{p_1} + \cdots + 2^{p_{k-1}}, 2^{p_1} + \cdots + 2^{p_k})\}$, where $\ell > p_1$. Since $c \in [0, c+w+1)$, we consider the following:

- If $c+1 = 2^q$, where $q \le p_1$, then $\mathcal{I}_{\mathcal{X}'} = \{[2^i, 2^{i+1})\}_{i\in\{q, q+1, \ldots, \ell-1\}}$, and it can be clearly seen that for every non-empty intersection $I \cap J$, either $I \subseteq J$ or $J \subseteq I$.

- If $2^{q-1} < c+1 < 2^q$, where $q \leq p_1$. Let k' be the Hamming weight of ℓ-bit binary encoding of $2^q-(c+1)$. Then, for $J \in \{[c+1, c+1+2^{p'_1}), \ldots, [c+1+\cdots+2^{p'_{k'-1}}, c+1+\cdots+2^{p'_{k'}})\}$, where $2^q = c+1+\cdots+2^{p'_{k'}}$, $J \subseteq I$, where $I = [0, 2^{p_1})$. Also note, for $J = [2^{p_1}, 2^{p_1+1})$ and $I \in \mathcal{I}_{\mathcal{X}} \setminus \{[0, 2^{p_1})\}$, $I \subseteq J$.
- If $2^{p_1}+\cdots+2^{p_{m-1}} \leq c+1 < 2^{p_1}+\cdots+2^{p_m}$, where $m \leq k$. Since $\mathsf{GenInt}_{\mathcal{X}'}(c)$ divides $[c+1, 2^\ell)$ into k' sub-intervals $\{[b_r, b_r+2^{p'_r})\}_{r\in[k']}$, then let $p'_1 < \cdots < p'_s \leq p_m < p'_{s+1} < \cdots < p'_{k'}$ for some $s < k'$. Let $I = [2^{p_1}+\cdots+2^{p_{m-1}}, 2^{p_1}+\ldots,+2^{p_m}) \in \mathcal{I}_{\mathcal{X}}$. Then, for a non-empty intersection J, let $J \in \mathcal{I}_{\mathcal{X}'}$ be such that $I \cap J \neq \phi$. Then J is an element of the set $\{[c+1, c+1+2^{p'_1}), \ldots, [c+1+\cdots+2^{p'_{s-1}}, c+1+\cdots+2^{p'_s})\}$. Note that, $2^{p_1}+\cdots+2^{p_m} = c+1+\cdots+2^{p'_s}$ as 0 to $m-1$ bits of $2^\ell-(c+1)$ and $2^{p_1}+\cdots+2^{p_m}-(c+1)$ are equal, and thus $J \subseteq I$. For all other non-empty intersections, $I \subseteq J$.

It is clear from Algorithms 6.1 and 6.2 that $|\mathcal{I}_{\mathcal{X}}|, |\mathcal{I}_{\mathcal{X}'}| \leq \ell$ (when the Hamming weight of $|\mathcal{X}|$ and $|\mathcal{X}'|$ are ℓ). Let $\mathcal{I} = \mathcal{I}_{\mathcal{X}} \cap \mathcal{I}_{\mathcal{X}'}$ and consider any $H \in \mathcal{I}$. By the above, either $H = I$ for some $I \in \mathcal{I}_{\mathcal{X}}$ or $H = J$ for some $J \in \mathcal{I}_{\mathcal{X}'}$. It follows that $|\mathcal{I}| \leq |\mathcal{I}_{\mathcal{X}}| + |\mathcal{I}_{\mathcal{X}'}|$. In the case $H = I$, there is some $J \in \mathcal{I}_{\mathcal{X}'}$ such that $I \subseteq J$, and so J itself is not counted in $|\mathcal{I}|$. Similarly in the case $H = J$ there is some $I \in \mathcal{I}_{\mathcal{X}}$ such that $J \subseteq I$, and so I is not counted in $|\mathcal{I}|$. Hence $|\mathcal{I}| \leq |\mathcal{I}_{\mathcal{X}}| + |\mathcal{I}_{\mathcal{X}'}| - 2 \leq 2\ell - 2$. □

6.5.2.3 Calculating the Encodings

Let $\mathcal{I}$ be a set of sub-intervals, all of the form $[a, a+2^j)$. The encoder (Algorithm 6.3) receives $\mathcal{I}$ as input and outputs the set of encodings $\{h_1, \ldots, h_{|\mathcal{I}|}\}$. Define a family of functions $\mathcal{F}$ as follows: $\mathcal{F} = \{f_0, \ldots, f_{\ell-1}\}$ where $f_i(y) : \{0,1\}^\ell \to \{0,1\}^{\ell-i}$ such that $f_i(y) = \lfloor \frac{y}{2^i} \rfloor$. Let $H : \{0,1\}^* \to \{0,1\}^\omega$ be a cryptographic hash function with $\omega > 2\ell$ so that H is injective when restricted to inputs of length at most ℓ. A random oracle is not generally injective, but when the output length is large enough compared to the input then it will be. Our argument behind imposing this additional constraint on H is that every $|\lfloor \frac{y}{2^i} \rfloor| \leq \ell$ maps to a unique encoding in E. Algorithm 6.4 receives as input $E \leftarrow \mathsf{IntEnc}(\mathcal{I})$ and $x \in \{0,1\}^\ell$ and outputs 1, if x belongs to any of the sub-intervals in $\mathcal{I}$.

Algorithm 6.3 $\mathsf{IntEnc}(\{[a_j, a_j + 2^{p_j})\}_{j\in[k]})$

1: $E = \emptyset$
2: **for** $j = 1$ to k **do**
3: Compute $\mu_j = f_{p_j}(a_j)$
4: Compute $h_j = H(\mu_j)$.
5: $E = E \cup \{h_j\}$
6: **end for**
7: **return** E

Decoding. The decoding algorithm 6.4 receives as input $E \leftarrow \mathsf{IntEnc}\ (\mathcal{I})$ and $x \in \{0,1\}^\ell$ and outputs 1 if x belongs to any of the sub-intervals in $\mathcal{I}$.

Algorithm 6.4 Dec (with embedded data E)

Input: $\ell \in \mathbb{N}, x \in \{0,1\}^\ell$
Output: 0 or 1.
1: **for** $i = 0$ to $\ell - 1$ **do**
2: Compute $H(f_i(x))$
3: **if** $H(f_i(x)) \in E$ **then**
4: **return** 1
5: **end if**
6: **end for**
7: **return** 0

Lemma 6.5.3 (Correctness). *Let $c, c + w \in [0, 2^\ell)$. Consider algorithms $\mathsf{GenInt}_\mathcal{X}$ (Algorithm 6.1), $\mathsf{GenInt}_{\mathcal{X}'}$ (Algorithm 6.2), IntEnc (Algorithm 6.3), Dec (Algorithm 6.4) and input $x \in \{0,1\}^\ell$. Let $\mathcal{I}_\mathcal{X} \leftarrow \mathsf{GenInt}_\mathcal{X}(c+w)$ and $\mathcal{I}_{\mathcal{X}'} \leftarrow \mathsf{GenInt}_\mathcal{X}(c)$ and $\mathcal{I} \leftarrow \mathcal{I}_\mathcal{X} \cap \mathcal{I}_{\mathcal{X}'}$. Let $H : \{0,1\}^* \to \{0,1\}^\omega$ be injective when restricted to inputs of length at most ℓ (e.g. $\omega > 2\ell$). Then $x \in (c, c + w]$ if and only if Dec outputs 1.*

Proof. Let $\mathcal{I} = \{[a_j, a_j + 2^{p_j})\}_{j\in[k]}$, then from Lemma 6.5.2, we can say that $\bigcup_{j=1}^{k}[a_j, a_j + 2^{p_j})$ contains all sub-intervals in $(c, c + w]$. Let x be an integer in $(c, c + w]$, then it must belong to at least one of the sub-intervals in $\mathcal{I}$. Algorithm 6.3 computes $f_{p_j}(y) = \mu_j$, for every $[a_j, a_j + 2^{p_j}) \in \mathcal{I}$. If $x \in [a_j, a_j + 2^{p_j})$, then there exists an $i \in \{0, \ldots, \ell - 1\}$ such that $f_i(x) = f_{p_j}(a_j) = \mu_j$. Hence $H(f_i(x)) \in \{h_1, \ldots, h_{|\mathcal{I}|}\} \leftarrow \mathsf{IntEnc}(\mathcal{I})$ and Dec outputs 1. If $x \notin [a_j, a_j + 2^{p_j})$, $\nexists\, i = \{1, \ldots, \ell - 1\}$, such that $f_i(x) \in (\mu_j)_{j=1}^{|\mathcal{I}|}$ and therefore, $(h_1, \ldots, h_{|\mathcal{I}|})$ will not contain $H(f_i(x))$. Finally, Dec will correctly reject the input. □

Example Parameters. We illustrate the interval encoding technique with the following concrete setting: consider the interval membership predicate $x \in (10, 14]$.

To encode the interval, calculate $\mathcal{I}_{\mathcal{X}} \leftarrow \mathsf{GenInt}_{\mathcal{X}}(14)$ and $\mathcal{I}_{\mathcal{X}'} \leftarrow \mathsf{GenInt}_{\mathcal{X}'}(10)$ which gives the set of sub-intervals in $[0, 15)$ and $[15, 256)$ for $\ell = 8$. Finally, calculate $\mathsf{IntEnc}(\mathcal{I})$, where $\mathcal{I} \leftarrow \mathcal{I}_{\mathcal{X}} \cap \mathcal{I}_{\mathcal{X}'}$. The sets are indicated as follows:

$\mathcal{I}_{\mathcal{X}} = \{[0, 8), [8, 12), [12, 14), [14, 15)\}$

$\mathcal{I}_{\mathcal{X}'} = \{[11, 12), [12, 16), [16, 32), [32, 64), [64, 128), [128, 256)\}$

$\mathcal{I} = \{[11, 12), [12, 14), [14, 15)\}$

$\mathsf{IntEnc}(\mathcal{I}) = \{H(f_0(00001011), H(f_1(00001100), H(f_0(00001110))\} = \{H(0000\ 1011), H(0000110), H(00001110)\}$

6.5.3 Obfuscator $\mathcal{O}$

We now present our decision tree obfuscator $\mathcal{O}$. We remind the readers that $\mathcal{S}$ is the set of terminal nodes, and $s_\tau = 1$ denotes an accepting path through the tree (from root to leaf).

Each accepting path is a conjunction of inequalities, and our objective is the obfuscate the conjunctions using our encoding technique for interval membership functions. Note that, our construction uses the fact that the terms in a conjunction can be reordered. Recall that x_i is compared at most twice along an accepting path, and hence the accepting path corresponds to $x_i \in (c_i, c_i + w_i]$ for every $i \in [n]$.

Precisely, the obfuscator works as follows: to encode $(c_i, c_i + w_i]$, calculate $\mathcal{I}^i_{\mathcal{X}} \leftarrow \mathsf{GenInt}_{\mathcal{X}}(c_i + w_i)$ and $\mathcal{I}^i_{\mathcal{X}'} \leftarrow \mathsf{GenInt}_{\mathcal{X}'}(c_i)$, which gives the set of sub-intervals in $[0, c_i + w_i + 1)$ and $[c_i + 1, 2^\ell)$. Next, determine $\mathcal{I}^i \leftarrow \mathcal{I}^i_{\mathcal{X}} \cap \mathcal{I}^i_{\mathcal{X}'}$ which is a set of sub-intervals whose union is $(c_i, c_i + w_i]$. To encode each sub-interval in $\mathcal{I}^i$, calculate encodings $\mathcal{B}^i \leftarrow \mathsf{IntEnc}(\mathcal{I}^i)$. Let $H_c : \{0,1\}^* \to \{0,1\}^q$ be a cryptographic hash function with $q \geq 2\omega n$ (and hence injective on restricted inputs). The main idea is to concatenate each combination of n hashes sorted in ascending order of i for each entry in $\mathcal{B}^i$, and apply H_c ; call the set of hashes $\mathcal{B}$. If $|\mathcal{B}| < 2^d(2\ell - 2)^n$, add dummy entries drawn uniformly at random from $\{0,1\}^q$. Finally, output $\mathcal{B}$. We give the formal details in Algorithm 6.5.

Algorithm 6.5 $\mathcal{O}\,(d, n, \ell \in \mathbb{N}, \mathsf{asset}(C))$

$\mathcal{B} = \emptyset$
$\alpha = 2^d(2\ell - 2)^n$
for all τ such that $s_\tau = 1$ **do**
  Compute $P_{s_\tau} = (\llbracket t_j, i, b \rrbracket : v_j$ is an ancestor of s_τ, and $b = g_j(t_j))$
  for $i = 1$ to n **do**
    $\mathcal{I}^i_{\mathcal{X}} = \mathcal{I}^i_{\mathcal{X}'} = [0, 2^\ell)$
    if $\llbracket t_{j_1}, i, 1 \rrbracket \in \mathsf{path}_{s_\tau}$ **then**
      $\mathcal{I}^i_{\mathcal{X}} \leftarrow \mathsf{GenInt}_{\mathcal{X}}(t_{j_1})$
    end if
    if $\llbracket t_{j_2}, i, 0 \rrbracket \in \mathsf{path}_{s_\tau}$ **then**
      $\mathcal{I}^i_{\mathcal{X}'} \leftarrow \mathsf{GenInt}_{\mathcal{X}'}(t_{j_2})$
    end if
    $\mathcal{I}^i \leftarrow \mathcal{I}^i_{\mathcal{X}} \cap \mathcal{I}^i_{\mathcal{X}'}$
    if $(\mathcal{I}^i == [0, 2^\ell))$ **then**
      $\mathcal{B}^i = \{1^\ell\}$
    else
      $\mathcal{B}^i \leftarrow \mathsf{IntEnc}(\mathcal{I}^i)$
    end if
  end for
  Denote $\mathcal{B}^i = (h^i_1, \ldots, h^i_{\sigma_i})$, where $\sigma_i \leq 2\ell - 2$ for each i.
  for all $((q_1, \ldots, q_n) \in [\sigma_1] \times \cdots \times [\sigma_n])$ **do**
    $\mathcal{B} = \mathcal{B} \cup \{H_c(\|^n_{i=1} h^i_{q_i})\}$
  end for
end for
while $(|\mathcal{B}| < \alpha)$ **do**
  $r \leftarrow\!\!\$\ \{0,1\}^q$
  $\mathcal{B} = \mathcal{B} \cup \{r\}$
end while
return $\mathcal{B}$

6.5.4 Obfuscated Decision Tree Evaluation

On input $(x_i)_{i\in[n]}$, the evaluation procedure calculates all possible encodings by evaluating $H(f_p(x_i))$ for every $p \in \{0, \ldots, \ell - 1\}$; call it $\mathcal{E}^i = \{h^i_1, \ldots, h^i_\sigma\}$. Finally compute all possible $H_c(\|_{i\in[n]} h^i_{q_i})$, where encodings are listed in ascending order of i. For an accepting input, one of the computed hash values belongs to the set $\mathcal{B}$ published by $\mathcal{O}$. Formally, the evaluation procedure is specified in Algorithm 6.6.

Algorithm 6.6 Eval ($\mathcal{B}$ with $|\mathcal{B}| = 2^d(2\ell-2)^n$)

Input: $(x_1, \ldots, x_n)$, ℓ, d
Output: 0 or 1

for $i = 1$ to n **do**
  $\mathcal{E}^i \leftarrow \{1^\ell\} \cup \{H(f_0(x_i)), \ldots, H(f_{\ell-1}(x_i))\}$
end for
$\mathcal{E}^i = \{h_1^i, \ldots, h_\sigma^i\}$, $\sigma \leq \ell + 1$
for all $(q_1, ..., q_i) \in [\sigma_1] \times \cdots \times [\sigma_n]$ **do**
  if $H_c(\|_{i\in[n]} h_{q_i}^i) \in \mathcal{B}$ **then**
    return 1
  end if
end for
return 0

6.6 Correctness and Efficiency

In this section we analyze the correctness and efficiency of the obfuscator.

Lemma 6.6.1 (Correctness). *Let $\lambda \in \mathbb{N}$ be the security parameter, and let n, ℓ, ω and q be polynomials in λ. Consider Algorithms $\mathcal{O}$ (Algorithm 6.5) and* Eval *(Algorithm 6.6), and an input $(x_i)_{i\in[n]}$ with $x_i \in \{0,1\}^\ell$. Let $H : \{0,1\}^* \to \{0,1\}^\omega$ with $\omega > 2\ell$, and let $H_c : \{0,1\}^* \to \{0,1\}^q$ with $q > 2\omega n$. Then for every $[\![t_j, i, b]\!] \in P_{s_\tau}$, where $t_j \in [0, 2^\ell)$, $i \in \{1, \ldots, n\}$, $b \in \{0,1\}$, for every $\mathcal{S} = (s_1, \ldots, s_{2^d})$ with $s_\tau \in \{0,1\}$, for every $\mathcal{B} \leftarrow \mathcal{O}$, and for every input $(x_i)_{i\in[n]}$, if $C((x_i)_{i\in[n]}) = 1$, then* Eval *outputs 1, else it outputs 0 with overwhelming probability in λ.*

Proof. Algorithm 6.5 calculates set of encodings $\mathcal{B}^i \leftarrow \mathsf{IntEnc}(\mathcal{I}^i)$ for every $i \in [n]$. If there is some i such that $[\![t_j, i, b]\!] \notin P_{s_\tau}$ (which means x_i is not compared in the path) then $\mathcal{B}^i = \{1^\ell\}$. Let $(x_i)_{i\in[n]}$ be an accepting input. From Definition 6.4.3, $x_i \in (c_i, c_i + w_i]$, for every $i \in [n]$. From Lemma 6.5.3, if $x_i \in (c_i, c_i + w_i]$, then there exists a unique $h_k^i \in \mathcal{E}^i$, such that $h_k^i \in \mathcal{B}^i$. If $\exists\, i$, such that $[\![t_j, i, b]\!] \notin P_{s_\tau}$, then $h_k^i = 1^\ell \in \mathcal{B}^i$. Thus, for an accepting input $(x_i)_{i\in[n]}$, there exists a unique $(h_{k_1}^1, \ldots, h_{k_n}^n)$, where $h_{k_i}^i \in \mathcal{E}^i$ and $H_c(h_{k_1}^1 \| \ldots \|_{k_n}^n)$ will be contained in $\mathcal{B}$, and Algorithm 6.6 will correctly output 1.

If $C((x_i)_{i\in[n]}) \neq 1$, then by Lemma 6.5.3, hash values computed from (x_i) will not match the hash values input to H_c. As we choose parameters such that H_c is injective, $H_c(h_{k_1}^1 \| \ldots \| h_{k_n}^n)$ will not be contained in $\mathcal{B}$, except if it equals to one

of its dummy entries. Since the number of possible encodings (in Eval) input to H_c is $2^{(\ell+1)n}$, and $w > 2\ell$, $q > 2\omega n$, the probability that Eval incorrectly accepts the input is given by $\frac{2^{(\ell+1)n}}{2^q} = \frac{1}{2^{3\ell n}}$. Finally, the probability that the α hash values output by $\mathcal{O}$ are good is given by $(1 - \frac{1}{2^{3\ell n}})^\alpha \approx 1 - negl(\lambda)$.

□

Efficiency. We now discuss the size complexity of the obfuscated decision tree. Let $\lambda \in \mathbb{N}$, and ℓ, q be polynomials in λ.

Along an accepting path, the upper bound on the size of $\mathcal{B}^i$ is $2\ell - 2$ (see Lemma 6.5.2). There are n such set of encodings, hence the total number of possible encodings input to H_c is $(2\ell - 2)^n$. Since, the number of accepting paths is $\mathrm{O}\big(2^d\big)$, the overall complexity of storing the obfuscated tree is $\mathrm{O}\big(2^{d+n} \cdot \ell^n \cdot q\big)$, where q is the output size of H_c. Note that the upper bound $\alpha = 2^d(2\ell - 2)^n$ on the number of hashes is much larger than will be needed for most evasive decision trees, so in practice this parameter could be chosen a lot smaller to get a more compact obfuscated program.

Next, we analyze the time complexity of the evaluation procedure (Algorithm 6.6). Each node corresponds to set of encodings $\mathcal{E}^i$, where $|\mathcal{E}^i| = \ell + 1$. For n input attributes, the overall running time of the evaluation algorithm is of the order $\mathrm{O}\big(\ell^n \log(\alpha)\big)$ operations. Since the query response time is exponential in n, we restrict to decision trees of constant number of input elements.

We now prove polynomial slowdown only for some special cases.

Lemma 6.6.2 (Polynomial Slowdown). *Let $\lambda \in \mathbb{N}$ be the security parameter and ℓ be a polynomial in λ. Define $\mathcal{T}_\lambda$ to be a special family of evasive decision trees, where $d = 5$, $n \leq 4$ and $\ell = \frac{\lambda}{4}$. Then for every $C \leftarrow \mathcal{T}_\lambda$, there exists a polynomial p such that the running time of $\mathcal{O}(C)$ is bounded by $\mathrm{O}\big(p(\lambda)\big)$.*

Proof. Let $C \leftarrow \mathcal{T}_\lambda$ computes whether an input $(x_i)_{i=1}^n$ is contained in the decision region defined by intervals $(c_i, c_i + w_i]$, where $w_i \in (0, w_{max}]$. From Lemma 6.4.2, $w_{max} \leq 2^{\ell - \frac{\lambda}{n}}$, which specifies the maximum width of the intervals. For evasiveness, we require $\ell - \frac{\lambda}{n} \geq 0$, which gives $\ell n \geq \lambda$. Now, for $\ell = \frac{\lambda}{4}$ and $n = 4$, we get $\ell n = \lambda$, which is a feasible condition for evasiveness. The cost of evaluating $\mathcal{O}$ is given by $\ell^n = (\frac{\lambda}{4})^4$. □

d	ℓ	n	λ	Program size	Evaluation cost
5	64	4	128	$126^4 \times 512$	$65^4 \times 512$
3	64	2	64	$126^2 \times 512$	$65^2 \times 512$

Table 6.1: Example parameter sets for an obfuscated decision tree with $w_{max} \leq 2^{\ell - \frac{\lambda}{n}}$. For $q = 512$ bits and *one* accepting path (q is the output size of hash function H_c), we calculate the size of the obfuscated program and the cost of the evaluation (Algorithms 6.5 and 6.6).

Parameters for Secure Construction of $\mathcal{O}_{\mathcal{D}}$. The aim of this section is to explain how to choose the necessary parameters that adhere to the restrictions of a secure and efficient obfuscator.

Our a priori knowledge on the parameters are as follows: $\ell = \ell(\lambda), d = d(\lambda), n = n(\lambda)$, where $\lambda \in \mathbb{N}$ is the security parameter. For evasiveness, the maximum width of the intervals representing the *decision regions* should be $w_{max}(\lambda) \leq 2^{\ell(\lambda) - \frac{\lambda}{n(\lambda)}}$ keeping to Lemma 6.4.2. Adding to that, we require $d \leq n + 1$.

We give some example parameters along with their bit-security in Table 6.1.

6.7 Proof of VBB Security

We prove VBB security in the random oracle model. For simplicity we restrict to decision trees with one accepting path, and let $\alpha = (2\ell - 2)^n$ be an upper bound on the number of hash values required for the obfuscated program. We start with a brief introduction to the random oracle model, followed by the proof of VBB security.

Random Oracle Model. The model is introduced by Mihir Bellare and Phillip Rogaway as a methodology for designing and validating cryptographic schemes, that cannot be proven secure in the standard model (under any computational hardness assumptions) and has found applications in a number of security proofs. Random oracle is an abstraction of an idealized model that incorporates a public random function $\mathcal{R}$, with all the parties (obfuscator, adversary, etc.) having oracle access to $\mathcal{R}$. These proofs are elegant in a way that it provides confidence in the 'soundness' of the designed scheme as the possible attacks can only stem out of the weaknesses in the real-world instantiation of the random oracle. The functionality of ROM is as follows:

Functionality of ROM ($\mathcal{F}_{RO}$). A random function $\mathcal{R}$ is chosen uniformly from the set of all functions and all the parties are given oracle access to the function. Let $\mathcal{E}$ be the domain and $\mathcal{K}$ be the co-domain of $\mathcal{R}$. Let L be the list of pairs in $\mathcal{E} \times \mathcal{K}$, which is initially empty. When queried with x, the oracle returns y, if $(x, y) \in L$, otherwise it stores and returns a string selected uniformly from the co-domain $\mathcal{K}$.

The security of our construction relies upon the existence of cryptographic hash functions $H : \{0,1\}^* \to \{0,1\}^\omega$ and $H_c : \{0,1\}^* \to \{0,1\}^q$, which we model as random oracles in our security proof. Our objective is to show that a PPT adversary having access to the obfuscated function has no advantage over a simulator having oracle access to the function. This is achieved by a simulating the adversary in execution and outputting what the adversary does, such that the adversary cannot distinguish between the simulation and the real environment, a notion called simulation-based obfuscation. We first give a brief intuition to our security proof.

Consider a simulator $\mathcal{S}$ who samples parameters ℓ, n following the conditions in Lemma 6.6.2, and sends them to adversary $\mathcal{A}$. $\mathcal{A}$ now samples C and provides $\mathcal{S}$ oracle access to C. Since $\mathcal{S}$ does not know the program C, it simulates the obfuscated program and random oracles and provides answers to $\mathcal{A}$'s queries.

If an adversary never queries the circuit with an accepting input, the everything is a correct simulation. However, if the adversary does query the circuit with an accepting input, then the security reduction immediately uses this clue to mount a model-extraction attack, and hence learn the corresponding accepting path in the decision tree. The security reduction can run the obfuscator correctly for that accepting path, and program the random oracles to be consistent with the simulated $\mathcal{O}$ (the reduction does not learn anything about the other possible accepting paths). Hence, again everything is a correct simulation.

Theorem 6.7.1. *Let $\lambda \in \mathbb{N}$ be the security parameter Let $\ell = \ell(\lambda)$ and $\alpha = \alpha(\lambda)$ satisfy the conditions required for Lemma 6.6.2. Let $\mathcal{D}_\lambda$ be a distribution of evasive decision trees. Then for random oracles $H : \{0,1\}^* \to \{0,1\}^\omega$ and $H_c : \{0,1\}^* \to \{0,1\}^q$, the decision tree obfuscator $\mathcal{O}$ is a VBB obfuscator.*

Proof. As evident from Lemma 6.6.1, $\mathcal{O}$ satisfies functionality preservation. Lemma 6.6.2 shows that the obfuscator causes polynomial slowdown. Thus it suffices to

show that there exists a (non-uniform) PPT simulator $\mathcal{S}$ for every (non-uniform) PPT adversary $\mathcal{A}$, such that for an ensemble of decision tree evasive distributions $\mathcal{D}_\lambda$, the following holds:

$$\left| \Pr_{C \leftarrow D_\lambda}[\mathcal{A}(\mathcal{O}(1^\lambda, C)) = 1] - \Pr_{C \leftarrow D_\lambda}[\mathcal{S}^C(1^\lambda) = 1] \right| \leq \mu(\lambda)$$

Every $C \leftarrow \mathcal{D}_\lambda$ identifies unique $(c_1, \ldots, c_n) \leftarrow X_n$ and $(w_1, \ldots, w_n) \leftarrow (0, w_{max}]^n$, and on input $(x_1, \ldots, x_n)$ determines if $x_i \in (c_i, c_i + w_i]$ for all $i \in [n]$. Let $\mathcal{O}(1^\lambda, C) = \{h_1, \ldots, h_\alpha\}$ denote the correct obfuscation of C. Let $\mathcal{A}$ be a PPT adversary that takes as input $\mathcal{O}(1^\lambda, C)$. We use this adversary to design a PPT simulator $\mathcal{S}$ that simulates an execution of $\mathcal{A}$.

Since $\mathcal{A}$ expects the oracles H and H_c, $\mathcal{S}$ provides a simulation of both the oracles. In order to record the choices of the random oracles, $\mathcal{S}$ maintains two tables : $\mathcal{T}_1$ to record responses for queries to H and $\mathcal{T}_2$ to record responses for queries to H_c. Since $\mathcal{S}$ does not have access to $\mathcal{O}(1^\lambda, C)$, it prepares a purported obfuscation of C as follows: $\mathcal{S}$ samples α values uniformly at random from the co-domain of H_c, and sends $\{h'_1, \ldots, h'_\alpha\}$ to $\mathcal{A}$.

We assume that $\mathcal{A}$ makes polynomially many queries to both the random oracles. When $\mathcal{A}$ queries oracle H with $\mathbf{u}^*$, $\mathcal{S}$ looks for v such that $(\mathbf{u}^*, v) \in \mathcal{T}_1$ and returns it to the adversary. If no such v exists, then the simulation samples a distinct $v \in \{0,1\}^\omega$ uniformly at random, adds $(\mathbf{u}^*, v)$ to $\mathcal{T}_1$, and returns v to $\mathcal{A}$.

When $\mathcal{A}$ makes a query $\mathbf{h}^*$ to the random oracle H_c, the simulator looks for a val such that $(\mathbf{h}^*, val) \in \mathcal{T}_2$ and returns it to $\mathcal{A}$. If there are no entries corresponding to $\mathbf{h}^*$, the simulator parses $\mathbf{h}^*$ as $(h_1^*, \ldots, h_n^*)$ and looks up table $\mathcal{T}_1$ to find an entry corresponding to each parsed string. If there does not exist such an entry in $\mathcal{T}_1$, then a distinct $val \in \{0,1\}^q$ is chosen uniformly at random, $(\mathbf{h}^*, val)$ is added to $\mathcal{T}_2$, and val is returned it $\mathcal{A}$.

If there exists a unique u such that $(u, h_i^*) \in \mathcal{T}_1$, then the simulator calculates $j \leftarrow \ell - |u|$, where $|u|$ denotes the bit length of u, and $x_i \leftarrow u \times 2^j$. Since u corresponds to μ_j (for a correct input), adding j zeroes yields an accepting input for C. Eventually, the simulator queries the oracle C with the $(x_i)_{i \in [n]}$. If C returns 1, $\mathcal{S}$ determines the c_i's and w_i's by doing standard model extraction attack for that *single* accepting path, calculates pairs (u, v) and registers the entries in $\mathcal{T}_1$.

Note that we do not learn anything about other possible accepting paths. Thereafter the simulator calculates the α input entries of $\mathcal{T}_2$, defines them to be α entries from the already published set $\{h'_1, \ldots, h'_\alpha\}$, registers the pairs in $\mathcal{T}_2$ and returns val to the adversary. If there are multiple entries in $\mathcal{T}_1$, the simulation halts.

We describe the simulation in form of pseudo code in Algorithm 6.7 and 6.8.

Algorithm 6.7 Oracle $H(\mathbf{u}^*)$

Find all v such that $(\mathbf{u}^*, v) \in \mathcal{T}_1$
if no such v exists **then**
  $v \leftarrow\$ \{0,1\}^\omega$
  $\mathcal{T}_1 \leftarrow \mathcal{T}_1 \cup (\mathbf{u}^*, v)$
else
  if $\exists\, w\ (\mathbf{u}^* \neq w)$, such that $(\mathbf{u}^*, v) \in \mathcal{T}_1$ and $(w, v) \in \mathcal{T}_1$ **then**
    return $\perp$
  end if
end if
return v

Algorithm 6.8 Oracle $H_c(\mathbf{h}^*)$

Find all val such that $(\mathbf{h}^*, val) \in \mathcal{T}_2$
if no such val exists **then**
  Parse $\mathbf{h}^* = (h_i^*)_{i\in[n]}$
  counter $= 0$
  for $i = 1$ to n **do**
    if $(u, h_i^*) \in \mathcal{T}_1$ **then**
      counter $\leftarrow$ counter $++$
      $j \leftarrow \ell - |u|$
      $x_i \leftarrow u \times 2^j$
    end if
  end for
  if (counter $== n$) **then**
    $b \leftarrow \mathcal{S}^C(x_1, \ldots, x_n)$
    if $(b == 1)$ **then**
      Calculate $(c_i, w_i)_{i\in[n]}$ using model-extraction attack
      Run Algorithm 6.5 and calculate (u, v), h
      $\mathcal{T}_1 \leftarrow \mathcal{T}_1 \cup (u, v)$
      if $\exists\, u_1, u_2\ (u_1 \neq u_2)$, such that $(u_1, v) \in \mathcal{T}_1$ and $(u_2, v) \in \mathcal{T}_1$ **then**
        return $\perp$
      else
        for $i = 1$ to α **do**
          $val_i \leftarrow h'_i$
          $\mathcal{T}_2 \leftarrow \mathcal{T}_2 \cup (h_i, val_i)$
        end for
      end if
    end if
    $val \leftarrow\!\!\$\ \{0,1\}^q$
    $\mathcal{T}_2 \leftarrow \mathcal{T}_2 \cup (\mathbf{h}^*, val)$
  end if
end if
return val

At any point, the simulated view is identical to the real view such that the adversary cannot distinguish between the real and purported obfuscation. □

We now analyze the scenario where the simulator fails due to conflicts in table $\mathcal{T}_1$ and show that the probability of such conflicts is negligible in λ.

Lemma 6.7.1. *Let $\lambda \in \mathbb{N}$ be the security parameter, and let ℓ, α be polynomials in λ. Let $\mathcal{D}_\lambda$ be an ensemble of evasive decision tree distributions, and let $\mathcal{O}(1^\lambda, C)$ be the obfuscation of $C \leftarrow \mathcal{D}_\lambda$. Consider Algorithms 6.7 and 6.8 and random oracles*

$H : \{0,1\}^* \to \{0,1\}^\omega$ and $H_c : \{0,1\}^* \to \{0,1\}^{q(\lambda)}$. Let $\eta = \eta(\lambda)$ be the number of entries in $\mathcal{T}_1$, then there exists a negligible function $\mu(\lambda)$ such that

$$\Pr_{C \leftarrow \mathcal{D}_\lambda}[\,\mathcal{S}^C(1^\lambda) = \perp] \;\leq\; \mu(\lambda)$$

where $\mathcal{S}$ is a PPT algorithm with oracle access to C.

Proof. The simulation fails when there is a conflict in table $\mathcal{T}_1$ and $\mathcal{S}$ halts. Conflicts may arise when $\mathcal{S}$ has responded to a random oracle query $\mathbf{u}^*$ to H with $v \leftarrow\$ \{0,1\}^\omega$ and later, on the hash query $\mathbf{h}^*$ to H_c, it makes a call to oracle C and populates $\mathcal{T}_1$ with a pair (w, v) such that $w \neq \mathbf{u}^*$. Let $\eta = \eta(\lambda)$ be the number of entries in $\mathcal{T}_1$. The probability that a conflict occurs in $\mathcal{T}_1$ is equal to the probability that a hash value is same as at least one of the η values in table $\mathcal{T}_1$. Since $H : \{0,1\}^* \to \{0,1\}^{\omega(\lambda)}$, there are $2^{\omega(\lambda)}$ choices for a hash value. When there are no entries in $\mathcal{T}_1$, the collision probability is 0, when there is one entry in $\mathcal{T}_1$, the collision probability is $\frac{1}{2^{\omega(\lambda)}}$ and continuing the same way, when there are $\eta - 1$ entries in $\mathcal{T}_1$, the probability of collision is $\frac{(\eta-1)}{2^{\omega(\lambda)}}$. Assuming all the samples are independent, the probability with which $\mathcal{S}^C(1^\lambda, \pi)$ fails is given by:

$$\begin{aligned}
\Pr_{C \leftarrow C_\lambda}[\,\mathcal{S}^C(1^\lambda, \pi) = \perp\,] &= \frac{1 + \cdots + (\eta - 1)}{2^{\omega(\lambda)}} \\
&= \frac{\eta^2 - \eta}{2^{\omega(\lambda)+1}} \\
&\leq \mu(\lambda)
\end{aligned}$$

□

6.8 Comparison Analysis

In this section, we present a comparative analysis of our solution with the state-of-art protocols for securing the privacy of the decision tree and user data. In what follows, we identify *three* main factors that contribute as our comparison criteria.

To start with, we identify that almost all decision trees are susceptible to model-extraction attacks, and all the previous works that claim privacy of generic decision trees[2, 188, 225] (to name a few) from a PPT adversary, fail to do so [193] (oracle calls suffice for learning the assets). With this, we base our construction only for a subset of decision trees, which we call evasive. We define the properties that

Scheme	Communication Rounds	Method
[35]	≥ 6	Leveled FHE
[69]	9	ASS
[122]	1	AHE, ASS
Proposed Scheme	0	Cryptographic Hash Functions

Table 6.2: Comparison summary with state-of-art protocols: AHE stands for Additive Homomorphic Encryption, ASS stands for Additive Secret Sharing.

makes a decision tree evasive, and provide computational security guarantees for this class of decision trees, *as it is impossible to preserve the privacy of generic decision trees.*

We next claim that our solution is completely non-interactive. The existing works in privacy-preserving classification in outsourced settings require more than one round of communication at the minimum [2, 122, 26, 195]. For generic two-party protocols, multiple rounds of interaction between the model-provider and the user are required to achieve private comparison at each node of the decision tree [26, 35, 37, 213]. We eliminate interaction between the user and the model-provider (0 round of communication), such that the evaluation is done locally by the user.

Finally, our solution relies upon computationally inexpensive hash functions as opposed to the prevalent heavyweight approaches based on homomorphic encryption [213, 35, 188], which incur heavy costs due to multiplication operations, imposes computational limitations on the degree of the evaluated polynomial, etc. Adding to that, such methods add noise, which results in the increase in size of the ciphertexts. In [60], the authors manage to bring down the overhead, by introducing a homomorphic calculation method that compares a plaintext decision tree model with the encrypted user-queries. We design an encoder for interval-membership function using a hash function H, which we then use as a building block for obfuscating a decision tree classification program, using another hash function H_c.

6.9 Conclusion

In this chapter, we have introduced a new special-purpose VBB obfuscator for evasive binary decision trees. While doing so, we have presented an encoder for hiding parameters in interval-membership functions. Our security analysis follows

the random oracle paradigm [29, 126, 86, 111]. To the best of our knowledge, our construction provides the first non-interactive solution for privacy-preserving classification with decision trees. Furthermore, our methods rely upon hash functions as opposed to computationally expensive cryptographic primitives used by state-of-art protocols, and provides computational security against model-extraction attacks for decision trees that are evasive in nature. Our obfuscation construction blows up exponentially in the depth of the tree, hence an interesting problem would be to investigate solutions that work for more general class of evasive decision trees.

Chapter 7

ObfCP: Platform to Prevent Reverse Engineering of Control Programs

PLCs automate industrial processes, with control programs constituting the *decision-making layer* that bring about desirable changes in the process measurements. In this chapter, we introduce a new adversarial model with a MATE adversary who aims at extracting the operational semantics of the process control framework by obtaining a copy of the control program downloaded from an engineering workstation to a PLC. We focus on preventing such efforts of the adversary, and present a formalization of control program abstraction and its *assets*, the secret values in the program that give away the operational semantics of the process. Our formalism is generic, in the sense that it captures all the comparison operations standardized in IEC 61131-3 [99]. Finally, we introduce ObfCP, a platform that makes use of *cryptographic obfuscation* to secure assets in a control program. We demonstrate an end-to-end case study of control program formalization and present a proof-of-concept implementation over *two* realistic ICS testbeds. Our micro-benchmarks indicate that the proposed platform incurs an overall increase of 3% in the execution time for a single scan cycle, with guarantees of computational security. To the best of our knowledge, this is the first attempt to prevent extraction of process semantics by reverse engineering a recovered implementation of control program.

7.1 Rationale

As discussed in Section 2.1, a PLC interprets input signals from sensors, executes control program pre-configured via engineering workstation, and outputs manipulated variables, which are then transmitted to actuators on top of a plant-wide integration network. Given PLCs and ICS-specific communication protocols do not have provisions for in-built security, a control program can be recovered via unauthorized access to vulnerable ICS equipment, which can then be reverse engineered to recover the process semantics to deliver targeted attacks. We are motivated to look for solutions that prevent such attempts of an adversary, and in this chapter we design a platform that makes use of cryptographic obfuscation to address the aforementioned concern.

Our Contributions. This chapter focuses on preventing an adversary, who obtains a copy of the control program by gaining unauthorized access to the target PLC or ICS network at the *configuration layer* (see Section 2.1), from extracting the operational semantics of the control application. The existing literature highlights how an adversary, with the knowledge of engineering concepts and practices applied within the target domain, extracts the control logic of the target control system by reverse-engineering the high-level process descriptions. However, none of the prior works, to the best of our knowledge, attempt to prevent such efforts of an adversary. To this end, we make use of *program obfuscation*, a tool that protects *assets* [142] by transforming a program to its semantically equivalent counterpart, such that access to the transformed version does not give away the assets, yet preserves the functionality. A summary of our offerings is given as follows:

- We formalize the abstraction of control programs and define its assets (see Definition 7.3.2), *learning* which, an adversary is able to extract the operational semantics of the control applications.
- We propose a new adversarial model, where an adversary obtains a control program, either via gaining access to the PLC or the ICS configuration network used for downloading the control program. As far as we are aware, none of the existing works have put forward similar threat models.
- We introduce ObfCP, a novel legacy-compliant platform for securing assets in a control program by making use of *cryptographic obfuscation*. A part of the proposed platform executes inside a trusted execution environment introduced in the control loop of the target control system.

- We demonstrate an end-to-end case study of our formalization approach and a proof-of-concept implementation of our platform over *two* realistic ICS control applications.

Overall, our work could be considered as the first attempt towards restraining an adversary in extracting the operational semantics of industrial processes, such that the risk of the targeted attacks could be mitigated. We believe that our efforts to put forward this research direction and formalize the control program abstraction would encourage the industrial practitioners and security research community in furthering stronger solutions that prevent reconnaissance of process control within environmental constraints of an ICS.

Organization. This chapter is organized into the following sections. Section 7.2 provides a brief description of the example testbed used to demonstrate our approaches, followed by an introduction to our threat model. Section 7.3 introduces the formal definitions of control programs, along with its assets. Section 7.4 gives a description of the proposed ObfCP platform. Section 7.5 provides a proof-of-concept implementation of our design. An overall conclusion is presented in Section 7.6.

7.2 Preliminaries

We start the section with an experimental water distribution system, which we use as a running example to demonstrate our approach. We follow this with a brief introduction to the *threat model* that we consider for this study.

7.2.1 Case Study

As complicated as the nature of a full-scale ICS could be, we consider a simple realistic water distribution system for demonstrating our approach. The water reservoir in Figure 7.1 (a) supplies chemically-treated water to consumers. A differential pressure sensor calculates the level of water, and a chlorine sensor measures the amount of chlorine in the tank. The water reservoir is connected to a pressure modulating valve that regulates outflow of water from the reservoir through a pump that allows precise control of water-flow rate. Mechanics of how the reservoir is filled and how chlorination level within the reservoir is maintained, is out of the scope of this study. The functional requirement for controlling pressure at the outlet of valve is described in Figure 7.1 (b).

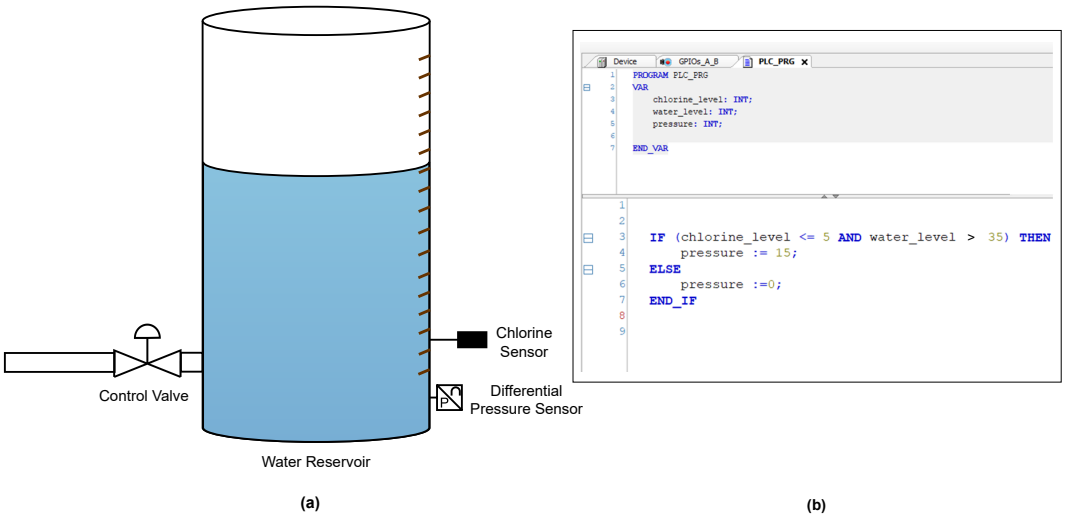


Figure 7.1: Experimental Testbed: (a) Schematics of an example Water Distribution System (b) Control Program written in Structured Text describing the operational behavior of the application.

7.2.2 Threat Model

Our threat model incorporates a MATE adversary (running in time, *polynomial* in the security parameter of the system), who has an understanding of the basic elements of a generic process control framework. The adversary has access to the *software implementation of the control program in the PLC* [129], as well as the ICS plant communication network at the *configuration* layer. The adversary is able to set up a physical device at any point in this network and is assumed to have tools to interact with the industrial protocols and interpret the semantics of the ICS payloads. We assume that the adversary does not have access to the control layer; this is admissible as attacks vectors with lower relative likelihood could be ignored [185]. Furthermore, we assume that the adversary does not have any knowledge about the target plant behaviour, the analog/discrete nature of signals used by sensors and type of sensors such as, temperature sensor, inductive sensor, ultrasonic sensor, etc. used in the control loop of the process.

Adversarial Goal: Reverse engineer the recovered implementation of the control program obtained through the attack surface of the system and *learn* its assets.

Dynamic analysis of control program is out of the scope of this study, as the adversary does not interact with the control layer of the process control framework.

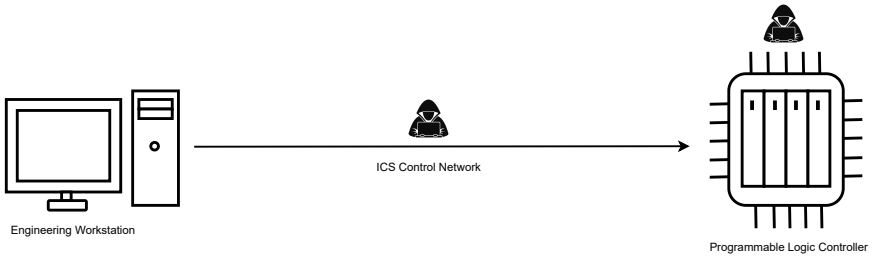


Figure 7.2: Entry points for a potential adversary to interact with the system for obtaining a copy of control program.

This distinguishes our threat model from the one described in [161], where the adversary derives the reactive behaviour of the process control system by recording the traces of control program in the production environment and re-running them in the host simulation environment. In particular, the adversary can monitor the control program execution in the PLC, but cannot modify the process variables to observe the change in values of manipulated variables. Our threat model is different from [133], where the adversary obtains a priori information on the behaviour and control flow of the target control system from electrical installations, compromised HMIs or low-ranking insiders.

7.3 Modeling Control Programs

In this section, we highlight the distinguishing features of a control program to develop an abstraction, and present an approach towards formalizing the derived abstraction. Following this, we define *assets* in a control program and what is meant by *learning* a control program.

7.3.1 Abstraction of Control Programs

Designing cryptographic solutions for preventing reverse engineering of control programs require formalizing them, followed by rigorous mathematical proofs of correctness of the composed solutions, a direction which has not been explored by the state-of-art protocols, to the best of our knowledge. In this section, we discuss the distinguishing features of control programs to develop an unambitious abstraction, independent of IEC 61131-3 specified implementations [99], such that a standardized definition of control programs could be formulated.

Software development of control programs is a step-by-step process that starts with obtaining process descriptions by an operations engineer. A list of field de-

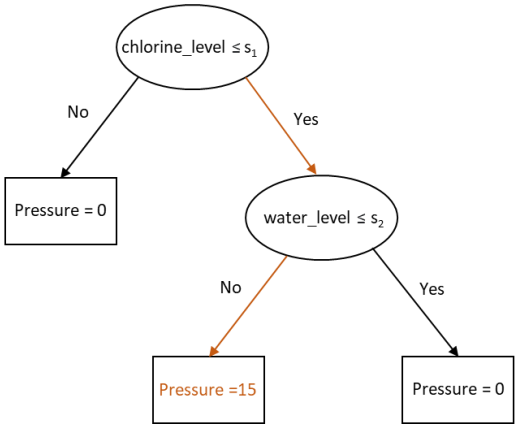


Figure 7.3: Structural representation of the control logic derived from the process description of the example testbed (see Section 7.2.1).

vices, containing the input (process variables) and output (manipulated variables) that need to be connected to the PLC, is prepared. This is followed by developing a structured representation of the program logic to get an initial intuitive understanding of control flow within the process control framework. A *model*, specifying how the process will react to all possible values of the input, defined within the scope of the system, is developed. Finally control program is written by a process engineer, which is but a mapping of process variables to manipulated variables, based on the derived model of the framework.

A structured representation of the functional requirements for regulating the pressure modulating valve is illustrated in Figure 7.3. Process variables gathered from the description include readings from the differential pressure sensor indicating the level of water in the reservoir, and readings from the chlorine sensor. The pressure to be regulated at the outlet of valve constitutes the manipulated variable. Such a representation could be translated to programming languages specified in IEC 61131-3 using *selection* construct IF-THEN-ELSE, *comparison* function LE(), and *logic* function AND() [99, 220]. Figure 7.1 (b) demonstrates the ST implementation of the control flow described in Figure 7.3.

7.3.2 Formalization of Control Programs

In this section, we present an approach towards formalizing control program abstraction, along with a briefing on the basic assumptions. We employ the use case presented in Section 7.2.1 to illustrate the idea central to our formalization.

As discussed beforehand, control program acts as the 'decision-making layer' in a process control framework, and as such, we use *binary decision trees* to develop a structured representation of the defined values of the manipulated variables, for all possible values of the process variables.

Consider the control loop of a process control system, which consists of a process, n sensors, m actuators, and a PLC. The sensors send process measurements $(x_1, \ldots, x_n)$ to the PLC, who in turn generates and sends the manipulated variables $(y_1, \ldots, y_m)$ to the actuators. We assume x_i, y_j to be integers in $[0, 2^\ell)$, for some $\ell \in \mathbb{N}$. *This assumption is reasonable as I/O signals can be both analog and digital.* We assume that the manipulated variables are initialized to zero at the beginning of the program execution, based on the standard initialization process of the commercial PLCs [79]. Also, internal variables and how they interact with the process, are out of the scope of this study. An informal description of our formalization follows:

A control program takes $(x_1, \ldots, x_n)$ as input, and outputs $(y_1, \ldots, y_m)$ based on state trees. For each $j \in [m]$, there exists a state tree Π^j : a binary tree of maximum depth $2n$, where internal nodes associate Boolean functions that compare the process variables against desired *setpoints*, with the outcomes deciding which branch to walk, while terminal nodes output the different values of y_j. Note that, we consider along each path from root to the leaf node, x_i can be compared at most twice, and this is the intuition behind the depth $2n$ for the state trees.

Definition 7.3.1 (Control Program)**.** *Let there be n sensors and m actuators in a process control system, where $n, m \in \mathbb{N}$. Let $(x_1, \ldots, x_n)$ be the sequence of process variables, where x_i represents the value of ith sensor. Let $(y_1, \ldots, y_m)$ be the sequence of manipulated variables, where y_j represents the value of jth actuator. Let x_i and y_j be integers between 0 and $2^\ell - 1$, for some $\ell \in \mathbb{N}$. For every $y_j \in (y_1, \ldots, y_m)$, classification of an input $(x_1, \ldots, x_n)$ is defined as evaluation of the function $\mathcal{C}^j : \mathbb{N}^n \to \{0,1\}^\ell$ based on a state tree Π^j defined as follows:*

A binary tree of maximum depth $2n$, where each internal node v_τ associates a Boolean threshold function $t_\tau : \{0,1\}^\ell \to \{0,1\}$, $(x_i) \mapsto t_\tau(x_i)$, such that $t_\tau(x_i) = 1$ if $x_i \leq s_\tau$, which specifies which branch to walk, where $s_\tau \in [0, 2^\ell)$ denotes the setpoint at node v_τ. At input $(x_1, \ldots, x_n)$, the function iterates from root to the leaf, and the output is the value of the leaf y_j at the end point of this walk in the tree.

A control program $\mathcal{P}$ is the defined as evaluation of the functions $(\mathcal{C}^1, \ldots, \mathcal{C}^m)$ on the process variables $(x_1, \ldots, x_n)$.

To illustrate this, consider the use-case in Section 7.2.1. The function $\mathcal{C}^{pressure}$ takes as input the process variables (*chlorine_level* , *water_level*) and outputs the manipulated variable (*pressure*) based on the state tree $\Pi^{pressure}$.

To understand the intuition behind comparing a process variable twice along a path, consider a scenario where temperature (*temp*) of water in a tank should be within the integer interval $[121, 125]$ for a certain valve to be opened, i.e. the condition (*temp* ≤ 125 and *temp* > 120) needs to hold. To align with our formalism, we write *temp* > 120 as *temp* $! \leq 120$ and compare *temp* with two setpoints along a specific path in Π^{v_2} (interested readers can refer to Figure 7.9).

It is noteworthy to mention that our formalization, in essence, captures all comparison operations defined in the IEC 61131-3 standard [99], with the operators $>$, $<$, $\geq$, $\leq$, $=$ and $\neq$.

7.3.3 Assets of Control Programs

Guided by our objective to prevent the modeled adversary from achieving his goal (see Threat Model in Section 7.2.2), we make an effort to define *assets* of a control program. Informally, assets could be some secret algorithms, cryptographic keys or some data structures within a program that a MATE adversary aims at extracting [178]. Also, the choice of assets depends upon the context and the malicious intent of the adversary [8].

In a process control system, it is important to maintain the correct system state, failure to achieve which could lead to catastrophic consequences. The challenge is to maintain process variables at the desired *setpoints*, in order to meet the production or manufacturing needs [4, 100]. We identify the setpoints to be critical to maintaining a stable state of the system, and are thus central to the operational behaviour of the system. This is our intuition behind defining *setpoints as assets of a control program.*

We do not consider manipulated variables to be secrets of a control program, as by the nature of distributed monitoring and control, manipulated variables can be obtained through other methods [46, 123, 169, 214].

Definition 7.3.2 (Assets of Control Program)**.** *For state tree Π^j, there exists tuple $\mathcal{S}^j = [\![s_1^j, \ldots, s_{2^{2n}}^j]\!]$, where s_τ^j denotes the desired setpoint at node v_τ^j. Asset of a control program $\mathcal{E}_\mathcal{P}$ is defined as the sequence of tuples $(\mathcal{S}^1, \ldots, \mathcal{S}^m)$ for all the state trees defined in a control program $\mathcal{P}$.*

In short, we consider setpoints to be the assets of a control program. Next, we define what it means to *learn* a control program. A MATE adversary aims to reverse-engineer a control program to identify $\mathcal{E}_\mathcal{P}$, and thus unlearnibility means that the adversary with access to the the control program, cannot learn $\mathcal{E}_\mathcal{P}$, except with a negligible probability.

Definition 7.3.3 (Unlearnable Control Programs)**.** *Let $\lambda \in \mathbb{N}$ be the security parameter. A control program $\mathcal{P}$ is unlearnable, if for every polynomial time adversary $\mathcal{A}$ with access to the implementation of $\mathcal{P}$, there exists a negligible function μ, such that for every $\lambda \in \mathbb{N}$:*

$$Pr\left[\mathcal{A}^{\mathcal{P}}(1^\lambda) = \mathcal{E}_\mathcal{P}\right] \leq \mu(\lambda)$$

7.3.4 Composing Attack Vectors

In the following, we discuss *three* scenarios based on the example use-case (see Section 7.2.2) to elaborate on the importance of making control programs unlearnable, as an adversary can leverage his knowledge of the assets to cause fatal consequences. In the given scenarios, we assume that the tank is filled with water during the time of the attack. *To the best of our knowledge, none of the existing works has put forward similar threat models.*

Scenario 1. In this scenario, we assume that the adversary has the ultimate goal of cutting off the water supply to the consumers. The adversary alters the setpoint from s_2 to s_2', where $s_2' > s_2$. The outcome of the attack is that, even when the water level in the tank is significantly high, pressure at the outlet of the valve v is set to 0 and water supply to the consumers is stopped.

Scenario 2. In this scenario, The goal of the adversary is toxicating the water supplied to the consumers, by increasing the concentration of chlorine in the water, which is achieved by changing s_1 to s_1', where $s_1' > s_1$.

Scenario 3. An adversary is simply motivated to perform industrial espionage and wants to learn the control logic of the process control system.

We aim at making a control program unlearnable, so that the adversary cannot extract the process semantics of the target control system.

7.4 Obfuscating Control Programs

In this section, we present our construction that makes use of cryptographic obfuscation to prevent assets of a control program from being *learned* by the modeled adversary (see Section 7.2.2).

7.4.1 Feasible Cryptographic Solutions

As evident from our prior discussion (refer to Section 2.3.1) , the choice of cryptographic solutions to secure assets in a control program is highly restricted by the constrained nature of a process control system. Our threat model allows an adversary to obtain unauthorized access to both network and host (PLC). Such an extreme threat environment, where the software implementation of control program is on an untrusted host, standard cryptographic solutions with encryption algorithms are not adequate. This is because the control program needs to be decrypted and then executed in the PLC. This drives us to seek for cryptographic implementations for such extremely exposed context, that provide practical and achievable security in the face of aforementioned constraints. The properties of program obfuscation (see Definition 2.5.1), where a program is transformed to its semantically equivalent counterpart, such that access to the transformed version does not give away the assets, yet preserves the functionality of the program, is promising in the light of the security guarantees that we want to achieve.

The primary idea behind our construction is to use *cryptographic obfuscation* to transform a control program $\mathcal{P}$ to $\mathcal{O}(\mathcal{P})$ and download it to PLC, such that access to $\mathcal{O}(\mathcal{P})$ does not allow the modeled adversary to learn $\mathcal{E}_{\mathcal{P}}$, yet preserving the functionality.

7.4.2 Setup

We consider a process control system with n sensors, m actuators, a PLC and a process that needs to be controlled. The PLC is connected to the engineering workstation, where control program $\mathcal{P}$ is developed, which is then downloaded to the PLC using ICS configuration network.

The sensors sample process variables and send them to the PLC, where they are stored in *input image table* [33]. PLC evaluates the control program on the process variables by querying the input image table, and outputs manipulated variables, which are then stored in *output image table*. Finally the output signals are sent to actuators to act upon the process.

7.4.3 Building Blocks

As discussed beforehand, we want to hide the setpoints in the state trees of a control program. Each node in a state tree associates a Boolean function describing the inequality $x \leq s$ and our objective is to check whether the input x satisfies the inequality, with access to only an encoded version of s. To achieve this, we use our construction in Chapter 6 for encoding Equation (7.1), where we transform the inequality into a sequence of equalities, such that x satisfies one of the equalities if, $x \leq s$ holds.

$$f(x) = \begin{cases} 1\,, & \text{if } x \leq s \\ 0\,, & \text{otherwise} \end{cases} \tag{7.1}$$

where x and s are integers between 0 and $2^\ell - 1$, for some $\ell \in \mathbb{N}$.

An informal description of our construction is given in the following. For details on formal proofs of correctness and security, interested readers can refer to Chapter 6.

Review of the VBB Obfuscator from Chapter 6. Let $\mathcal{F} = \{f_0, \ldots, f_{\ell-1}\}$ where $f_i(y) : \{0,1\}^\ell \to \{0,1\}^{\ell-i}$ such that $f_i(y) = \lfloor \frac{y}{2^i} \rfloor$. Let $H : \{0,1\}^* \to \{0,1\}^\omega$ be a hash function with $\omega > 2\ell$ such that H is injective on the set of all strings of length less than or equal to ℓ. Let $\mathcal{X} = [0, s+1)$ be an interval containing ℓ bit binary encoding of integers.

Encoding. Compute the hamming weight $k = w(\mathcal{X})$, and partition $\mathcal{X}$ into k disjoint sub-intervals $[a_j, a_j + 2^{p_j})$, such that $a_j = a_{j-1} + 2^{p_{j-1}}$ and $a_1 = 0$, where $p_1, \ldots, p_k$ ($p_1 > p_2 > \cdots > p_k$) denote the bit positions of the 1's in the binary encoding of $w(\mathcal{X})$. Next, calculate $H(f_{p_j}(a_j))$ for every $j \in [k]$ and publish the set of k hashes.

Decoding. For an input $x \in \{0,1\}^{\ell}$, evaluate $H(f_i(x))$, for every $i \in \{0,\ldots,\ell-1\}$ and check if any of the ℓ hashes are included in the set of k hashes published by the encoder.

One of the key observations from the above discussion is that for correct functionality, each node needs to store k hashes, and decoding requires a computation of ℓ hashes. However, it is important to note that such a solution is infeasible for implementing in our setup due to the overhead incurred by the computations. This is our inspiration behind modifying the construction in Chapter 6, such that the solution is lightweight and viable for achieving practical level of protection under the constraints discussed in Chapter 1.

Modifying the Construction in Chapter 6. We do not seek perfect protection or long-term security guarantees, rather our goal is to provide practical level of protection. We rely upon some assumptions, where the *encoding* algorithm (Algorithm 7.1) generates some secret parameters (r, q) and shares them with the *decoding* algorithm (Algorithm 7.2).

We leverage this seeming weakness in the construction (due to such an assumption) to develop a framework for hiding assets in a control program. We now present a high-level survey of our encoding and decoding algorithm, along with their formal descriptions (Algorithm 7.1 and Algorithm 7.2).

The idea is to transform the inequality $x \leq s$ into *one* equality, such that x satisfies the equality, if the condition holds. To achieve this, the encoding procedure (Algorithm 7.1) randomly samples q, where $0 \leq q < \ell$, such that $2^{\alpha} < s < 2^{\alpha+1} \leq 2^{q}$ and publishes the hash $h = H(f_q(2^q - 1))$. We assume that $r = 2^q - 1 - s$ and q is secret shared with the decoding procedure.

Algorithm 7.1 Enc $(s \in [0, 2^{\ell}))$

1: Sample $0 \leq q < \ell$, such that $2^{\alpha} < s < 2^{\alpha+1} \leq 2^{q}$
2: Compute $r = 2^q - 1 - s$
3: Compute $\mu = f_q(2^q - 1)$
4: Compute $h = H(\mu)$
5: **return** h

The decoding procedure (Algorithm 7.2) receives as input $x \in \{0,1\}^\ell$ and the embedded hash value h. It adds r to the input, i.e. $x' = x + r$ and evaluates $g = H(f_q(x'))$. Note that, if $x \leq s$, then $h = g$ always holds.

Algorithm 7.2 Dec (r, q, h)

Input: $\ell \in \mathbb{N}, x \in \{0,1\}^\ell$
Output: 0 or 1.
1: Compute $x' = x + r$
2: Compute $\mu' = f_q(x')$.
3: Compute $g = H(\mu')$.
4: **if** $(g == h)$ **then**
5: **return** 1
6: **else**
7: **return** 0
8: **end if**

Lemma 7.4.1 (Correctness.). *Let $\ell \in \mathbb{N}$. Consider Algorithms* Enc *(Algorithm 7.1),* Dec *(Algorithm 7.2) and input $x \in \{0,1\}^\ell$. Let $H : \{0,1\}^* \to \{0,1\}^\omega$ be injective on the set of all strings of length less than or equal to ℓ, where $\omega > 2\ell$. Then for every integer $s \in [0, 2^\ell)$, for every $(h, r, q) \leftarrow \mathsf{Enc}(s)$, if $x \leq s$ then $\mathsf{Dec}(h, r, q)$ outputs 1, else it outputs 0.*

Proof. From Chapter 6, it is evident that an integer $s \in [0, 2^\ell)$ could be represented in either of the two forms : (a) $s = 2^q - 1$ ($0 \leq q < \ell$), then all integers in the interval $[0, s+1)$ could be encoded with the hash value $H(f_q(2^q - 1))$, and (b) $2^\alpha < s + 1 < 2^{\alpha+1} \leq 2^q$ ($0 \leq q < \ell$), then we encode interval $[0, 2^\alpha)$ with the value $H(f_\alpha(2^\alpha - 1))$, and we encode $[2^\alpha, s+1)$ with k' hash values, where $k' = w(s + 1 - 2^\alpha)$. Thus, the number of hashes generated in Case (b) is $k' + 1$, for some $k' \in \mathbb{N}$.

If $x \leq s$, then for any $r \in \mathbb{N}^+$, $x + r \leq s + r = 2^q - 1$ (equality holds when $x = s$).

The interval $[0, s + r + 1)$ can be encoded with the hash value $H(f_q(2^q - 1))$ as $s + r + 1 = 2^q$. For an $x \in [0, s+1)$, $x' = x + r$ will belong to the interval $[0, 2^q)$ and for an $(h, r, q) \leftarrow \mathsf{Enc}(s)$, $H(f_q(x'))$ will always be equal to h, and the Dec (Algorithm 7.2) will correctly output 1.

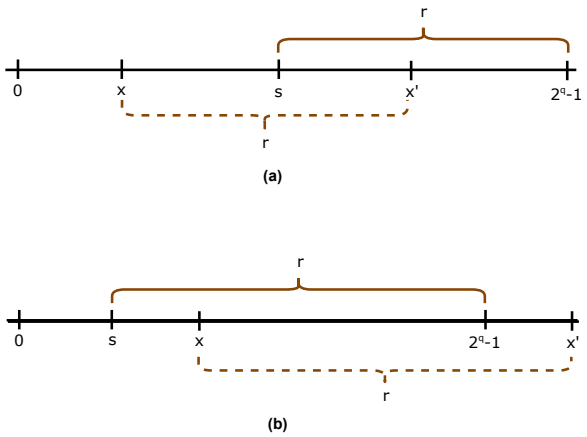


Figure 7.4: Representation of integer $s \in [0, 2^{\ell})$ in the number line: (a) $x \leq s$; (b) $x > s$.

If $x > s$, then $x + r > s + r$, for some $r \in \mathbb{N}^{+}$. This implies $x + r > 2^q$ and $H(f_q(x')) \neq H(f_q(2^q - 1))$, and $\mathsf{Dec}(h, r, q)$ correctly outputs 0. □

Example Parameters. We illustrate the above with a concrete setting: Let $\ell = 8$, then x and s are integers between 0 and $2^8 - 1$. Let $s = 10$. We encode 10 using $\mathsf{Enc}(10)$ and identify whether the condition $x \leq 10$ holds by evaluating Algorithm 7.2 on the encoding. Algorithm 7.1 samples $0 \leq q \leq 8$, such that $2^3 < 10 < 2^4 \leq 2^q$. Let $q = 4$, then $r = 5$ and $\mu = f_4(15)$. The 8- bit binary representation of 15 is 00001111 and $f_4(00001111) = 0000$. Finally $h = H(0000)$ is published by the algorithm.

Let $x = 8$, then $\mathsf{Dec}(5, 4, h)$ (Algorithm 7.2) calculates $x' = 13$. The 8-bit binary representation of 13 is 00001011, and thus $\mu' = f_4(00001011) = 0000 = \mu$. Finally, $H(\mu') = h$, and the algorithm outputs 1. For an $x = 12$, $x' = 17$ and $\mu' = f_4(00010001) = 0001 \neq \mu$, and thus the hashes will not match.

7.4.4 Proposed Platform for Securing Assets in Control Programs

We now put forward the proposed platform that secures assets $\mathcal{E}_{\mathcal{P}}$ in a control program $\mathcal{P}$, from the modeled adversary. In what follows, we implement the building blocks in our setup environment by adding an Encode module and an Obfuscate module to the basic control loop and engineering architecture of a process control system.

Technical Overview. The Obfuscate module is implemented in the Engineering Workstation, which takes as input a control program $\mathcal{P}$ and outputs the obfuscated

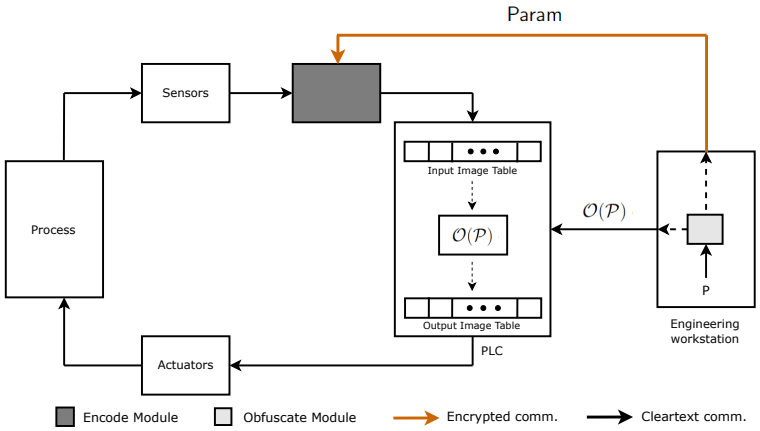


Figure 7.5: Block Diagram of the ObfCP platform. The Obfuscate module is implemented in the Engineering Workstation, and the Encode module is added to the control loop between the sensors and the PLC.

program $\mathcal{O}(\mathcal{P})$ and parameters Param using Algorithm 7.3. $\mathcal{O}(\mathcal{P})$ is downloaded to the PLC over the insecure configuration network, and Param is sent over an encrypted channel to the Encode module, which is installed in the control loop between sensors and PLC. The Encode module captures process variables from the sensors, and based on the embedded data Param, outputs encoded values using Algorithm 7.4 and sends them to the PLC. Note that, the Encode module is implemented in a *Trusted Execution Environment* (TEE) that is available with the modern processors (e.g. Raspberry Pi). Finally, PLC evaluates the obfuscated control program on the encoded inputs, and outputs manipulated variables (see Algorithm 7.5). An overall schematic design of the proposed framework is given in Figure 7.5.

Remark 7.1. *One may wonder, why we cannot push the execution of obfuscated control program $\mathcal{O}(\mathcal{P})$ within the same host used for deploying the* Encode *module, i.e. within the secure enclaves. This could be problematic due to the following:*

(a) *The TEEs do not have the required library support and will require significant engineering effort to execute the real-time tasks, thus increasing the possibility of introducing vulnerabilities and bugs. Also, lack of real-time OS cannot guarantee the responsiveness of the TEEs towards changing inputs* [4].

(b) *Our initial claim that compromising the network module in the control loop is challenging due to the device-dependent nature of the control messages* [196].

Next, we present a description of the Obfuscate module and the Encode module, along with how the obfuscated function is evaluated in the PLC.

Obfuscate Module. The idea behind obfuscating $\mathcal{P}$ is as follows: For each manipulated variable $j \in [m]$, there exists a state tree Π^j, and for each such Π^j, there exists a maximum of 2^{2n} paths from the root (at level 0) to the terminal nodes. If a process variable is compared less than twice along a path, we set the corresponding setpoint(s) to 0 (we call them *dummy nodes*) and allow execution along any one direction.

Let $[\![s_1^\kappa, \ldots, s_{2n}^\kappa]\!]$ denote the $2n$ setpoints along κth path in the tree. Let $\mathcal{Y} = (y^\kappa)_{\kappa \in [2^{2n}]}$ and $\mathcal{Z} = (z^\kappa)_{\kappa \in [2^{2n}]}$, where $y^\kappa \in \{0,1\}^\ell$ denotes label of the terminal node for κth path, and z^κ is an $2n$-bit vector defined as $z_{i_\gamma}^\kappa = 1$ if $x_i \leq s_{i_\gamma}^\kappa$, and 0 otherwise.

For every state tree Π^j and for every $i \in [n]$, $\kappa \in [2^{2n}]$, and $\gamma \in \{0,1\}$, the Obfuscate module sends $\mathcal{O}(\mathcal{P})$ to the PLC, in form of $h_{i_\gamma}^\kappa \leftarrow \mathsf{Enc}(s_{i_\gamma}^\kappa)$, $\mathcal{Y}$ and $\mathcal{Z}$ over insecure ICS network and sends $\mathsf{Param} = (r_{i_\gamma}^\kappa, q_{i_\gamma}^\kappa)_{\gamma \in \{0,1\}, i \in [n], \kappa \in [2^{2n}]}$ to the Encode module over an encrypted channel.

Algorithm 7.3 Obfuscate ($\mathcal{P}$)

Input: $\ell, n, m \in \mathbb{N}, \mathcal{P}$
Output: $\mathcal{O}(\mathcal{P})$, Param

for $j = 1$ to m **do**
  for $\kappa = 1$ to 2^{2n} **do**
    for $i = 1$ to n **do**
      for $\gamma = 1$ to 2 **do**
        return $((h_{i_\gamma}^\kappa)^j, (r_{i_\gamma}^\kappa)^j, (q_{i_\gamma}^\kappa)^j) \leftarrow \mathsf{Enc}((s_{i_\gamma}^\kappa)^j)$
      end for
    end for
    return $(y^\kappa)^j$ {Label of leaf node in κth path }
    return $(z^\kappa)^j$ {$2n$-bit vector }
  end for
end for

Encode Module. This module is assumed to be able to interpret with ICS network protocols and is deployed in between the sensors and the PLC. It captures the process variables $(x_1, \ldots, x_n)$ from the sensors and outputs encoded values using

Algorithm 7.4. Note that the modeled adversary does not have access to the Encode module.

Algorithm 7.4 Encode (with embedded data Param)

Input: $\ell, n, m \in \mathbb{N}, (x_1, \ldots, x_n) \in \mathbb{N}^n$
Output: $((g_{i_\gamma}^{\kappa})^j)_{\gamma \in \{0,1\}, i \in [n], j \in [m], \kappa \in [2^{2n}]}$

```
for j = 1 to m do
  for κ = 1 to 2^{2n} do
    for i = 1 to n do
      for γ = 1 to 2 do
        return (g_{i_γ}^κ)^j) = H(f_{(q_{i_γ}^κ)^j}(x_i + (r_{i_γ}^κ)^j))
      end for
    end for
  end for
end for
```

Obfuscated Program Evaluation. We now discuss how to evaluate the obfuscated program $\mathcal{O}(\mathcal{P})$ on the encoded process variables $((g_{i_\gamma}^{\kappa})^j)_{\gamma \in \{0,1\}, i \in [n], j \in [m], \kappa \in [2^{2n}]}$.

The PLC starts its scan-cycle by reading the encoded process variables from the input image table. For every state tree Π^j, and every path κ in the tree, if $z_{i_\gamma}^k = 1$, the evaluation algorithm checks if $g_{i_\gamma}^{\kappa} = h_{i_\gamma}^{\kappa}$, and if $z_{i_\gamma}^k = 0$, the algorithm checks if $g_{i_\gamma}^{\kappa} \neq h_{i_\gamma}^{\kappa}$. Finally, if all the $2n$ conditions are correct along the path κ, then the algorithm assigns y^{κ} to the jth manipulated variable. The formal description of the same is given in Algorithm 7.5.

Algorithm 7.5 Evaluation (with embedded data $\mathcal{O}(\mathcal{P})$)

Input: $((g_{i_\gamma}^{\kappa})^j)_{\gamma\in\{0,1\},i\in[n],j\in[m],\kappa\in[2^{2n}]}$
Output: $(y_1,\ldots,y_m)$

for $j=1$ to m **do**
  for $\kappa=1$ to 2^{2n} **do**
    for $i=1$ to n **do**
      for $\gamma=1$ to 2 **do**
        if $(z_{i_\gamma}^{k})^j=1$ **AND** $(g_{i_\gamma}^{\kappa})^j == (h_{i_\gamma}^{\kappa})^j$ **then**
          $t=t+1$
        end if
        if $(z_{i_\gamma}^{k})^j=0$ **AND** $(g_{i_\gamma}^{\kappa})^j != (h_{i_\gamma}^{\kappa})^j$ **then**
          $t'=t'+1$
        end if
        if $(t+t')==2n$ **then**
          return $y_j=(y^{\kappa})^j$
        else
          return $y_j=0$
        end if
      end for
    end for
  end for
end for

Remark 7.2. Note that, an Engineering Workstation is usually deployed using a high-end reliable computing platform that supports both ICS and IT protocols. The Encode module could be deployed in a factory settings using a Raspberry Pi (in a TEE) which supports both ICS and IT communication, and *this is our intuition behind using an encrypted channel between the* Obfuscate *module and the* Encode *module.*

7.5 Correctness and Efficiency

We now analyze the correctness and efficiency of the proposed platform.

Lemma 7.5.1 (Correctness.). *Let ℓ, m, $n\in\mathbb{N}$. Consider Algorithms 7.3, 7.4, 7.5 and an input $(x_i)_{i\in[n]}$, where $x_i\in\{0,1\}^{\ell}$. Then for every $(\mathcal{O}(\mathcal{P})$, Param$)\leftarrow$ Obfuscate$(\mathcal{P})$, for every $((g_{i_\gamma}^{\kappa})^j)_{i\in[n],j\in[m],\kappa\in[2^n],\gamma\in\{0,1\}}\leftarrow$ Encode(Param), Algorithm 7.5 correctly outputs $(y_j)_{j\in[m]}$.*

Proof. Algorithm 7.3 calculates $((h_{i_\gamma}^{\kappa})^j,(r_{i_\gamma}^{\kappa})^j,(q_{i_\gamma}^{\kappa})^j)\leftarrow$ Enc $((s_{i_\gamma}^{\kappa})^j)$, for every $j\in[m]$, $k\in[2^{2n}]$, $i\in[n],\gamma\in\{0,1\}$. For an input $(x_i)_{i\in[n]}$, Algorithm 7.4

calculates $(g_{i_\gamma}^{\kappa})^j$ (using steps 1, 2, 3 of Algorithm 7.2). From Lemma 7.4.1, it is evident that, for every integer $s \in \{0,1\}^\ell$, for every $(h, r, q) \leftarrow \mathsf{Enc}(s)$, and for every $g \leftarrow \mathsf{Dec}(h, r, q)$, $g = h$ if $x \leq s$ holds.

Thus, for every path κ of the state tree Π^j, and every $2n$-bit vector $(z^{\kappa})^j$, $(g_i^{\kappa})^j$ will always be equal to $(h_i^{\kappa})^j$, for $(z_i^{\kappa})^j = 1$ and $(g_i^{\kappa})^j \neq (h_i^{\kappa})^j$, otherwise, and Algorithm 7.5 will correctly output y_j. □

Complexity Analysis. We now analyze the efficiency of the proposed construction. As given in Definition 7.3.1, state tree is a binary tree of maximum depth $2n$. Evaluation at each internal node is of order $\mathrm{O}(\ell)$. Given a manipulated variable is set to zero at the beginning of program execution, we are interested in paths, where terminal nodes are assigned non-zero values; we call them *accepting paths*. Each accepting path requires a total of $2n\ell$ computations. Note that, a "sufficiently" large number of accepting paths would lead to computations incurring exponential complexity. Thus, it remains to determine the conditions that make our construction efficient.

Lemma 7.5.2 (Polynomial Slowdown)**.** *Let $\lambda \in \mathbb{N}$ be the security parameter of the system, and let n, ℓ, ω be polynomials in λ. Let T_λ be a special family of control programs where number of accepting paths are of order $\mathrm{O}(n \log \ell)$. Then, for every $\mathcal{P} \leftarrow T_\lambda$, there exists a polynomial p, such that the running time of $\mathcal{O}(\mathcal{P})$ is bounded by $p(|P|, \lambda)$.*

To get a rough approximation on the number of accepting paths in state trees corresponding to real PLC control programs, we examined programs available in public code repositories corresponding to varied sectors of industrial automation. Note that, our study of control programs is limited to the publicly available codes, due to restrictions in accessing a fully functioning ICS facility. Our analysis indicates that a majority of the control programs could be efficiently obfuscated using our scheme. For example, an ICS application to control the speed of an AC motor using proximity sensors, would form a state tree as given in Figure 7.6. For $\ell = 64$, the proposed ObfCP platform would efficiently obfuscate the control application, while achieving computational security.

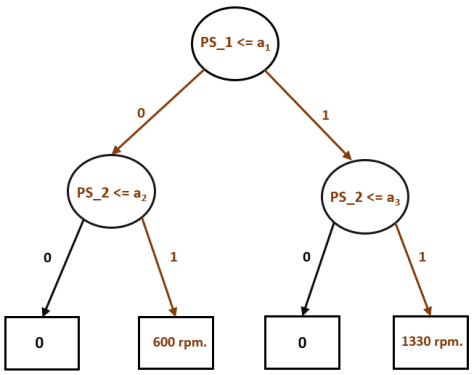


Figure 7.6: State tree corresponding to real-world ICS application on controlling speed of a motor using proximity sensors PS_1 and PS _2.

7.6 Security Analysis

The goal of control program obfuscation is to hide $\mathcal{E}_P$ of a control program P from the modeled adversary. We now present the security analysis of the proposed ObfCP platform.

Lemma 7.6.1. *Let λ, n, m, ℓ and T_λ satisfy the conditions in Lemma 7.5.2. Then for every $P \leftarrow T_\lambda$, there is no brute-force attack on $\mathcal{O}(P)$ to learn $\mathcal{E}_P$.*

Proof. For a control program P, $\mathcal{O}(P)$ consists of hashed encodings $(h^\kappa_{i_\gamma})^j$ corresponding to setpoints $(s^\kappa_{i_\gamma})^j$, where $\gamma \in \{0,1\}, i \in [n], j \in [m], \kappa \in [2^{2n}]$. Given the hashed encodings are of the form $H(0), H(00), \ldots, H(00\ldots0)(\ell$ times), the adversary can compute the $\ell + 1$ hash values, compare them with $(h^\kappa_{i_\gamma})^j$ and learn $(q^\kappa_{i_\gamma})^j$. Since $(s^\kappa_{i_\gamma})^j < (q^\kappa_{i_\gamma})^j$, the adversary can identify the upper bound (if $z^k_{i_\gamma} = 1$) or lower bound (if $z^k_{i_\gamma} = 0$) of $s^k_{i_\gamma}$. However, since $(r^\kappa_{i_\gamma})^j$ is hidden from the adversary's view, he cannot learn $(s^\kappa_{i_\gamma})^j$ as any value that satisfies the upper or lower bound, could also have led to the same $(h^\kappa_{i_\gamma})^j$. Thus there is no brute-force attack on $\mathcal{O}(P)$.

7.7 Prototype Implementation

In this section, we provide a proof-of-concept of the proposed ObfCP platform over *two* realistic real-time ICS platforms, and an in-depth analysis of our empirical evaluations.

7.7.1 Experimentation Setup

We implemented a prototype of the proposed ObfCP platform using a simulation setup in a Raspberry Pi 3 Model B [169]. The Obfuscate and Encode modules are developed using Python 2.7.7 on a device with an Intel(R) Core(TM) i7-9750H CPU rated at 2.60 GHz (see Table 7.2 for specification details). We designed the ST codes describing the *obfuscated control programs* corresponding to the example testbeds in Codesys v3.5 (script engine 4.0.0.0). We used the SHA-256 and MD5 implementations (available in Python 2.7.7) from the hashlib cryptographic library [93] to generate the respective hash values for the encodings. We note that our implementation using Raspberry Pi and Codesys serves as a good proof-of-concept and can be extended with other suitable hardware modules, without loss of generality. A summary of the system configurations and implementation details are given in Table 7.1 and Table 7.2.

Table 7.1: Implementation Details of Raspberry Pi (as PLC)

Artifact	**Configuration**
Platform	Raspberry Pi 3 Model B
Hardware	BCM2835
Revision	a22082
Processor	ARMv7 Processor rev 4 (v7l)
Operating System	Raspbian GNU Linux 10 (buster)

7.7.2 Experiments over ICS platforms

We make use of *two* realistic real-time ICS platforms as case studies to demonstrate the efficacy of ObfCP: (1) Water Distribution System, and (2) Industrial Steam Boiler Application. The first case study requires *comparing each process variable against a desired setpoint*. While on the contrary, the second case study imposes an additional requirement; it necessitates *checking whether a process variable belongs to a particular integer interval*. We limit ourselves to the above two applications, as the control programs describing their operational behavior cover all the standard "selection and comparison" functions described in IEC 61131-3 standard [99], and thus should suffice to demonstrate the significance of our approach. Our construction applies to *both analog and digital control signals* used in industrial control environment, and employs *bit-wise comparisons* and *bit-shift functions* specified by the IEC standard, thus making the proposed construction available for generic ICS applications.

Table 7.2: System configurations while implementing Encode and Obfuscate Module

Artifact	Configuration
Device name	MSI
Installed RAM	8.00 GB (7.85 GB usable)
Device ID	B5E83B2B-14EC-4832-BB29-6FCF5EBE9798
Product ID	00325-81471-80538-AAOEM @ 2.60GHz
Processor	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
System type	64-bit operating system, x64-based processor

Case Study # 1 - Water Distribution System

Our first experimental testbed is introduced earlier in Section 7.2.1. Let `PLC_PRG` be the control program describing the functional requirements of the water distribution system. Then, following Definition 7.3.1, `PLC_PRG` takes as input the process variables (*chlorine_level* , *water_level*), and outputs the manipulated variable (*pressure*). Note that the setpoints corresponding to chlorine and water levels are instantiated following the concrete settings in real-world applications. The ST code for the control program is given as follows:

```
PROGRAM PLC_PRG                          (Case Study #1)
VAR
        chlorine_level : INT;
        water_level : INT;
        pressure : INT;
END_VAR

IF (chlorine_level <= 5 and water_level > 35) THEN
    pressure := 15;
ELSE
    pressure := 0;
END_IF
```

Obfuscate Module: For the program `PLC_PRG`, this module generates Param and $\mathcal{O}$(`PLC_PRG`).

Figure 7.7 shows the optimized state tree $\Pi^{pressure}$ (without adding the dummy nodes). Since none of the process variables are compared twice, we conduct the

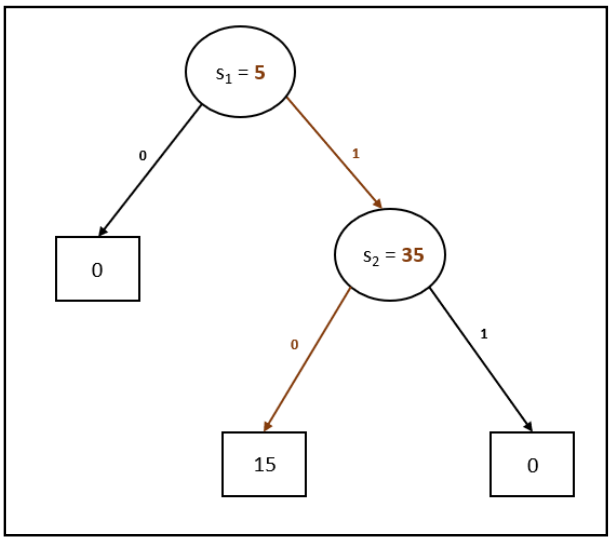


Figure 7.7: State Tree $\Pi^{pressure}$ for **Case Study # 1**.

experiment with the optimized tree as this approach would lead to better efficiency. The leaf nodes $y^{\kappa} = 0$ for $\kappa = \{1, 2, 3\}$. For $\kappa = 4$, $y^{\kappa} = 15$ and $z^{\kappa} = 11$, with the corresponding *setpoints* $[\![5, 35]\!]$. Instantiating $\ell = 16$, Algorithm 7.3 generates Param and $\mathcal{O}$(PLC_PRG) as follows:

$$\mathsf{Param} = \{(q_1^4 = 5, r_1^4 = 26), (q_2^4 = 6, r_2^4 = 28)\}$$

The q and r values are generated using $\mathsf{Enc}(s_1)$ and $\mathsf{Enc}(s_2)$ (see Algorithm 7.1), where $s_1 = 5$ and $s_2 = 35$. The obfuscated encodings generated by the module are: $h_1^4 =$ *chlorine_level_ob* and $h_2^4 =$ *water_level_ob*, where the values are generated with MD5 and SHA-256. The obfuscated control program $\mathcal{O}$(PLC_PRG) is downloaded to the Raspberry Pi (used as a PLC).

The obfuscated encodings generated by the module are as follows: $h_1^4 =$ chlorine_level_ob and $h_2^4 =$ water_level_ob.

where `chlorine_level_ob = 'c6f057b86584942e415435ffb1fa93d4'` and `water_level_ob = 'b4b147bc522828731f1a016bfa72c073'` denote the obfuscated encodings, corresponding to the setpoints `5` and `35` respectively (generated using Algorithm 7.1).

The obfuscated control program $\mathcal{O}$(`PLC_PRG`) is downloaded to the Raspberry Pi (used as a PLC) using an ethernet cat 6 cable. *The intuition is that if* `(chlorine_level <= 5 and water_level <= 35)` *in* `PLC_PRG` *holds, then* `(chlorine_level == chlorine_level_ob and water_level == water_level_ob)` *in* $\mathcal{O}$(`PLC_PRG`) *will always be true.* In what follows, we write the ST code corresponding to the obfuscated control program.

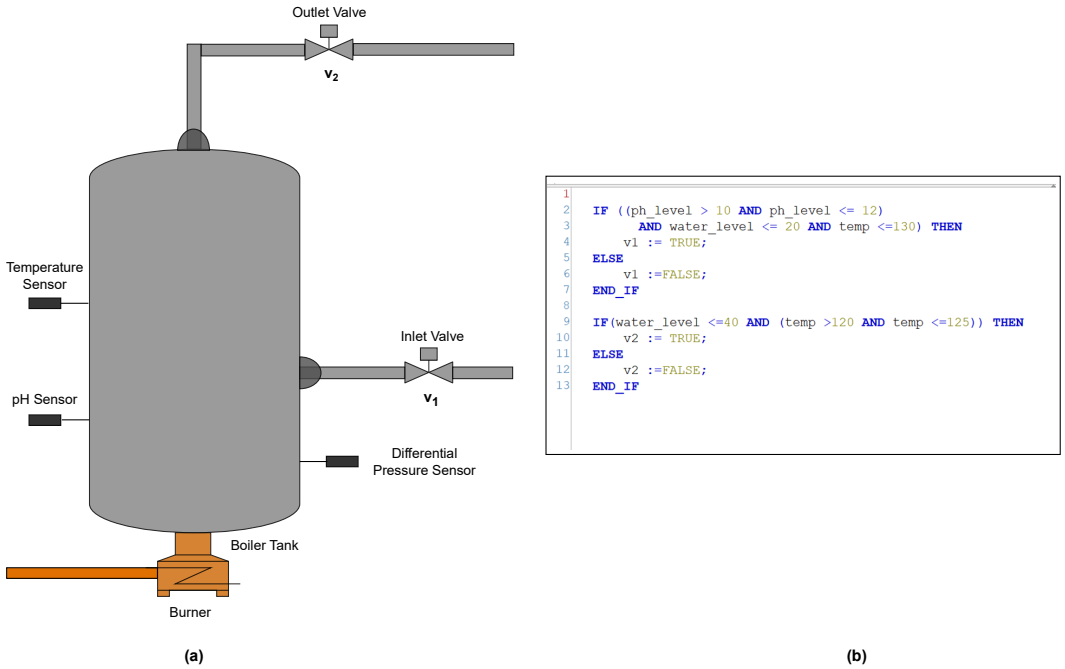
```
PROGRAM O(PLC_PRG)                    (Case Study #1)
VAR
        chlorine_level : INT;
        water_level : INT;
        pressure : INT;
        chlorine_level_ob : STRING(INT#65);
        water_level_ob : STRING(INT#65);
END_VAR

IF (chlorine_level == chlorine_level_ob and
     water_level <> water_level_ob) THEN
    pressure := 15;
ELSE
    pressure := 0;
END_IF
```

Encode Module: We execute Algorithm 7.4 corresponding to 100 individual trials. In particular, we generate test values corresponding to *chlorine_level* as `[13, 11, 5, 3, 15, 12, 1, 8, 4, 2]` and *water_level* as `[30, 23, 45, 10, 35, 50, 20, 40, 28, 37]`, and execute 10 runs each. The encodings generated are sent to the Raspberry Pi over ethernet adapter and compared to `chlorine_level_ob` and `water_level_ob` in the obfuscated control program $\mathcal{O}$(`PLC_PRG`).

Case Study # 2 - Industrial Steam Boiler Application

We design a simple realistic steam boiler application, which is key to critical industrial utilities such as thermal power plants, chemical manufacturing, etc.




```
1
2   IF ((ph_level > 10 AND ph_level <= 12)
3         AND water_level <= 20 AND temp <=130) THEN
4       v1 := TRUE;
5   ELSE
6       v1 :=FALSE;
7   END_IF
8
9   IF(water_level <=40 AND (temp >120 AND temp <=125)) THEN
10      v2 := TRUE;
11  ELSE
12      v2 :=FALSE;
13  END_IF
```

Figure 7.8: Experimental Testbed: (a) Schematics of an Industrial Steam Boiler Application (b) Control Program written in Structured Text describing the operational behavior of the application.

As evident from Figure 7.8, a water tank is connected to a feed-water valve v_1, which supplies chemically treated cool water to the tank, and a burner which heats the water in the tank to its boiling point. When water vaporizes, the steam outlet valve v_2 releases the steam from the tank. The tank is connected to a temperature sensor, which measures the temperature of water in the tank, a pH sensor, which checks the pH level of the chemically treated water and a differential pressure sensor which determines the water level in the tank. Mechanics of how the burner is operated, and how the pH level is maintained is out of the scope of this study. The functional requirements of the control application are specified in Figure 7.8(b).

As indicated in the description of the application, the process variables are (*pH_level*, *water_level*, *temp*) and manipulated variables are (v_1, v_2).

The ST code for this application is given as follows:

```
PROGRAM PLC_PRG          (Case Study #2)
VAR
```

```
        pH_level : INT;
        water_level : INT;
        temp : INT;
        v1 : BOOL;
        v2 : BOOL;
END_VAR
-------------------------------------------------------
/* Functional requirement for controlling valve v1 */

IF (pH_level > 10 and pH_level <= 12 and
    water_level <= 20 and temp <= 130) THEN
    v1 := TRUE;
ELSE
    v1 := FALSE;
END_IF
-------------------------------------------------------
/* Functional requirement for controlling valve v2 */

IF (water_level <= 40 and temp > 120 and temp <= 125)
THEN
    v2 := TRUE;
ELSE
    v2 := FALSE;
END_IF
```

An important observation to make is that the collection of inequalities `pH_level > 10` and `pH_level <= 12` is equivalent to comparing whether `pH_level` belongs to the integer interval `[11,12]`. Similarly, `temp > 120` and `temp <= 125` is equivalent to checking whether `temp` belongs to `[121, 125]`. In other words, the control program `PLC_PRG` checks whether process variables belong to particular integer intervals for actuating the manipulated variables.

Given that the building blocks for the proposed ObfCP platform is developed upon inequalities of the form $x \leq s$ (see Section 7.4.3), we transform $x > s$ to the form $x \not\leq s$ (x not less than or equal to s). In particular, we transform `pH_level > 10` to the form `pH_level !<= 10` and `temp > 120` to the form `temp !<=`

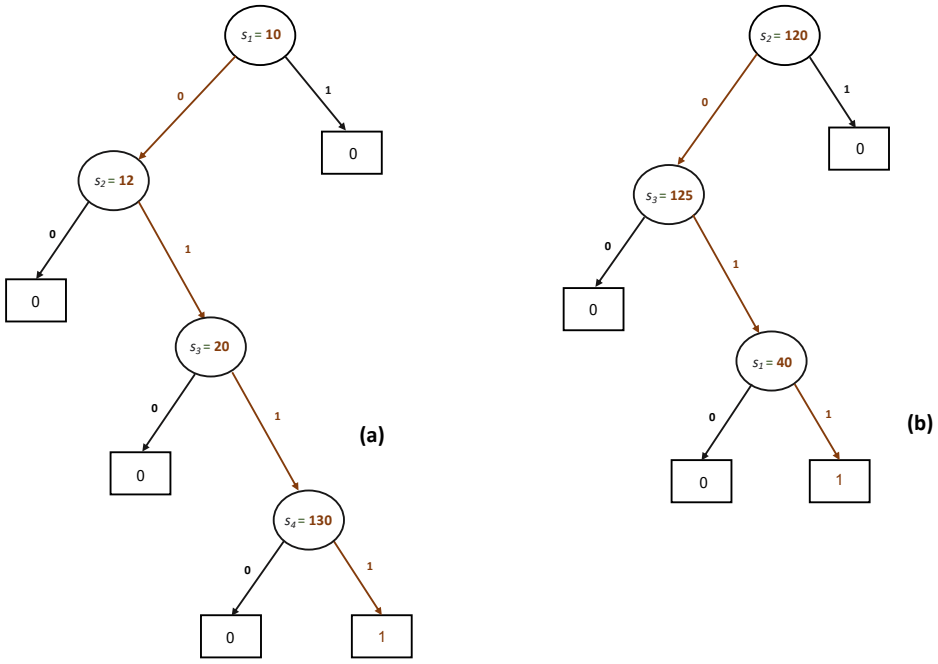


Figure 7.9: Optimized state trees corresponding to the manipulated variables v_1 and v_2: (a) Π^{v_1} and (b) Π^{v_2} for Case Study # 2.

120. Figure 7.9 demonstrates the state trees Π^{v_1}, and Π^{v_2} for the control program PLC_PRG corresponding to the industrial steam boiler application.

Obfuscate Module: For the control program PLC_PRG, the Obfuscate module outputs Param and $\mathcal{O}$(PLC_PRG). As clear from Figure 7.9, the state tree Π^{v_1} has 16 paths. For $\kappa = 8$, $y^\kappa = 1$ and $z^\kappa = 0111$, with the corresponding *setpoints* $[\![10, 12, 20, 130]\!]$. Similarly, state tree Π^{v_2} has 8 paths, and for $\kappa = 4$, $y^\kappa = 1$ and $z^\kappa = 011$, with the corresponding *setpoints* as $[\![120, 125, 40]\!]$. Instantiating $\ell = 16$, Algorithm 7.3 generates Param (see Table 7.3) and obfuscated encodings (see Figure 7.10).

Table 7.3: Experimentation Parameters for Case Study # 2

q-value	$(q_1^8)^1$	$(q_2^8)^1$	$(q_3^8)^1$	$(q_4^8)^1$	$(q_1^4)^2$	$(q_2^4)^2$	$(q_3^4)^2$
	5	5	7	8	6	8	7
r-value	$(r_1^8)^1$	$(r_2^8)^1$	$(r_3^8)^1$	$(r_4^8)^1$	$(r_1^4)^2$	$(r_2^4)^2$	$(r_3^4)^2$
	21	19	107	125	23	130	7

The obfuscated control program is given as follows:

```
PROGRAM O(PLC_PRG)                                    (Case Study #2)
VAR
        ph_level_c1: INT;
        ph_level_c1_ob: STRING(INT#65);
        ph_level_c2: INT;
        ph_level_c2_ob: STRING(INT#65);

        water_level_c1: INT;
        water_level_c1_ob: STRING(INT#65);

        temp_c1: INT;
        temp_c1_ob: STRING(INT#65);

        water_level_c2: INT;
        water_level_c2_ob: STRING(INT#65);

        temp_c2: INT;
        temp_c2_ob: STRING(INT#65);

        temp_c3: INT;
        temp_c3_ob: STRING(INT#65);

        v1: BOOL;
        v2: BOOL;

END_VAR
------------------------------------------------------------
/* Functional requirement for controlling valve v1 */

IF  (pH_level_c1 <> ph_level_c1_ob  and
     ph_level_c2 = ph_level_c2_ob and
     water_level_c1 = water_level_c1_ob and
     temp_c1 = temp_c1_ob)
THEN
    v1 := TRUE;
```

```
ELSE
    v1 :=FALSE;
END_IF
------------------------------------------------------
/* Functional requirement for controlling valve v2 */

IF (water_level_c2 = water_level_c2_ob and
    temp_c2 = temp_c2_ob and
    temp_c3 <>temp_c3_ob)
THEN
    v2 := TRUE;
ELSE
    v2 :=FALSE;
END_IF
```

Encode Module: To make it analogous to the methodologies adopted for the previous case study, we execute Algorithm 7.4 corresponding to 100 individual trials.

7.7.3 Performance Analysis

We now provide an in-depth analysis of feasibility and scalability of the proposed construction. The goal of our empirical evaluation is to study the performance impacts and runtime overheads due to ObfCP in order to estimate the trade-offs between security and real-time requirements of industrial control applications. In particular, we compare the execution times for scan cycles with and without implementing the obfuscation platform to identify the delay introduced due to ObfCP (this includes the time required for running the Encode module (see Algorithm 7.1) and obfuscated control program evaluation (see Algorithm 7.5) at the Raspberry Pi). Furthermore, we benchmark the performance of ObfCP when implemented individually with MD5 and SHA-256. We note that, using lightweight hashing algorithms over the standard cryptographic implementations used in our experiments, the presented numbers are expected to improve.

Table 7.4 shows a comparison of the scan cycle times for unobfuscated control program `PLC_PRG` and obfuscated program $\mathcal{O}$(`PLC_PRG`) corresponding to the

```
                                                MD5 implementation
******************************
------------------------------
Obfuscating values [10, 12, 20, 130, 40, 125, 120]
Current values of  ph_level_c1_ob '645a8aca5a5b84527c57ee2f153f1946'
Current values of  ph_level_c2_ob '645a8aca5a5b84527c57ee2f153f1946'
Current values of  water_level_c1_ob '4c93008615c2d041e33ebac605d14b5b'
Current values of  temp_c1_ob 'dd4b21e9ef71e1291183a46b913ae6f2'
Current values of  water_level_c2_ob 'f1b708bba17f1ce948dc979f4d7092bc'
Current values of  temp_c2_ob 'dd4b21e9ef71e1291183a46b913ae6f2'
Current values of  temp_c3_ob '4c93008615c2d041e33ebac605d14b5b'
------------------------------


                                                SHA-256 implementation
******************************
------------------------------
Obfuscating values [10, 12, 20, 130, 40, 125, 120]
Current values of  ph_level_c1_ob '9c9f57efe04f8f5a65b699db1c777238d5876b44cef240654c749dd09e1790ef'
Current values of  ph_level_c2_ob '9c9f57efe04f8f5a65b699db1c777238d5876b44cef240654c749dd09e1790ef'
Current values of  water_level_c1_ob 'f120bb5698d520c5691b6d603a00bfd662d13bf177a04571f9d10c0745dfa2a5'
Current values of  temp_c1_ob '7e071fd9b023ed8f18458a73613a0834f6220bd5cc50357ba3493c6040a9ea8c'
Current values of  water_level_c2_ob '84d9c4b849506b6d8f8075a9000e7e0a254be71060ea889fad3c88395988f4fc'
Current values of  temp_c2_ob '7e071fd9b023ed8f18458a73613a0834f6220bd5cc50357ba3493c6040a9ea8c'
Current values of  temp_c3_ob 'f120bb5698d520c5691b6d603a00bfd662d13bf177a04571f9d10c0745dfa2a5'
------------------------------
```

Figure 7.10: Obfuscated encodings generated by Algorithm 7.3 using both MD5 and SHA-256 implementations from the hashlib library.

case studies. We note that the scan cycle time with the obfuscated control program is the average of execution times using MD5 and SHA-256. Elaborating on the task instances, we conduct 100 trials, with 10 test values generated for each of the process variables and execute 10 runs each (approximated by Experiment IDs).

Our results indicate that the proposed ObfCP adds an overall delay of 20% (approx) for Case Study # 1 and 14% (approx) for Case Study # 2 to the scan cycle time. For instance, a single task requires 17.1 milliseconds (scan cycle time), while with the proposed implementation it takes 17.8 milliseconds.

In Figure 7.11, we compare the running time of ObfCP with independent implementations of MD5 and SHA-256. An interesting takeout is that SHA-256 outperforms MD5, introducing less than 5.8% (approx) delay for Case Study 1 and 2.4% (approx) delay for Case Study 2.

We have used $\ell = 16$ for conducting the experiments as the micro-benchmarking helps envision how the proposed platform works. However, *to increase the adversary's cost of attack, the recommended value of* ℓ *is* 64. We note that, the delay introduced due to ObfCP could be unacceptable for some ICS applications (e.g. turbo-machinery, magnetic suspension systems allowing up to 1 millisecond delay), which invariably implies choosing lower values of ℓ for minimizing complexity penalties, nevertheless lowering the security guarantees. Hence, for increasing adversary's cost of attack, the engineering workstation could execute the Obfuscate module (Algorithm 7.5) with new Param values at regular/random intervals.

Table 7.4: ObfCP Performance Evaluation.

Exp. ID	Scan Cycle Time (in ms.)			
	Case Study # 1		Case Study # 2	
	PLC_PRG	$\mathcal{O}$(PLC_PRG)	PLC_PRG	$\mathcal{O}$(PLC_PRG)
0	17.9	15.6	18.2	22.2
1	13.4	16.3	18.4	19.5
2	10	16.8	16.6	21.2
3	13.3	15.5	17.3	18.2
4	17.1	18.1	17.8	18.7
5	13.3	16.6	15.9	21.4
6	14.3	20	19.6	19.3
7	13.9	14.6	15.7	19.1
8	11.0	16.1	16.8	20.7
9	13.1	15.4	18.4	20

We remind the readers that industrial control infrastructures are time-invariant, with deviations in terms of added delay leading to fatal consequences, where the *maximum permissible delay* directly proportional to the response rate of the field devices. Let Δ be the maximum permissible delay of a control infrastructure, and let δ be overall execution time of a scan cycle with the proposed ObfCP implementation, then $\delta \leq \Delta$ should hold in order to maintain the stability of the system.

7.8 Discussion and Conclusion

The extreme threat environment with real-time requirements, along with the potential consequences quite naturally point at the limits to the practical and achievable security, and thus is our aim to make use of cryptography viable under the constraints of critical control infrastructures (see discussion in Section 1.1). Our work can be considered as the *first attempt to secure the operational semantics of an industrial control infrastructure* from a polynomial adversary, who attempts to reverse engineer a recovered implementation of control program through the threat surface specified in Section 7.2.2. We have introduced a novel legacy-compliant platform to secure assets in a control program, by making use of cryptographic obfuscation. We have considered a strong and practical threat model, where the software implementation of a control program is on an untrusted host, such that use of encryption algorithms fails in achieving the desired security

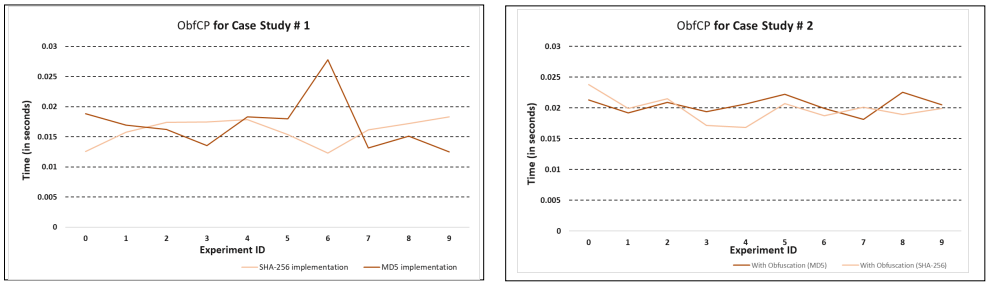


Figure 7.11: Performance benchmark of ObfCP using MD5 and SHA-256 implementations for the example testbeds.

goals. Our threat model along with the defined attack scenarios (see Section 7.3.4) has not been introduced before, to the best of our knowledge.

Chapter 8

Towards Verifiability of Cryptographic Obfuscators

In this chapter, we introduce a new variant of malicious obfuscation. Our formalism is incomparable to the existing definitions by Canetti and Varia (TCC 2010), Canetti *et al.* (EUROCRYPT 2022) and Badrinarayanan *et al.* (ASIACRYPT 2016). We show that this concept is natural and applicable to obfuscation-as-a-service platforms. We next define a new notion called *verifiable obfuscation* which provides security against malicious obfuscation. We demonstrate undetectable malicious obfuscators for some obfuscation schemes in the theoretical literature, and follow this with a general approach to verifiable obfuscation. Our exclusive goal behind pursuing this direction is *two-fold: (1) provide security against malicious control program obfuscation, and (2) prove the existence of malicious obfuscators in literature, leading to rigorous foundations of black hat obfuscation, and subsequent constructions that provide security against such obfuscators, which is a subject of independent interest and has much wider applicability in malware obfuscation* [219].

8.1 Rationale

As discussed in Chapter 2, program obfuscation is widely used in black hat scenarios, where obfuscators hide malicious behavior of software programs. There have been instances where obfuscation concealed malicious Android APKs from evading detection by anti-virus scanners. In this chapter, we look into another interesting problem where obfuscators inject malicious functionality in software programs. This issue has not received attention in the past, to the best of the authors' knowledge.

In the interest of establishing the usefulness and significance of the developed notions, we explain the following scenario: Consider an app-store that distributes software programs, and wants to maintain a reputation as a trusted supplier of good apps. Due to intellectual property reasons, the app-store may be selling obfuscated programs, for which it employs third-party tools/platforms (such as Tigress [54]). This leads to the theoretical possibility of malicious obfuscators that cause the obfuscated programs to have malicious functionality. Of note, we do not imply that the current tools/platforms are malicious. The malicious behaviour could range from simple errors that were not present in the original program, to inserting a master backdoor that allows access to the program when running on user devices. This leads to a compelling and general question: *how can the app-store know that these programs are not malicious for users?* Surprisingly, the cryptographic community has neglected the study of such obfuscators, *which is the central focus of this chapter.*

Note that the app-store knows the intended behaviour of the program, so one might think that they could easily verify the obfuscated program by simply running the program on some chosen test inputs. However, the task of checking correctness of a program can often be much more complex than this. Also, as we will show in our examples, there are situations where a malicious obfuscator can introduce a secret hard-to-guess master password that unlocks every program, and there is no way for the app-store to be able to guess such a password. Hence, black-box testing the obfuscated code might not be enough to solve the problem.

We consider obfuscation of evasive programs (a random program from the program-family evaluates to 0 with outstanding probability for a fixed input). An obfuscator is supposed to have correctness, meaning the good inputs are still accepted and the others are rejected. Our main conceptual contribution is that malicious obfuscators do not satisfy the same correctness, but are indistinguishable from honest obfuscators. Defending against such malicious obfuscators by ensuring correctness of obfuscation is the goal of this study. In particular, we require a verification property that proves that the obfuscated program satisfies the same correctness as output of the honest obfuscator; we call them *verifiable obfuscators.*

Our Contributions. On the whole, we initialize a theoretical investigation towards a new variant of verifiability in obfuscators. In what follows, we summarize the main contributions of this work.

- The first contribution of this work is introducing a new and powerful notion of *malicious obfuscators*, that inject malicious functionality in the program in a way that is undetectable even by the app-store. We argue that this concept is natural, since most app-stores use third party obfuscation tools/platforms [223].
- Our next contribution is in formulating the definitional framework of *verifiability in obfuscators*, which provides security against malicious obfuscators. Our formalism builds on existing security definitions of obfuscation, while adding a verifiability property. As a proof of concept, we provide a general approach to verifiable obfuscation, however this proof of concept is not very efficient as it essentially requires the verifier to re-do the obfuscation process. *We leave as an open problem for future work the development of more efficient methods.*
- Finally, we demonstrate malicious obfuscators for several well-known theoretical obfuscation proposals with strong security guarantees. In particular, we show malicious obfuscators for the conjunction obfuscators by Bishop, Kowalczyk, Malkin, Pastro, Raykova and Shi [30] and Bartesuk, Lepoint, Ma and Zhandry [27].

Organization. This chapter is organized into the following sections. Section 8.2 discusses the prior work in verifiable obfuscation. Section 8.3 introduces the formal definitions of malicious obfuscation and verifiable obfuscation. Section 8.4 demonstrates undetectable malicious obfuscators for the conjunction obfuscator by Bishop, Kowalczyk, Malkin, Pastro, Raykova and Shi [30]. Section 8.5 demonstrates undetectable malicious obfuscators for the conjunction obfuscator by Bartesuk, Lepoint, Ma and Zhandry [27]. Section 8.6 presents a broad-level discussion on verifiable control program obfuscation. Section 8.7 provides an overall conclusion.

8.2 Prior Work

In this section, we discuss the efforts by the cryptographic community in designing obfuscators that allow verifying the correctness of obfuscation.

The notion of verifiable obfuscation was introduced by Canetti and Varia [43]. They mention the significance of a *verifiability* property in proving that an obfuscator does not inject any malicious scripts in the program. Our approach to verifiable

obfuscation allows the app-store to verify the obfuscation process, as long as the app-store knows the original unobfuscated source code. This is because our verification concept requires verifying the steps taken by the obfuscator to convert the original source into the obfuscated code. This is where we deviate from [43], which do not require a verifier to know the un-obfuscated source.

In [228], Zobernig, Galbraith and Russello present a construction for verifying whether a seemingly opaque predicate is triggered by a secret input known to the obfuscator such that some potentially malicious code gets activated. Their verification algorithm relies upon cryptographic hash functions, and provides an appealing intuitive towards furthering research in formalizing definitional framework for verifiability in obfuscators and subsequent efficient constructions for verifiable obfuscators.

Badrinarayanan, Goyal, Jain and Sahai [21] consider the case of a malicious obfuscator and aim to provide assurance to the user running the protocol, by allowing the user to check a predicate on the program. They give a solution for iO that triples the overhead (the user has to run three versions of the obfuscated program).

Canetti, Chakraborty, Khurana, Kumar, Poburinnaya and Prabhakaran [44] follow a similar notion where they use a perfectly binding commitment that attests some property of the unobfuscated circuit. The obfuscator returns a semi-functional circuit allowing the user to run a verification procedure and derive a fully-functional obfuscated circuit that satisfies the attested property using subexponentially-secure iO.

To the best of our knowledge, none of the existing notions of verifiability consider verifying the correctness of obfuscation, when provided with the original unobfuscated program. Additionally, we believe we are the first to point out undetectable malicious obfuscators for schemes in the theoretical literature, which is in fact our main technical contribution.

8.3 Notions of Verifiability in Obfuscators

In this section, we present background definitions for obfuscation from the literature, and introduce the definitional framework of malicious obfuscators. Following this, we formally define verifiable obfuscation. We use the circuit model for program, although the programs will be written in pseudo-code.

Ideally, obfuscation entails *perfect* correctness, where the obfuscated program $\tilde{C}$ computes the exact same function as C. While this is an ambitious goal, a weaker yet practical variation allows $\tilde{C}$ to approximate C with overwhelming probability *over the coin tosses of the obfuscator* [186]. A number of subsequent studies [3, 28, 106] impose different variations of approximate correctness. Bitansky and Paneth [31] introduce a weaker variant, where $\tilde{C}$ is allowed to err on *poly* many inputs, with high enough probability over inputs drawn from some distribution.

We recall the definition of distributional VBB obfuscator [24], with *three* variations of approximate functionality requirements prevalent in the obfuscation literature, which we call ϕ_1, ϕ_2 and ϕ_3.

Definition 8.3.1 (Distributional Virtual Black-Box Obfuscator (DVBB) [24] [23])**.** *Let $\lambda \in \mathbb{N}$ be the security parameter. Let $\mathcal{C} = \{\mathcal{C}_\lambda\}$ be a family of polynomial-size programs parameterized by inputs of length $n(\lambda)$, and let $\mathcal{D} = \{\mathcal{D}_\lambda\}$ be a class of distribution ensembles, where $\mathcal{D}_\lambda$ is a distribution over $\mathcal{C}_\lambda$. A PPT algorithm $\mathcal{O}$ for the family $\mathcal{C}$ and the distribution $\mathcal{D}$ with correctness ϕ_i should satisfy the correctness condition ϕ_i:*

- *Correctness (ϕ) :*
  - ϕ_1 : *For every $\lambda \in \mathbb{N}$ and for every $C \in \mathcal{C}_\lambda$, and for every $\tilde{C} = \mathcal{O}(C)$*

    $$\forall x \in \{0,1\}^{n(\lambda)} : \ \tilde{C}(x) = C(x)$$

  - ϕ_2 : *For every $\lambda \in \mathbb{N}$ and every $C \in \mathcal{C}_\lambda$, there exists a negligible function $\mu(\lambda)$, such that:*

    $$\Pr_{\mathcal{O}} \ [\, \forall x \in \{0,1\}^{n(\lambda)} : \ \tilde{C}(x) = C(x) \,] > 1 - \mu(\lambda)$$

    where the probability is over the coin tosses of $\mathcal{O}$ in computing $\tilde{C} = \mathcal{O}(C)$. This is formulated in Definition 3.1 of [202].

  - ϕ_3 : *For every $\lambda \in \mathbb{N}$, every $C \in \mathcal{C}_\lambda$, and for every $x \in \{0,1\}^{n(\lambda)}$ to C, there exists a negligible function $\mu(\lambda)$, such that:*

    $$\Pr_{\mathcal{O}} \ [\, \tilde{C}(x) = C(x) \,] > 1 - \mu(\lambda)$$

    where the probability is over the coin tosses of $\mathcal{O}$. This is called "weak functionality preservation" in [27].

- *Polynomial Slowdown : For every $\lambda \in \mathbb{N}$ and for every $C \in \mathcal{C}_\lambda$, there exists a polynomial q such that the running time of $\tilde{C} = \mathcal{O}(C)$ is bounded by $q\,(|C|)$, where $|C|$ denotes the size of the program.*
- *Virtual Black-box : For every (non-uniform) polynomial size adversary $\mathcal{A}$, there exists a (non-uniform) polynomial size simulator $\mathcal{S}$ with oracle access to C, such that for every distribution $D \in \mathcal{D}_\lambda$:*

$$\left| \Pr_{C \leftarrow \mathcal{D}_\lambda, \mathcal{O}, \mathcal{A}} [\mathcal{A}(\mathcal{O}(C)) = 1] - \Pr_{C \leftarrow \mathcal{D}_\lambda, \mathcal{S}} [\mathcal{S}^C(1^\lambda) = 1] \right| \leq \mu(\lambda)$$

where $\mu(\lambda)$ is a negligible function.

8.3.1 Defining Malicious Obfuscators

Our formalism of malicious obfuscators is that the output program does not have the same correctness guarantee as the honestly obfuscated program. For example, suppose the malicious obfuscator inserts a master backdoor y that is accepted by every program it obfuscates. Then correctness ϕ_1 and ϕ_2 are not possible (a program is never correct on all inputs) and correctness ϕ_3 is also not possible (for the specific input y, we have $C(y) = 0$ but $\tilde{C}(y) = 1$).

In addition, we require that a polynomial distinguisher cannot detect the fact that $\tilde{C}$ is maliciously generated, either by inspecting the source code of $\tilde{C}$, running it on chosen inputs, or both.

Given an obfuscation tool is used by many users to obfuscate many programs, we require a malicious obfuscator to be undetectable even when used to obfuscate many programs. Therefore our security definition allows the distinguisher to receive obfuscations of polynomially many chosen programs C, including repeated obfuscations of the same program. We do this by providing oracle access to the obfuscator. We also consider the case where the malicious obfuscator may be introducing a master backdoor that is the same for every execution. This is modeled in our formalism as a fixed auxiliary input aux (that is used for all executions) which represents some randomly generated secret data that is known to the malicious obfuscator, such as the value of a master backdoor.

Definition 8.3.2 (Malicious Obfuscation)**.** *Let λ, n satisfy the conditions as given in Definition 8.3.1. For any family of programs $\mathcal{C}$ and distribution class $\mathcal{D}$ over $\mathcal{C}$, let $\mathcal{O}$ be an obfuscator that satisfies the conditions given in Definition 8.3.1 with*

correctness ϕ_i. Then a malicious obfuscator for the family $\mathcal{C}$ and distribution $\mathcal{D}$ is a PPT algorithm $\mathcal{A}$ that takes an auxiliary input $\mathsf{aux} \in \{0,1\}^\lambda$*, such that:*

- *(Correctness violation) : For any choice of* aux*, $\mathcal{A}(1^\lambda, \cdot, \mathsf{aux})$ does not satisfy ϕ_i.*
- *(Indistinguishability) : There exists a negligible function $\mu(\lambda)$, such that for every PPT distinguisher $\mathcal{B}$ that has oracle access to an obfuscator (so can adaptively ask for obfuscations of polynomially many adaptively chosen $P \in \mathcal{P}_\lambda$)*

$$\left| \Pr_{\mathsf{aux},\mathcal{B},\mathcal{A}} [\, \mathcal{B}^{\mathcal{A}(1^\lambda,\cdot,\mathsf{aux})} = 1] - \Pr_{\mathcal{B},\mathcal{O}} [\, \mathcal{B}^{\mathcal{O}(1^\lambda,\cdot)} = 1] \right| \leq \mu(\lambda)$$

where the first probability is taken over the choice of aux *and the coin tosses of $\mathcal{B}$, $\mathcal{A}$, and the second probability is taken over the coin tosses of $\mathcal{B}$, $\mathcal{O}$.*

A distinguisher can request and receive many obfuscations of C. We write $\mathcal{B}^{\mathcal{O}(\cdot)}$ to indicate that $\mathcal{B}$ has oracle access to $\mathcal{O}$ that can be queried on any C (but with respect to fixed aux in the malicious case). The distinguisher has to decide if it is interacting with the honest obfuscator or a malicious one.

8.3.2 Formalizing Verifiability in Obfuscators

We now present the notion of verifiable obfuscation. Verifiability is to defend against malicious obfuscation. Furthermore, we restrict to an efficient verifier who has a priori knowledge of the program being obfuscated. Crucially, we require the verifiable obfuscation scheme to inherit the properties given in Definition 8.3.1, while incorporating a verifiability property that proves the correctness of obfuscation.

At a high-level, *verifiable obfuscation* ($\mathcal{VO}$) is a two-step process: The first step is carried out by an obfuscator, who runs an efficient algorithm $\mathcal{VO}.\mathsf{Obf}$, that takes as input a program C, and outputs the obfuscated program $\tilde{C}$ along with a proof π. Informally, π allows the verifier (app-store) in determining the correctness of the obfuscation.

The second step is performed by a verifier, who knows C. Precisely, $\mathcal{VO}.\mathsf{Verify}$ is an efficient algorithm, such that $\mathcal{VO}.\mathsf{Verify}\ (1^\lambda, C, \tilde{C}, \pi) = 1$, if $\tilde{C}$ is a correct obfuscation of C. We consider an honest obfuscator $\mathcal{O}$ with correctness ϕ_i and we require: (i) if $\tilde{C}$ satisfies ϕ_i, then $\mathcal{VO}.\mathsf{Verify}\ (1^\lambda, C, \tilde{C}, \pi) = 1$, and (ii) given a purported obfuscation $(\tilde{C}, \pi) \leftarrow \mathcal{A}(1^\lambda, C)$, if $\mathcal{VO}.\mathsf{Verify}\ (1^\lambda, C, \tilde{C}, \pi) = 1$, then $\tilde{C}$ satisfies ϕ_i.

We now present the formal definition of verifiable obfuscation.

Definition 8.3.3 (Verifiable Obfuscation)**.** *Let $\lambda \in \mathbb{N}$ be the security parameter of the system. For any family of polynomial-size programs $\mathcal{C} = \{\mathcal{C}_\lambda\}$ and distribution ensembles $\mathcal{D} = \{\mathcal{D}_\lambda\}$, let $\mathcal{O}$ be an honest obfuscator that satisfies the conditions as given in Definition 8.3.1 with correctness $\phi_i \in \phi$. Then verifiable obfuscation $\mathcal{VO}$ for the family $\mathcal{C}$ and distribution class $\mathcal{D}$ is a pair of PPT algorithms $(\mathcal{VO}.\mathsf{Obf}, \mathcal{VO}.\mathsf{Verify})$ such that for every $\lambda \in \mathbb{N}$ and every $C \in \mathcal{C}_\lambda$ and every ϕ_i:*

- *For every $(\tilde{C}, \pi) \leftarrow \mathcal{VO}.\mathsf{Obf}\ (1^\lambda, C)$, $\mathcal{VO}.\mathsf{Verify}\ (1^\lambda, C, \tilde{C}, \pi) \rightarrow \{0,1\}$.*
- *(Soundness) : For every PPT adversary $\mathcal{A}$, if $[(\tilde{C}, \pi) \leftarrow \mathcal{A}\ (1^\lambda, C)] \wedge [1 \leftarrow \mathcal{VO}.\mathsf{Verify}\ (1^\lambda, C, \tilde{C}, \pi)]$, then $\tilde{C}$ satisfies ϕ_i.*

We now show that some of the published obfuscation schemes could be leveraged to inject malicious functionality.

8.4 Reviewing the [30] Construction

In this section, we review the construction by Bishop *et al.* [30] for obfuscating conjunctions (alternatively called *pattern matching with wildcards*). We first recall the definition of conjunctions.

Definition 8.4.1 (Conjunctions)**.** *Let $n \in \mathbb{N}$ and let $\mathsf{pat} \in \{0,1,\star\}^n$ be a pattern, where $\star$ is a wildcard character. Let $W = \{i : \mathsf{pat}_i = \star\}$ be the set of wildcard positions in pat. A conjunction function $C : \{0,1\}^n \rightarrow \{0,1\}$, $x \mapsto C(x)$ on an input $x \in \{0,1\}^n$ is defined as*

$$C(x) = \begin{cases} 1\ , & \text{if } \forall\ \mathsf{pat}_i \neq \star \wedge x_i = \mathsf{pat}_i \\ 0\ , & \text{otherwise.} \end{cases}$$

8.4.1 The [30] Obfuscation Scheme

Bishop *et al.* designed an efficient DVBB obfuscator for conjunction functions using Lagrange interpolation, while proving its security in the generic group model. In particular, they assume that the discrete log problem is hard in a group G of order q. Hence we need $q > 2^{2\lambda}$ where λ is the security parameter. Their security goal roughly states that, a PPT adversary cannot distinguish the obfuscation of C from obfuscation of a function that always outputs 0.

The high-level overview of their construction is as follows: to obfuscate a pattern $\mathsf{pat} \in \{0,1,\star\}^n$ with $|W|$ many wildcards, define a degree $n-1$ polynomial $F(t) = \sum_{k=1}^{n-1} a_k t^k \in \mathbb{F}_q[t]$, with $F(0) = 0$. The coefficients $a_1, \ldots, a_{n-1}$ are sampled uniformly random in $\mathbb{F}_q$, where q is exponential in n. If the ith bit of the pattern is j, where $j \in \{0,1\}$ or if $\mathsf{pat}_i = \star$, then evaluate the polynomial at $2i + j$, otherwise sample a uniformly random element from $\mathbb{F}_q$. The final step is to publish the $2n$ field elements in the exponent of a group $\mathbb{G} = \langle g \rangle$. The formal description of the obfuscator is given in Algorithm 8.1. Note that, sampling $C \leftarrow \mathcal{C}$ is equivalent to choosing $\mathsf{pat} \leftarrow \mathcal{D}$, and hence we assume that it is easy to determine pat from C.

Algorithm 8.1 Obfuscator $\mathcal{O}_{Con}(1^\lambda, C)$

Input: $n = n(\lambda)$
Output: $(h_{i,j})_{i \in [n], j \in \{0,1\}}$

Sample large prime $q > 2^{2\lambda}$
Select $G = \langle g \rangle$ of order q
Sample $(a_1, \ldots, a_{n-1}) \xleftarrow{R} \mathbb{F}_q$ uniformly
Let $F(t) = a_1 t^1 + \cdots + a_{n-1} t^{n-1}$
for $i = 1$ to n **do**
  for $j = 0$ to 1 **do**
    if $(\mathsf{pat}_i = \star \vee \mathsf{pat}_i = j)$ **then**
      $h_{i,j} \leftarrow g^{F(2i+j)}$
    else
      $h_{i,j} \xleftarrow{R} \mathbb{F}_q$ uniformly
    end if
  end for
end for
return $(h_{i,j})$

Interpolating the polynomial in the exponent with n Lagrange's coefficients corresponding to a correct input $x \in \{0,1\}^n$ gives g^0, and the evaluation algorithm correctly accepts the input. For an input that does not match the pattern, a uniformly random field element in the exponent is returned by the algorithm, and the evaluator correctly rejects the input with overwhelming probability. The procedure is formally described in Algorithm 8.2.

Algorithm 8.2 Evaluation Eval (with $2n$ embedded values $(h_{i,j})_{i\in[n],j\in[2]}$)

Input: $n \in \mathbb{N}, x \in \{0,1\}^n$
Output: 0 or 1.

1: **for** $i = 1$ to n **do**
2: $\quad \gamma_i \coloneqq \prod_{j\neq i} \frac{2j-x_j}{2i-x_i-x_j+2j}$
3: **end for**
4: Compute $T \coloneqq \prod_{i=0}^{n-1}(h_{i,x_i})^{\gamma_i}$
5: **if** $T = g^0$ **then**
6: $\quad$ **return** 1
7: **else**
8: $\quad$ **return** 0
9: **end if**

The above construction satisfies 'approximate' *correctness* (ϕ_3 in Definition 8.3.1), which at high-level requires the obfuscation to be correct for some fixed C and x, with overwhelming probability over randomness of the obfuscator.

8.4.2 Designing Malicious Obfuscator for [30]

A conjunction function C for a pattern $\mathsf{pat} \in \{0,1,\star\}^n$ with $|W|$ many wildcard characters has $2^{|W|}$ accepting inputs. Our goal is to design a malicious obfuscator that accepts a certain input string that does not match the pattern. Furthermore, we require that any poly-time distinguisher $\mathcal{B}$ with a priori knowledge on pat, cannot distinguish between honest and purported obfuscation instances.

Constructing such an obfuscator involves many subtleties. Firstly, it has to accept the $2^{|W|}$ inputs strings that correctly match the pattern. This means that for every $i \in W$ or $\mathsf{pat}_i = j$, the $h_{i,j}$s should be correctly structured elements. Given we require the obfuscator to accept bad inputs, a naïve solution would be to output $h_{i,j} \leftarrow g^{F(2i+j)}$ for every $i \in [n]$ and $j \in \{0,1\}$, as the obfuscator would then accept all strings of length n. Such malicious behaviour can be detected by the distinguisher.

Similarly, one might expect to find a solution by allowing the obfuscator to output $h_{i,(1-j)} \leftarrow g^{F(2i+(1-j))}$ for some $i \notin W$. This essentially turns pat_i into a wildcard. However, a PPT distinguisher who knows pat can simply flip the $n - |W|$ non-wildcard bits one by one, and check if any of the inputs are accepted, to detect the malicious behaviour in $\mathrm{O}(n)$ time. Hence it is easy to detect this malicious

behaviour. Thus it remains to construct an un-detectable malicious obfuscator around the apparent constraints

To design a malicious obfuscator, we start with a string $y \in \{0,1\}^n$ that does not satisfy the pattern, and so $C(y) = 0$. Given y is fixed and independent of C, it might well be the case that y is an accepting input. In such case, there is no malicious behavior and the original obfuscation works. So we start with a string $y \in \{0,1\}^n$ such that $C(y) = 0$ and y does not match the pattern in at least *two* positions. Note that such a string would be correctly rejected with overwhelming probability by the [30] construction. We require the malicious obfuscator to agree with the honest obfuscator *except for the input* y.

To achieve this, we choose a random degree $n-1$ polynomial $F(t)$ following Algorithm 8.1, while we define another polynomial $H(t) \in \mathbb{F}_q[t]$, such that $H(0) = 0$, and $H(2i + y_i) = F(2i + y_i)$, if either $\mathsf{pat}_i = \star$ or $\mathsf{pat}_i = y_i$. Since y is a bad input and there exists at least two positions where it does not match the pattern, there are at most $n - 1$ conditions on H. Hence it is possible to choose a degree $n - 1$ polynomial $H(t) \in \mathbb{F}_q[t]$ such that $H \neq F$.

Our justification behind not allowing y to differ by only one position is that this would imply $H = F$, which would be equivalent to turning the position to a wildcard. In this case it is impossible to have an obfuscated program that accepts y but that can't be detected by a distinguisher who checks the wildcard positions.Requiring that H is not equal to F means that y is accepted but other strings close to y are probably not accepted.

Next, we compute $h_{i,j} = g^{F(2i+j)}$, if $\mathsf{pat}_i = \star$ or $\mathsf{pat}_i = j$, for every $i \in [n]$, for $j \in \{0,1\}$. Else, if $y_i = j$ then $h_{i,j} = g^{H(2i+y_i)}$, else $h_{i,j}$ is sampled randomly from $\mathbb{F}_q$. This ensures that the newly designed obfuscator accepts correct inputs, along with y. The formal description follows in Algorithm 8.3.

Algorithm 8.3 Malicious Obfuscator $\mathcal{A}_{Con}(1^\lambda, C, y \in \{0,1\}^n$ s.t. $C(y) = 0$ and $y_i \neq \mathsf{pat}_i$ for at least two non-wildcard positions)

Input: $n = n(\lambda)$
Output: $(h^*_{i,j})_{i\in[n],j\in\{0,1\}}$

1: Sample large prime $q > 2^{2\lambda}$
2: Select $G = \langle g \rangle$ of order q
3: **if** $(C(y) == 1)$ **then**
4: **return** $\mathcal{O}_{Con}(1^\lambda, C)$
5: **else**
6: Sample $(a_1, \ldots, a_{n-1}) \xleftarrow{R} \mathbb{F}_q$ uniformly
7: Let $F(t) = a_1 t^1 + \cdots + a_{n-1} t^{n-1}$
8: Randomly choose $n-1$ polynomial $H(t) \in \mathbb{F}_q[t]$ s.t. $H(0) = 0$, $H \neq F$, and $H(2i + y_i) = F(2i + y_i)$ if $\mathsf{pat}_i = \star$ or $\mathsf{pat}_i = y_i$ for all $i \in [n]$, $j \in \{0,1\}$
9: **for** $i = 1$ to n **do**
10: **for** $j = 0$ to 1 **do**
11: **if** $(\mathsf{pat}_i = \star \vee \mathsf{pat}_i = j)$ **then**
12: $h^*_{i,j} \leftarrow g^{F(2i+j)}$
13: **else**
14: **if** $y_i = j$ **then**
15: $h^*_{i,j} \leftarrow g^{H(2i+y_i)}$
16: **end if**
17: **else**
18: $h^*_{i,j} \xleftarrow{R} \mathbb{F}_q$ uniformly
19: **end if**
20: **end for**
21: **end for**
22: **end if**
23: **return** $(h^*_{i,j})$

Theorem 8.4.1. *$\mathcal{A}_{Con}$ violates ϕ_3.*

Proof. As stated before, Bishop's construction [30] satisfies correctness ϕ_3. Concretely, for a conjunction function C defined by a pattern pat sampled from $\{0, 1, \star\}^n$, Algorithm 8.1 satisfies the following for every $x \in \{0,1\}^n$:

$$\Pr_{\mathcal{O}_{Con},C}\left[\, \mathcal{O}_{Con}(1^\lambda, C)(x) = C(x) \,\right] > 1 - \mu(\lambda)$$

where $\mu(\lambda)$ is a negligible function in λ. Determining whether $\mathcal{A}_{Con}$ violates ϕ_3 is equivalent to finding whether $\mathcal{A}_{Con}(1^\lambda, C, y)$ gives noticeable advantage in accepting an input that does not satisfy the pattern pat.

Fix an input string $y \in \{0,1\}^n$. Then

$$\Pr_{\mathcal{O}_{Con},C}\left[\, \mathcal{O}_{Con}(1^\lambda, C)(y) = C(y) \,\right] > 1 - \mu(\lambda)$$

Consider the correctness ϕ_3 experiment for $\mathcal{A}_{Con}$ on the fixed y. The experiment samples C from $\mathcal{C}_\lambda$, which implies sampling pat from $\mathcal{D}_\lambda$. Since $\mathcal{D}_\lambda$ is a distribution on evasive conjunction functions, $C(y) = 0$ with overwhelming probability in λ. However, we need to augment A_{Con} so that it can also handle the case when $C(y) = 1$. We explain this next.

If $C(y) = 1$, then set $\mathcal{A}_{Con}(1^\lambda, C, y) = \mathcal{O}_{Con}(1^\lambda, C)$. If y differs from pat in only *one* position $\mathsf{pat}_i \neq y_i$, then set $\mathcal{A}_{Con}(1^\lambda, C, y)$ to $\mathcal{O}(1^\lambda, C^*)$, where C^* is the circuit for the pattern pat^*, where $\mathsf{pat}^*_k = \mathsf{pat}_k$ for all $k \neq i$ and $\mathsf{pat}^*_i = \star$.

If y differs from pat in two positions or more, then executing Algorithm 8.3 on input y returns correctly structured elements of the polynomial H for all the positions where $y_i \neq \mathsf{pat}_i$, while for the positions where $\mathsf{pat}_i = y_i$, the algorithm returns elements of F. This ensures that evaluating the obfuscated program on y returns 1.

Since $\mathcal{A}_{con}(1^\lambda, C, y)(y) = 1$ for all C, we have

$$\Pr_{\mathcal{A}_{Con},C}\left[\, \mathcal{A}_{Con}(1^\lambda, C, y)(y) \neq C(y) \,\right] > 1 - \mu(\lambda)$$

where $\mu(\lambda)$ is negligible in λ. This proves that $\mathcal{A}_{Con}$ violates ϕ_3. □

8.4.3 Indistinguishability of Obfuscators

In the previous section, we have proved that $\mathcal{A}_{Con}(1^\lambda, C, y)$ does not satisfy ϕ_3. In this section, we give an 'informal'explanation of indistinguishability of honest and malicious obfuscators in the generic group model, which generally implies that the malicious obfuscator is distributional VBB secure under the same conditions as in [30].

We consider a distinguisher $\mathcal{B}$ that can request polynomially many obfuscated programs from the obfuscation oracle, and determine whether it is interacting with an honest or malicious obfuscator. We remind the readers that $\mathcal{B}$ knows pat.

Theorem 8.4.2. *Let $\lambda \in \mathbb{N}$ be the security parameter and let n, w be polynomials in λ with $w = n - \omega(\log n)$. Let $\mathbb{G}$ be a group of prime order $q > 2^{2\lambda}$. Then for all PPT distinguishers $\mathcal{B}$ in the generic group model, there exists a negligible function $\mu(\lambda)$, such that for all $\lambda \in \mathbb{N}$, the following holds:*

$$\left| \Pr_{y,\mathcal{B},\mathcal{A}_{Con}} [\, \mathcal{B}^{\mathcal{A}_{Con}(1^\lambda,\cdot,y)} = 1] - \Pr_{\mathcal{B},\mathcal{O}_{Con}} [\, \mathcal{B}^{\mathcal{O}_{Con}(1^\lambda,\cdot)} = 1] \right| \leq \mu(\lambda)$$

Informal argument. We consider distinguisher $\mathcal{B}$ can make polynomially many adaptive queries to the obfuscation oracle, including repeated obfuscations of the same circuit. For simplicity, we assume that all the queries are with respect to the same parameters n, $|W|$ and q. The malicious input $y \in \{0,1\}^n$ represents the choice of aux in Definition 8.3.2. Since there are polynomially many chosen C, each corresponding to some evasive pattern pat, y does not satisfy any of the patterns with overwhelming probability.

For $\mathcal{B}$ to distinguish between $\mathcal{A}_{Con}(1^\lambda, \cdot, y)$ and $\mathcal{O}_{Con}(1^\lambda, \cdot)$, $\mathcal{B}$ has to determine the input y which is accepted which is accepted by the obfuscated programs, for all choices of C. But $\mathcal{B}$ does not know y and recovering F is hard due to the assumed hardness of discrete log, thus determining H is hard (for each choice of C). Since there are exponentially many choices for F and inputs that do not match pat, $\mathcal{B}$ cannot distinguish between the malicious and honest obfuscation with overwhelming probability in λ.

□

8.4.4 Verifiable Obfuscator for [30]

Our objective is to design a verifiable obfuscation scheme that provides security against the malicious obfuscator described in Section 8.4.2. More specifically, we require the verifiable scheme to provide a proof that the obfuscated program satisfies ϕ_3.

For a $y \in \{0,1\}^n$, where $y_i \neq \mathsf{pat}_i$ at m positions ($2 \leq m \leq n-1$), $\mathcal{A}_{Con}(1^\lambda, C, y)$ outputs m specially chosen group elements depending on the polynomial H, instead of following the honest computations and returning uniformly random group elements (Step 10 of Algorithm 8.1).

Therefore, to prove that the obfuscated program is not maliciously constructed, we require the verifiable obfuscator to show that for all $i \in [n]$ and $j \in \{0,1\}$ if $\mathsf{pat}_i \neq \star$ and $\mathsf{pat}_i \neq j$, $h_{i,j}$'s are uniformly random in $\mathbb{F}_q$.

To achieve this, we define a pseudo-random generator $\mathsf{PRG}_{\mathsf{seed}} : \{0,1\}^{\lceil \log 2n \rceil} \to \mathbb{F}_q$ indexed by seed $\mathsf{seed} \in \mathbb{F}_q$. We construct the $n-1$ degree polynomial $F(t)$ with coefficients generated using $\mathsf{PRG}_{\mathsf{seed}}$. Next, instead of sampling uniformly random field elements, the verifiable obfuscator generates $h_{i,j}$ using the $\mathsf{PRG}_{\mathsf{seed}}$. Finally, the verifiable obfuscator outputs $(h_{i,j})$ and the proof seed. The formal description follows in Algorithm 8.5.

Algorithm 8.4 Obfuscator $\mathcal{VO}_{Con}(1^\lambda, C)$

Input: $n = n(\lambda)$
Output: $(\hat{h}_{i,j})_{i\in[n],j\in\{0,1\}}$, seed

Sample large prime $q > 2^{2\lambda}$
Select $G = \langle g \rangle$ of order q
Sample $\mathsf{seed} \xleftarrow{R} \mathbb{F}_q$
for $k = 1$ to $n-1$ **do**
  $a_k \leftarrow \mathsf{PRG}_{\mathsf{seed}}(k)$
end for
Let $F(t) = a_1 t^1 + \cdots + a_{n-1} t^{n-1}$
for $i = 1$ to n **do**
  for $j = 0$ to 1 **do**
    if $(\mathsf{pat}_i = \star \vee \mathsf{pat}_i = j)$ **then**
      $\hat{h}_{i,j} \leftarrow g^{F(2i+j)}$
    else
      $\hat{h}_{i,j} \leftarrow \mathsf{PRG}_{\mathsf{seed}}(n+i-1)$
    end if
  end for
end for
return $(\hat{h}_{i,j}), \mathsf{seed}$

We now give the description of $\mathcal{VO}_{Con}.\mathsf{Verify}$, which takes as input the pattern pat and $((\hat{h}_{i,j})_{i\in[n],j\in\{0,1\}}, \mathsf{seed}) \leftarrow \mathcal{VO}_{Con}(1^\lambda, C)$ and outputs 1 if the obfuscation is correct. Note that, $\mathcal{VO}_{Con}(1^\lambda, C)$ is deterministic once the seed is chosen and so the verifier simply re-runs the obfuscation and checks whether the output is equal to the obfuscated program.

Algorithm 8.5 $\mathcal{VO}_{Con}$.Verify (with embedded values $(\hat{h}_{i,j})_{i\in[n],j\in\{0,1\}}$, seed)

Input: $n = n(\lambda)$, $\mathsf{pat} \in \{0,1,\star\}^n$
Output: 1 or $\perp$

for $k = 1$ to $n-1$ **do**
  $a'_k \leftarrow \mathsf{PRG}_{\mathsf{seed}}(k)$
end for
Let $F'(t) = a'_1 t^1 + \cdots + a'_{n-1} t^{n-1}$
for $i = 1$ to n **do**
  for $j = 0$ to 1 **do**
    if $(\mathsf{pat}_i = \star \vee \mathsf{pat}_i = j)$ **then**
      $b_{i,j} \leftarrow g^{F'(2i+j)}$
      if $\hat{h}_{i,j} \neq b_{i,j}$ **then**
        return $\perp$; EXIT ()
      end if
    else
      $b'_{i,j} \leftarrow \mathsf{PRG}_{\mathsf{seed}}(n+i-1)$
      if $\hat{h}_{i,j} \neq b'_{i,j}$ **then**
        return $\perp$; EXIT ()
      end if
    end if
  end for
end for
return 1

Theorem 8.4.3. *Let* PRG *be indistinguishable from uniform. Then for every PPT adversary* $\mathcal{A}$ *and every* $((h_{i,j}), \mathsf{seed}) \leftarrow \mathcal{A}(1^\lambda, C, y)$*, if* $\mathcal{VO}_{Con}.\mathsf{Verify}(1^\lambda, \mathsf{pat}, (h_{i,j}), \mathsf{seed}) = 1$*, then* $(h_{i,j})$ *satisfies* ϕ_3*.*

Proof. Towards a contradiction, we suppose the obfuscated program $(h_{i,j})$ does not satisfy ϕ_3, and there exists a $y \in \{0,1\}^n$ such that the following holds:

$$\Pr_{\mathcal{A},C}\left[\, \mathcal{A}(1^\lambda, C, y)(y) \neq C(y) \,\right] > \epsilon(\lambda)$$

where $\epsilon(\lambda) = \lambda^{-\mathrm{O}(1)}$ is a noticeable function in λ.

Since the output of $\mathcal{A}$ is accepted by $\mathcal{VO}_{Con}$.Verify, this implies that the same output could also have been generated by the honest obfuscator $\mathcal{O}$ for some choice of seed. But, $\mathcal{A}$ could have tried at most polynomially many seeds, say $p(\lambda)$, which

means the honest obfuscator could also have generated the same output as $\mathcal{A}$ with probability at least $\frac{1}{p(\lambda)}$. This implies that running $\mathcal{O}$ once yields

$$\Pr_{\mathcal{O},C}\left[\,\mathcal{O}(1^\lambda, C)(y) \neq C(y)\,\right] > \frac{\epsilon(\lambda)}{p(\lambda)}$$

which is still a noticeable function in λ. This contradicts the result of Bishop *et al.* [30] that for every input, $\mathcal{O}(1^\lambda, C)$ is correct with overwhelming probability over the coin tosses of the obfuscator and the program (see Definition 8.3.1). □

8.5 Reviewing the [27] Construction

We now recall the dual interpretation [27] of conjunction obfuscation by Bartusek, Lepoint, Ma and Zhandry, which is more efficient than the [30] construction.

8.5.1 The [27] Obfuscation Scheme

The dual scheme takes into account evasive conjunctions with patterns of length n, and achieves distributional virtual black box security for $n - \omega(\log n)$ wildcards in the generic group model with $n+1$ group elements, rather than $2n$. We start with a high-level overview of the scheme, followed by their formal descriptions (Algorithms 8.6 and 8.7).

Definition 8.5.1. *Let* $\boldsymbol{B}$ *be a* $(n+1) \times 2n$ *dimensional matrix defined as follows:*

$$\begin{pmatrix} 1 & 2 & \dots & 2n \\ 1 & 2^2 & \dots & (2n)^2 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 2^{n+1} & \dots & (2n)^{n+1} \end{pmatrix}$$

Then matrix $\mathbf{B}$ has the property that any of its $n+1$ columns form a full rank matrix.

To encode a pattern $\mathsf{pat} \in \{0, 1, \star\}^n$, compute a $2n$ dimensional *error vector* $\mathbf{e}$ structured as follows: if the ith bit of the pattern is b, then $\mathbf{e}_{2i-b} = 0$, while $\mathbf{e}_{2i-(1-b)}$ is sampled randomly from $\mathbb{Z}_q$. If $\mathsf{pat}_i = \star$, then $\mathbf{e}_{2i-1} = \mathbf{e}_{2i} = 0$. Finally, the obfuscator outputs the encoding of the vector $\mathbf{B}.\mathbf{e}$ in the exponent of the group $\mathbb{G} = \langle g \rangle$, as $g^{\mathbf{B}.\mathbf{e}} \in \mathbb{G}^{2n}$, which is tested by computing in the group.

On input string $x \in \{0,1\}^n$, the evaluation procedure solves for a vector $\mathbf{t}$, such that $\mathbf{t}.\mathbf{B} = 0$ at positions $2i + x_i - 1$, for every $i \in [n]$. Finally, x is accepted if $\mathbf{t}.\mathbf{B}.\mathbf{e} = 0$.

Algorithm 8.6 Obfuscator $\mathcal{O}_{Dual}(1^\lambda, C)$

Input: $n = n(\lambda)$
Output: $\mathbf{B}$, $\mathbf{v} = g^{\mathbf{B}.\mathbf{e}}$

1: Sample large prime $q > 2^{2\lambda}$.
2: Select $G = \langle g \rangle$ of order q.
3: Let $\mathbf{B} \in \mathbb{Z}_q^{(n+1)\times 2n}$ as in Definition 8.5.1.
4: Initialize error vector $\mathbf{e} \leftarrow \mathbb{Z}_q^{2n\times 1}$
5: **for** $i = 1$ to n **do**
6: **if** $\mathsf{pat}_i = \star$ **then**
7: $\mathbf{e}_{2i-1} = \mathbf{e}_{2i} = 0$
8: **end if**
9: **if** $\mathsf{pat}_i = b$ **then**
10: $\mathbf{e}_{2i-b} = 0$; $\mathbf{e}_{2i-(1-b)} \xleftarrow{R} \mathbb{Z}_q$
11: **end if**
12: **end for**
13: **return** $\mathbf{B}$, $\mathbf{v} = g^{\mathbf{B}.\mathbf{e}}$

Algorithm 8.7 Evaluation Eval (with embedded data $1^\lambda, \mathbf{B}, \mathbf{v}$)

Input: $x \in \{0,1\}^n$
Output: 0 or 1

1: Define $\mathbf{B}_x \in \mathbb{Z}_q^{(n+1)\times n}$, where column j is set to $(\mathbf{B}_x)_j = \mathbf{B}_{2j-x_j}$
2: Initialize error vector $\mathbf{e} \leftarrow \mathbb{Z}_q^{1\times(n+1)}$
3: Solve for non-zero vector $\mathbf{t} \in \mathbb{Z}_q^{1\times(n+1)}$ such that $\mathbf{t}.\mathbf{B}_x = 0$
4: **if** $\prod_{i=1}^{n+1} \mathbf{v}_i^{\mathbf{t}_i} = g^0$ **then**
5: **return** 1
6: **else**
7: **return** 0
8: **end if**

Bartusek *et al.* [27] claim to achieve *correctness* ϕ_3, which informally states that for a given program and an input, obfuscation is correct with overwhelming probability over the coins of the obfuscator (see Definition 8.3.1).

8.5.2 Designing Malicious Obfuscator for [27]

Designing a malicious obfuscator for the [27] scheme has several challenges. Recall from Definition 8.3.2, the malicious obfuscation instance need to be "sufficiently similar" in correctness preservation such that a poly-time verifier with a priori knowledge on C cannot distinguish between the honest and purported obfuscation instances. This naturally puts a constraint on replacing the zero elements in $\mathbf{e}$ with non-zero entries, as the obfuscated program might then reject correct inputs.

Elaborating on this, for $i \in [n]$, if both $\{\mathbf{e}_{2i-1}, \mathbf{e}_{2i-1}\}$ are non-zero, then this corresponds to a position where the input string can neither be 0 nor 1, and the obfuscator would reject all inputs with overwhelming probability. Furthermore, we cannot replace the non-zero entries in $\mathbf{e}$ with zero elements, as a poly-time verifier, who knows pat, can flip the non-wildcard bits one by one and determine if an incorrect input is accepted.

To construct an obfuscated program that accepts the bad input $y \in \{0,1\}^n$ with noticeable advantage, we again replace random values with specially chosen values. We assume y does not match pat (i.e. $C(y) = 0$ in Definition 8.4.1) otherwise we just have to return an honest obfuscation. Hence we may assume that y does not match the pattern in at least one entry. Recall that any $n+1$ columns of $\mathbf{B}$ are linearly independent, thus $\mathbf{B}_y \in \mathbb{Z}_q^{(n+1)\times n}$ will have rank n and there would be exactly one vector $\mathbf{t}$ up to scalar multiplication, such that $\mathbf{t}.\mathbf{B}_y = 0$ (by rank-nullity theorem). Recall that the $2n$ dimensional vector $\mathbf{e}$ has $n + |W|$ zero entries by construction. To design a purported error vector $\mathbf{e}^*$ that accepts y, along with the correct inputs, we fix the $n + |W|$ positions with zero entries (corresponding to $\mathbf{e}$). Computing $\mathbf{e}^*$ is done by finding a non-zero vector in the $(2n-1)$-dimensional subspace orthogonal to the vector $\mathbf{t}.\mathbf{B}$ that also has the correctly structured zero entries.

Let $\mathbf{E} \in \mathbb{Z}_q^{2n\times 1}$ be the subspace of vectors with basis $\{\mathbf{e}_{2i-(1-b)} : \mathsf{pat}_i = b\}$. Let $\mathbf{E}' = \{\mathbf{w} : \mathbf{w} \in \mathbf{E} \wedge \mathbf{t}.\mathbf{B}.\mathbf{w} = 0\}$. Since $n + |W|$ are fixed zero entries, the dimension of $\mathbf{E}'$ is $n - |W| - 1$. Thus, for an input y that does not match the pattern pat, if the obfuscator selects a vector $\mathbf{e}^*$ from $\mathbf{E}'$ and publishes $\mathbf{B}.\mathbf{e}^*$, the evaluation algorithm will accept y as $\mathbf{t}.\mathbf{B}.\ \mathbf{e}^* = 0$. We now describe the procedure formally in Algorithm 8.8.

Algorithm 8.8 Malicious Obfuscator $\mathcal{A}_{Dual}(1^\lambda, C, y \in \{0,1\}^n$, such that $C(y) = 0)$

Input: $n = n(\lambda)$
Output: $\mathbf{B}, \mathbf{v}^* = g^{\mathbf{B}.\mathbf{e}^*}$

1: Sample large prime $q > 2^{2\lambda}$.
2: Select $G = \langle g \rangle$ of order q.
3: Define fixed matrix $\mathbf{B} \in \mathbb{Z}_q^{(n+1)\times 2n}$ as given in Definition 8.5.1.
4: **if** $(C(y) == 1)$ **then**
5: **return** $\mathcal{O}_{Dual}(1^\lambda, C)$
6: **else**
7: Define $\mathbf{B}_y \in \mathbb{Z}_q^{(n+1)\times n}$, where column j is set to $(\mathbf{B}_y)_j = \mathbf{B}_{2j-y_j}$
8: Solve for non-zero vector $\mathbf{t} \in \mathbb{Z}_q^{1\times(n+1)}$ such that $\mathbf{t}.\mathbf{B}_y = 0$
9: Compute $\mathbf{E} \in \mathbb{Z}_q^{2n\times 1}$ with the basis $\{\mathbf{e}_{2i-(1-b)} : \mathsf{pat}_i = b\}$
10: Compute $\mathbf{E}' = \{\mathbf{w} : \mathbf{w} \in \mathbf{E} \wedge \mathbf{t}.\mathbf{B}.\mathbf{w} = 0\}$
11: Sample $\mathbf{e}^* \xleftarrow{R} \mathbf{E}'$
12: **return** $\mathbf{B}, \mathbf{v}^* = g^{\mathbf{B}.\mathbf{e}^*}$
13: **end if**

Theorem 8.5.1. *$\mathcal{A}_{Dual}$ violates ϕ_3.*

Proof. The dual scheme satisfies ϕ_3, i.e. the following holds for every $x \in \{0,1\}$:

$$\underset{\mathcal{O}_{Dual},C}{Pr}\ [\ \mathcal{O}_{Dual}(1^\lambda, C)(x) = C(x)\ ] > 1 - \mu(\lambda)$$

where $\mu(\lambda)$ is a negligible function in λ. To see if $\mathcal{A}_{Dual}$ violates ϕ_3, we need to determine whether it allows any bad input with noticeable probability.

Consider a fixed master backdoor $y \in \{0,1\}^n$. Then

$$\underset{\mathcal{O}_{Dual},C}{Pr}\ [\ \mathcal{O}_{Dual}(1^\lambda, C)(y) = C(y)\ ] > 1 - \mu(\lambda)$$

Consider correctness ϕ_3 experiment for $\mathcal{A}_{Dual}$ on the fixed input y. The experiment samples C from $\mathcal{C}_\lambda$, which implies sampling pat from $\mathcal{D}_\lambda$. Since $\mathcal{D}_\lambda$ is a distribution on evasive conjunction functions, $C(y) = 0$ with overwhelming probability in λ.

If $C(y) = 1$, then set $\mathcal{A}_{Dual}(1^\lambda, C, y) = \mathcal{O}_{Dual}(1^\lambda, C)$. If y does not satisfy the pattern pat, then execute Algorithm 8.8 on y, which calculates $\mathbf{t}$ such that $\mathbf{t}.\mathbf{B}$ is zero at the entries $2i + y_i - 1$, and computes $\mathbf{e}^*$ such that $\mathbf{t}.\mathbf{B}.\mathbf{e}^* = 0$. This ensures that evaluation procedure $\mathsf{Eval}(1^\lambda, \mathbf{B}, \mathbf{v}^* = g^{\mathbf{B}.\mathbf{e}^*})$ accepts y.

Since $\mathcal{A}_{Dual}(1^\lambda, C, y)(y) = 1$ for all C, we have

$$\Pr_{\mathcal{A}_{Dual}, C}\left[\, \mathcal{A}_{Dual}(1^\lambda, C, y)(y) \neq C(y) \,\right] > 1 - \mu(\lambda)$$

where $\mu(\lambda)$ is negligible in λ. This proves that $\mathcal{A}_{Dual}$ violates ϕ_3. □

8.5.3 Indistinguishability of Obfuscators

We now present an informal explanation for computational indistinguishability of the malicious and honest obfuscators for the dual scheme in the generic group model.

Theorem 8.5.2. *Let $\lambda \in \mathbb{N}$ be the security parameter and let n, w be polynomials in λ with $w = n - \omega(\log n)$. Let $\mathbb{G}$ be a group of prime order $q > 2^{2\lambda}$. Then for all PPT distinguishers $\mathcal{B}$ in the generic group model, there exists a negligible function $\mu(\lambda)$, such that for all $\lambda \in \mathbb{N}$, the following holds:*

$$\left| \Pr_{y, \mathcal{B}, \mathcal{A}_{Dual}}\left[\, \mathcal{B}^{\mathcal{A}_{Dual}(1^\lambda, \cdot, y)} = 1\right] - \Pr_{\mathcal{B}, \mathcal{O}_{Dual}}\left[\, \mathcal{B}^{\mathcal{O}_{Dual}(1^\lambda, \cdot)} = 1\right] \right| \leq \mu(\lambda)$$

Informal argument. We assume that the distinguisher $\mathcal{B}$ makes $T = T(\lambda)$ many adaptive queries to the obfuscation oracle, including repeated obfuscations of the same circuit. For simplicity, we assume that all the queries are with respect to the same parameters n, $|W|$ and q. The malicious input $y \in \{0,1\}^n$ represents the choice of aux in Definition 8.3.2. Since there are polynomially many chosen C, each corresponding to some evasive pattern pat, y does not satisfy any of the patterns with overwhelming probability.

To distinguish between $\mathcal{A}_{Dual}(1^\lambda, \cdot, y)$ and $\mathcal{O}_{Dual}(1^\lambda, \cdot)$, $\mathcal{B}$ has to determine the input y which is accepted which is accepted by the obfuscated programs, for all choices of C. But $\mathcal{B}$ does not know y.

Let $(\hat{e}^t)_{t=\{1,\ldots,T\}}$ be the sequence returned by $\mathcal{A}_{Dual}(1^\lambda, \cdot, y)$, and let $(e^t)_{t=\{1,\ldots,T\}}$ be the sequence returned by $\mathcal{O}_{Dual}(1^\lambda, \cdot)$ for the same choices of C. Based on the assumed hardness of solving discrete log in the group $\mathbb{G}$, $\mathcal{B}$ can only recover the group elements raised to the exponent with negligible probability in λ. Since y is not known and (e^t) is random, $(\hat{e}^t)$ behaves as random. Hence, $\mathcal{B}$ cannot decide if it is interacting with the malicious oracle or the honest one. □

8.5.4 Verifiable Obfuscator for [27]

We now design the verifiable obfuscator $\mathcal{VO}_{Dual}(1^\lambda, C)$ that provides security against the malicious obfuscator $\mathcal{A}_{Dual}(1^\lambda, C, y)$.

We want the obfuscator to give evidence that it samples the non-zero entries in the structured error vector uniformly at random, and not in an adversarially generated way.

We start by defining a pseudo-random generator $\widehat{\mathsf{PRG}}_{\mathsf{seed}} : \{0,1\}^{\log n} \to \mathbb{Z}_q$ indexed by seed $\mathsf{seed} \in \mathbb{Z}_q$. Note that the $2n$-dimensional structured vector $\mathbf{e}^*$ generated maliciously by $\mathcal{A}_{Dual}(1^\lambda, C, y)$ and $\mathbf{e}$ generated by $\mathcal{O}_{Dual}(1^\lambda, C)$ are elements of the subspace defined by $\{\mathbf{e}_{2i-(1-b)} : \mathsf{pat}_i = b\}$ such that the $n + |W|$ fixed zeroes allow inputs that match the pattern pat.

While $\mathcal{O}_{Dual}(1^\lambda, C)$ chooses the non-zero entries randomly from $\mathbb{Z}_q$, $\mathcal{A}_{Dual}(1^\lambda, C, y)$ chooses $\mathbf{w}$, such that $\mathbf{w} \in \mathbf{E} \wedge \mathbf{t}.\mathbf{B}.\mathbf{w} = 0$, where $\mathbf{t}.\mathbf{B}$ is zero at positions $2i + y_i - 1$ for every $i \in [n]$. To prove that the error vector is some randomly selected element of $\mathbf{E}$, and not some maliciously chosen $\mathbf{w}$ following the above discussion, our construction of verifiable obfuscator generates the non-zero entries using $\widehat{\mathsf{PRG}}_{\mathsf{seed}}$.

We now describe the construction formally in Algorithm 8.9.

Algorithm 8.9 Obfuscator $\mathcal{VO}_{Dual}(1^\lambda, C)$

Input: $n = n(\lambda)$
Output: $\mathbf{B}, \hat{\mathbf{v}} = g^{\mathbf{B}.\hat{\mathbf{e}}}$, seed

1: Sample large prime $q > 2^{2\lambda}$.
2: Select $G = \langle g \rangle$ of order q.
3: Sample $\mathsf{seed} \xleftarrow{R} \mathbb{Z}_q$
4: Define fixed matrix $\mathbf{B} \in \mathbb{Z}_q^{(n+1)\times 2n}$ as given in Definition 8.5.1.
5: Initialize error vector $\hat{\mathbf{e}} \leftarrow \mathbb{Z}_q^{2n\times 1}$
6: **for** $i = 1$ to n **do**
7: **if** $\mathsf{pat}_i = \star$ **then**
8: $\hat{\mathbf{e}}_{2i-1} = \mathbf{e}_{2i} = 0$
9: **end if**
10: **if** $\mathsf{pat}_i = b$ **then**
11: $\hat{\mathbf{e}}_{2i-b} = 0$
12: $\hat{\mathbf{e}}_{2i-(1-b)} \leftarrow \widehat{\mathsf{PRG}}_{\mathsf{seed}}(i)$
13: **end if**
14: **end for**
15: **return** $\mathbf{B}, \hat{\mathbf{v}} = g^{\mathbf{B}.\hat{\mathbf{e}}}$, seed

We now explain the construction of $\mathcal{VO}_{Dual}.\mathsf{Verify}(1^\lambda, \mathsf{pat}, \mathbf{B}, g^{\mathbf{B}.\hat{\mathbf{e}}}, \mathsf{seed})$, which outputs 1, if $g^{\mathbf{B}.\hat{\mathbf{e}}}$ is the correct obfuscation for the pattern pat.

$\mathcal{VO}_{Dual}.\mathsf{Verify}$ constructs the $2n$-dimensional structured error vector $\tilde{\mathbf{e}}$ using $\widehat{\mathsf{PRG}}$ indexed by seed seed, with zero entries fixed as follows: $\tilde{\mathbf{e}}_{2i-1} = \tilde{\mathbf{e}}_{2i} = 0$ if $\mathsf{pat}_i = \star$, and $\tilde{\mathbf{e}}_{2i-b} = 0$ if $\mathsf{pat}_i = b$. The non-zero entries in $\tilde{\mathbf{e}}$ are generated using $\widehat{\mathsf{PRG}}_{\mathsf{seed}}$.

Since $\widehat{\mathsf{PRG}}$ is a deterministic function, for the fixed seed seed, $\tilde{\mathbf{e}}$ will be exactly same as $\hat{\mathbf{e}}$. Finally, the verifier outputs 1, if the values match. The formal description is given in Algorithm 8.10.

Algorithm 8.10 $\mathcal{VO}_{Dual}$.Verify (with embedded values $\mathbf{B}, g^{\mathbf{B}.\hat{\mathbf{e}}}$, seed)

Input: $n = n(\lambda)$, $\mathsf{pat} \in \{0,1,\star\}^n$
Output: 1 or $\perp$

Initialize error vector $\tilde{\mathbf{e}} \leftarrow \mathbb{Z}_q^{2n\times 1}$
for $i = 1$ to n **do**
  if $\mathsf{pat}_i = \star$ **then**
    $\tilde{\mathbf{e}}_{2i-1} = \tilde{\mathbf{e}}_{2i} = 0$
  end if
  if $\mathsf{pat}_i = b$ **then**
    $\tilde{\mathbf{e}}_{2i-b} = 0$
    $\tilde{\mathbf{e}}_{2i-(1-b)} \leftarrow \widehat{\mathsf{PRG}}_{\mathsf{seed}}(i)$
  end if
end for
Compute $g^{\mathbf{B}.\tilde{\mathbf{e}}}$
if $g^{\mathbf{B}.\tilde{\mathbf{e}}} == g^{\mathbf{B}.\hat{\mathbf{e}}}$ **then**
  return 1
else
  return $\perp$
end if

Theorem 8.5.3. *Let* $\widehat{\mathsf{PRG}}$ *be indistinguishable from uniform. Then for every PPT adversary* $\mathcal{A}$ *and every* $(\boldsymbol{B}, g^{\boldsymbol{B}.\boldsymbol{e}}, \mathsf{seed}) \leftarrow \mathcal{A}(1^\lambda, C, y)$, *if* $\mathcal{VO}_{Dual}.\mathsf{Verify}(1^\lambda, \mathsf{pat}, \boldsymbol{B}, g^{\boldsymbol{B}.\boldsymbol{e}}, \mathsf{seed}) = 1$, *then* $g^{\boldsymbol{B}.\boldsymbol{e}}$ *satisfies* ϕ_3.

Proof. Towards a contradiction, we suppose the obfuscated program $g^{\mathbf{B}.\mathbf{e}}$ does not satisfy ϕ_3, and there exists a $y \in \{0,1\}^n$ such that the following holds:

$$\Pr_{\mathcal{A},C}\left[\,\mathcal{A}(1^\lambda, C, y)(y) \neq C(y)\,\right] > \epsilon(\lambda)$$

where $\epsilon(\lambda) = \lambda^{-\mathrm{O}(1)}$ is a noticeable function in λ.

Since the purported obfuscation $(g^{\mathbf{B}.\mathbf{e}}, \mathsf{seed})$ is accepted by $\mathcal{VO}_{Dual}.\mathsf{Verify}$, this implies that the honest obfuscator $\mathcal{O}$ could have generated the same error vector $\mathbf{e}$ by choosing the same seed seed. But, $\mathcal{A}$ could have tried at most polynomially many seeds, say $p(\lambda)$, which means the honest obfuscator could also have generated the same output with probability $\frac{1}{p(\lambda)}$. This implies that running $\mathcal{O}$ once yields

$$\Pr_{\mathcal{O},C}\left[\,\mathcal{O}(1^\lambda, C)(y) \neq C(y)\,\right] > \frac{\epsilon(\lambda)}{p(\lambda)}$$

which is still a noticeable function in λ. This contradicts the results of Bartusek *et al.* [27] that for every input, $\mathcal{O}(1^{\lambda}, C)$ is correct with overwhelming probability over the coin tosses of the obfuscator and the program (see Definition 8.3.1). □

8.6 Verifiable Control Program Obfuscation

In Chapter 1 we have discussed the significance of preventing adversarial attempts of extracting the operational semantics of an industrial control application in defending against targeted attacks. In Chapter 7 we have introduced a MATE adversary who aims at reverse-engineering a recovered implementation of control program to learn the process semantics of a target control application, and presented the proposed ObfCP platform that prevents such adversarial attempts by making use of cryptographic obfuscation. However, there exists obfuscators that inject malicious functionality to manipulate critical control parameters, leading to fatal consequences. In what follows, we elaborate on *malicious control program obfuscation, and verification techniques that provide security against such malicious obfuscators.*

8.6.1 Malicious Obfuscator for the Proposed ObfCP Construction

In this section, we discuss how a malicious obfuscator could trigger conditions to manipulate a critical control parameter. We start with a brief intuition to the importance of verifiability of control program obfuscation through the following motivating scenario.

Consider the example Water Distribution System introduced in Section 7.2.1. The control program describing its operational behaviour is given as follows:

```
PROGRAM PLC_PRG
IF (chlorine_level <= s1 and water_level > s2) THEN
    pressure := 15;
ELSE
    pressure := 0;
END_IF
```

To obfuscate PLC_PRG, the proposed ObfCP platform encodes the setpoints $[\![s_1, s_2]\!]$, such that it is well hidden from the MATE adversary introduced in Section

7.2.2. Now consider a malicious obfuscator $\mathcal{A}$ who requires the obfuscated program to allow inputs that do not satisfy the condition in PLC_PRG, in a way that the purported obfuscation is hard to distinguish from the honest obfuscation instance to a verifier who has a priori knowledge on PLC_PRG. To illustrate this, suppose $\mathcal{A}$ modifies the setpoint s_2 to s_2', such that $s_2' > s_2$. The obfuscated program now accepts additional inputs for triggering conditions to manipulate the pressure valve. The outcome of this attack is that even when the water level in the tank is significantly high, pressure is set to 0, which stops the water supply to the consumers.

We now design a malicious obfuscator for the decision tree obfuscation scheme (introduced in Chapter 6) which is employed in developing the ObfCP platform (introduced in Chapter 7). The malicious obfuscator simply replaces some of the randomly generated values with specially chosen ones, such that a fixed master backdoor $y = (y_1, \ldots, y_n) \in \mathbb{N}^n$ is accepted by the obfuscated program. We assume that y is a bad input, and hence $C(y) = 0$, otherwise we would have to return the honest obfuscation (Algorithm 6.5).

Since $(y_1, \ldots, y_n)$ is not an accepting input, there exists at least one value y_i, such that $y_i \notin (c_i, c_i + w_i]$. To allow the input, the malicious obfuscator could simply perform $H(y_i)$, add it to the set $\mathcal{A}^i$, and finally for each entry in $\mathcal{A}^i$, concatenate n entries sorted in order of i, apply cryptographic hash H_c, and publish the set of hash values $\mathcal{E} = (h_1, \ldots, h_\alpha)$. This ensures that obfuscated program accepts all the good inputs, along with the secret master backdoor y. Note that, a distinguisher $\mathcal{B}$ who does not know y is unable to determine whether the allegedly random dummy entry is instead the hashed encoding corresponding to y.

Hence, following our previous solutions to verifiable obfuscation, we first derandomize obfuscation by arranging the randomly chosen dummy entries (Steps 25-28 in Algorithm 6.5) to be generated using a pseudorandom number generator on an initially randomly chosen seed. Then the proof of correct obfuscation is simply the value of the seed. The distinguisher can simply calculate the hashed encodings corresponding to the intervals $(c_i, c_i + w_i]$'s, generate the dummy entries using the pseudorandom generator indexed by the same seed, and check if the output is same as $\mathcal{E}$.

We note that the proposed ObfCP platform does not employ enough randomness, and hence secure against malicious obfuscators.

8.7 Conclusion

In this chapter, we introduce a new variant of malicious obfuscation that inserts a secret hard-to-guess master backdoor in the original program. We show undetectable malicious obfuscators for a number of published obfuscation schemes in the theoretical literature. We introduce verifiable obfuscation that adds a verifiability property for securing against malicious obfuscation. We provide a general approach to verifiable obfuscation. Our approach is not very efficient as the proof involves re-computing obfuscation for verifying the steps taken in converting the original code to the obfuscated program. Finally, we show verifiable schemes for the proposed decision tree obfuscator, used for designing the ObfCP platform. As our future initiative, we aim to explore other randomized obfuscation schemes in the literature to determine if they can be exploited for undetectable malicious functionality.

Chapter 9

Conclusions and Future Work

In this chapter, we present an overall conclusion to the key contributions made towards addressing the research problem identified in Chapter 1. We also suggest some possible directions for future research.

9.1 Summary Description

This research work focuses upon studying the characterizing features of high-profile targeted attacks delivered against industrial infrastructures, understanding the importance of evolving adversary tradecraft in delivering such attacks, and examining the defense and policy responses by the ICS security research community and industry practitioners towards preventing, detecting and deterring the attacks.

In Section 3.1.1, we investigate the existing methodologies for detecting false-data injection attacks, highlighting the inaccuracies of the predictive intrusion detection models, while stating that the bump-in-the-wire solutions cause unacceptable overheads. Towards this end, in Chapter 4 *we propose* TaDeT*, a general-purpose tamper detection framework that is neither inaccurate, nor does it impact the network bandwidth used for critical control communication.*

In Section 3.1.2, we discuss the significance of application-specific security in industrial control infrastructures, and why the existing literature/recommendations fail to provide such customized security controls. To this end, in Chapter 5 *we propose a framework* SelEnc *that enables application-specific security, along with incurring significantly low overhead compared to the existing methodologies.*

In Chapter 6, we design an efficient VBB obfuscator for evasive binary decision trees, whose security is based on the random oracle paradigm. While doing so, we present an encoder for interval-membership functions. *Our exclusive goal behind designing the obfuscator is that, not only will the solution increase the class of functions that has cryptographically secure obfuscators, but also address the open problem of non-interactive prediction in privacy-preserving classification using computationally inexpensive cryptographic hash functions, along with preventing adversarial attempts of extracting operational semantics of industrial control applications, the last of which is the ultimate aim of this work.*

In Section 3.1.3, we study the importance attributed to the knowledge of process control in bringing about high-profile targeted attacks, and analyze the adversarial attempts in learning the operational semantics of target process control. In Section 3.1.4, we show the lack of methodical approach in formalizing control programs. To this end in Chapter 7, *we present a formalization of abstraction of control programs and define its assets, the secret values in the program that give away the process semantics. Finally, we introduce* ObfCP*, a legacy-compliant, general-purpose platform for securing assets in control programs, thus preventing extraction of process semantics.*

In Section 2.5.2, we *introduce a new and powerful variant of malicious obfuscators* that implement subliminal channels to deviate from correct functionality, and are indistinguishable from honest obfuscation instances. We *prove the existence of such obfuscators in the literature, and propose verifiable obfuscating schemes to defend against such obfuscators.*

9.2 Future Directions of Research

We now highlight the possible directions of research that we aim to investigate further, as a part of our future initiatives.

9.2.1 Preserving Integrity of Control Programs with Software Watermarking

In this section, we discuss the significance of software watermarking in preventing piracy of control programs. We begin with a general discussion on software watermarking along with its security guarantees, followed by a brief discussion on the state-of-art methodologies for embedding watermarks in software programs.

Software watermarking is a technology that encompasses embedding an identifier in a software program, which acts as a copyright notice discouraging intellectual property theft. The intuition is to reliably locate and extract the identifier, even if the program has been subjected to authorized (obfuscation, optimization) and/or unauthorized (tampering) changes. A watermarking scheme is typically evaluated w.r.t the following parameters: (a) *credibility* (how strongly the watermarked program proves authorship), (b) *secrecy* (how difficult it is to extract the watermark), (c) *transparent* (how difficult it is to distinguish a watermarked program from the original program), (d) *functionality preservation* (whether the input-output behavior is identical for both the watermarked and original program), (e) *resilience* (how difficult it is to compromise the correct extraction of the identifier).

A number of software watermarking techniques have been proposed in the literature [226]. In [68], the authors embed a watermark by calculating hash values for each basic block in the original code, sorting the blocks according to the hash values, and applying a permutation to reorder the hashed values, which serves as an unique identifier. Authors in [138], propose an algorithm which embeds a watermark by replacing instructions in a dummy method. In [181], Sharma *et al.* propose a methodology that reorders arithmetic operations in functions, and then modifies the sequence in which functions are defined, such that a definite watermark is encoded in the sequencing. However, these approaches lack theoretical investigations and rigorous definitional frameworks involving security analysis of the schemes. In [217], the authors propose the notion of watermarking based on cryptographic signatures, the security analysis of which relies upon vector decomposition problem [218]. In [148], Nishimaki characterized two fundamental requirements for watermarking functions:

- The watermarked function should be functionally equivalent to the original program.
- Removing embedded watermark from a watermarked function should be computationally difficult for a polynomial adversary.

Our construction in Chapter 7 prevents adversarial attempts to reverse engineer a control program. *A future research topic is to develop tools for watermarking control programs, such that adversarial attempts to modify the programs (by exploiting the insecure communication protocols at the configuration layer) could be efficiently*

determined. Furthermore, we aim to determine whether we can efficiently watermark obfuscated implementations of control programs to achieve protection against program piracy.

9.2.2 Extending $SelEnc$ to Achieve Diverse Security Goals

In Chapter 5 we have presented a framework that helps achieve security in communicating a subset of ICS payloads critical to the control behavior of the application, incurring significantly low overhead. As a part of the future initiatives, we aim to extend our framework towards providing *authentication* and *integrity* in ICS communication. Furthermore, we aim to *examine ICS products and protocols* (other than Allen Bradley controllers and EtherNet/IP protocol as used in our experiments), *conduct in-depth analysis on the customer security requirements based on extensive user stories*, such that standard paradigms of application-level security could be designed for generic industrial infrastructures.

9.2.3 Extending the notion of Verifiable Obfuscation

An interesting direction of future research in verifiable obfuscation would be:

- Our proposed verifiable obfuscation concept requires the verifier to be provided with the original program C. Can verification be performed in a "zero knowledge" way with respect to the original program, so that the verifier takes a commitment $\mathsf{Commit}(C)$ to the original program C, the obfuscated program $\tilde{C}$, and the proof π?
- Our proof-of-concept verifiable obfuscation is not very efficient as it essentially requires re-doing the obfuscation process. An open problem is to develop more efficient schemes that does not require the verifier to do roughly the same amount of work as the obfuscator. One way to achieve this is to consider variants that verify a random subset of steps (This can actually be done for the [30] scheme, for example). This begs the question: what level of assurance can be achieved when the verifier's work is much less than the prover's work ? Are there verifiable obfuscation schemes where the verifier's work is substantially less than the obfuscator's work?
- Our design requires the verifier to understand the obfuscation tool. However, the obfuscation techniques can be proprietary and hence need to be protected from the verifier. A future work in this direction would be to build verifiable obfuscation around this constraint.

9.2.4 Designing New Foundations of Practical Program Obfuscation for Control Programs

Another possible future direction that would be interesting to work on, is the design of *new cryptographically secure obfuscators for branching programs*, involving lower computational overheads compared to our proposed ObfCP construction, such that they could implemented in generic industrial control infrastructures. Our interest in branching program is due to its alignment to the structural representation of control programs.

9.3 Conclusion

The specter of adversary tradecraft in delivering targeted attacks against critical infrastructures is increasing at a significant rate. The countermeasures to defend against such adversarial attempts are mainly restricted to patch management, compliance-based regimes, and IDS with signatures of vulnerabilities being discovered every day, which do not prevent exploitations leveraging the design flaws of a generic industrial control framework. With an aspiring ambition of Industry 5.0, ICS security should be a collaborative initiative between industry practitioners and academic research community, such that industry can implement custom-made security controls based on rigorous foundations of lightweight cryptographic primitives instead of importing IT security technologies, and the research community, who is currently reliant on simulations and emulations, can avail fully functioning ICS testbeds to evaluate their approaches.

Bibliography

[1] Chuadhry Mujeeb Ahmed, Venkata Reddy Palleti, and Aditya P Mathur. "WADI: a water distribution testbed for research in the design of secure cyber physical systems". In: *Proceedings of the 3rd international workshop on cyber-physical systems for smart water networks*. 2017, pp. 25–28.

[2] Adi Akavia et al. "Privacy-preserving decision trees training and prediction". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2020, pp. 145–161.

[3] Pedro Albertos and Manuel Olivares. "Time delay limitations in control implementation". In: *1999 European Control Conference (ECC)*. IEEE. 1999, pp. 1288–1293.

[4] Wolfgang Altmann. *Practical process control for engineers and technicians*. Elsevier, 2005.

[5] ANSI/ISA. *Security for Industrial Automation and Control Systems*. https://webstore.ansi.org/preview-pages/ISA/preview_S_990001_2007.pdf. 2007.

[6] Genevieve Arboit. "A method for watermarking java programs via opaque predicates". In: *The Fifth International Conference on Electronic Commerce Research (ICECR-5)*. 2002, pp. 102–110.

[7] Industry ARC. *Industrial Ethernet/IP Market - Forecast(2022 - 2027)*. https://www.industryarc.com/Report/19415/industrial-ethernet-ip-market.html.

[8] Muhammad Rizwan Asghar et al. "Towards a theory of special-purpose program obfuscation". In: *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE. 2020, pp. 394–401.

[9] Michael J Assante and Robert M Lee. "The industrial control system cyber kill chain". In: *SANS Institute InfoSec Reading Room* 1 (2015).

[10] ControlNet InternationalandOpen DeviceNet Vendor Association. *EtherNet/IP Adaptation of CIP*. https://silo.tips/download/ethernet-ip-adaptation-of-cip-specification.

[11] Open DeviceNet Vendor Association. *Quick Start for Vendors Handbook*. https://www.odva.org/wp-content/uploads/2020/05/PUB00213R0_EtherNetIP_Developers_Guide.pdf. 2008.

[12] Giuseppe Ateniese et al. "Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers". In: *International Journal of Security and Networks* 10.3 (2015), pp. 137–150.

[13] AusNet. *Analysis of Cyber Attack on the Ukrainian Power Grid*. https://www.aer.gov.au/system/files/AusNet.

[14] Automatak. *TSSP 21, CES-21 TLP WHITE*. https://ssp21.github.io/spec/latest/pdf/ssp21.pdf. 2020.

[15] Real Time Automation. *A LITTLE BACKGROUND ON ETHERNET/IP*. https://www.rtautomation.com/technologies/ethernetip/.

[16] Real Time Automation. *MODBUS TCP/IP*. https://www.rtautomation.com/technologies/modbus-tcpip/.

[17] Rockwell Automation. *PLC Programmable Controllers*. https://www.rockwellautomation.com/en-nz/products/hardware/allen-bradley/programmable-controllers.html.

[18] Rockwell Automation. *Rockwell Automation Allen-Bradley Logix 5000 Series Programming Manual*. https://www.manualslib.com/manual/1597627/Rockwell-Automation-Allen-Bradley-Logix-5000-Series.html#manual.

[19] Mike Bacidore. *Ethernet vs. fieldbus: the right network for the right application*. https://www.controldesign.com/connections/edge-technology/article/11320574/ethernet-vs-fieldbus-the-right-network-for-the-right-application. 2016.

[20] David F Bacon, Susan L Graham, and Oliver J Sharp. "Compiler transformations for high-performance computing". In: *ACM Computing Surveys (CSUR)* 26.4 (1994), pp. 345–420.

[21] Saikrishna Badrinarayanan et al. "Verifiable functional encryption". In: *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*. Springer. 2016, pp. 557–587.

[22] Baesystems. *Havex*. https://www.baesystems.com/en/cybersecurity/feature/havex.

[23] Boaz Barak et al. "Obfuscation for evasive functions". In: *Theory of Cryptography Conference*. Springer. 2014, pp. 26–51.

[24] Boaz Barak et al. "On the (im) possibility of obfuscating programs". In: *Journal of the ACM (JACM)* 59.2 (2012), pp. 1–48.

[25] Mauro Barni et al. "Efficient privacy-preserving classification of ECG signals". In: *2009 First IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE. 2009, pp. 91–95.

[26] Mauro Barni et al. "Secure evaluation of private linear branching programs with medical applications". In: *European symposium on research in computer security*. Springer. 2009, pp. 424–439.

[27] James Bartusek et al. "New techniques for obfuscating conjunctions". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2019, pp. 636–666.

[28] Brian Batke, Joakim Wiberg, and Dennis Dubé. "CIP security phase 1 secure transport for Ethernet/IP". In: *ODVA Industry Conference*. 2015.

[29] Mihir Bellare and Phillip Rogaway. "Random oracles are practical: A paradigm for designing efficient protocols". In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. 1993, pp. 62–73.

[30] Allison Bishop et al. "A simple obfuscation scheme for pattern-matching with wildcards". In: *Annual International Cryptology Conference*. Springer. 2018, pp. 731–752.

[31] Nir Bitansky and Omer Paneth. "On the impossibility of approximate obfuscation and applications to resettable cryptography". In: *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. 2013, pp. 241–250.

[32] Edward S Blurock. "Automatic learning of chemical concepts: Research octane number and molecular substructures". In: *Computers & chemistry* 19.2 (1995), pp. 91–99.

[33] Terry Borden and Richard A Cox. *Technician's guide to programmable controllers*. Cengage Learning, 2022.

[34] Joppe W Bos, Kristin Lauter, and Michael Naehrig. "Private predictive analysis on encrypted medical data". In: *Journal of biomedical informatics* 50 (2014), pp. 234–243.

[35] Raphael Bost et al. "Machine learning classification over encrypted data". In: *Cryptology ePrint Archive* (2014).

[36] Elette Boyle et al. "On Low-End Obfuscation and Learning". In: *14th Innovations in Theoretical Computer Science Conference (ITCS 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2023.

[37] Justin Brickell et al. "Privacy-preserving remote diagnostics". In: *Proceedings of the 14th ACM conference on Computer and communications security*. 2007, pp. 498–507.

[38] Danilo Bruschi, Lorenzo Martignoni, and Mattia Monga. "Code normalization for self-mutating malware". In: *IEEE Security & Privacy* 5.2 (2007), pp. 46–54.

[39] Danilo Bruschi, Lorenzo Martignoni, and Mattia Monga. "Detecting self-mutating malware using control-flow graph matching". In: *International conference on detection of intrusions and malware, and vulnerability assessment*. Springer. 2006, pp. 129–143.

[40] Géraud Canet et al. "Towards the automatic verification of PLC programs written in Instruction List". In: *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*. Vol. 4. IEEE. 2000, pp. 2449–2454.

[41] Ran Canetti. "Towards realizing random oracles: Hash functions that hide all partial information". In: *Annual International Cryptology Conference*. Springer. 1997, pp. 455–469.

[42] Ran Canetti, Guy N Rothblum, and Mayank Varia. "Obfuscation of hyperplane membership". In: *Theory of Cryptography Conference*. Springer. 2010, pp. 72–89.

[43] Ran Canetti and Mayank Varia. "Non-malleable obfuscation". In: *Theory of Cryptography Conference*. Springer. 2009, pp. 73–90.

[44] Ran Canetti et al. "COA-Secure obfuscation and applications". In: *Advances in Cryptology–EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30–June 3, 2022, Proceedings, Part I*. Springer. 2022, pp. 731–758.

[45] Jan Cappaert et al. "Self-encrypting code to protect against analysis and tampering". In: *1st Benelux Workshop Inf. Syst. Security*. 2006, p. 14.

[46] Alvaro A Cárdenas et al. "Attacks against process control systems: risk assessment, detection, and response". In: *Proceedings of the 6th ACM symposium on information, computer and communications security*. 2011, pp. 355–366.

[47] Marco Caselli, Emmanuele Zambon, and Frank Kargl. "Sequence-aware intrusion detection in industrial control systems". In: *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*. 2015, pp. 13–24.

[48] John Henry Castellanos et al. "Legacy-compliant data authentication for industrial control system traffic". In: *International Conference on Applied Cryptography and Network Security*. Springer. 2017, pp. 665–685.

[49] Crate Chacha20. *Implementation of the ChaCha Family of Stream Ciphers*. https://docs.rs/chacha20/latest/chacha20/#:~:text=ChaCha%20stream%20ciphers%20are%20lightweight,with%20no%20cost%20to%20performance..

[50] Hong Chen. "Applications of cyber-physical system: a literature review". In: *Journal of Industrial Integration and Management* 2.03 (2017), p. 1750012.

[51] Steven Cheung et al. "Using model-based intrusion detection for SCADA networks". In: *Proceedings of the SCADA security scientific symposium*. Vol. 46. Citeseer. 2007, pp. 1–12.

[52] Stelvio Cimato, Alfredo De Santis, and U Ferraro Petrillo. "Overcoming the obfuscation of Java programs by identifier renaming". In: *Journal of systems and software* 78.1 (2005), pp. 60–72.

[53] CISA. *Cross-Sector Cybersecurity Performance Goals (CPGs) Common Baseline: Controls List*. https://www.cisa.gov/sites/default/files/publications/Common_Baseline_v2_Controls_List_508c.pdf.

[54] C. Collberg. *the tigress c obfuscator*. https://tigress.wtf/index.html. 2023.

[55] Christian Collberg, Clark Thomborson, and Douglas Low. *A taxonomy of obfuscating transformations*. Tech. rep. Department of Computer Science, The University of Auckland, New Zealand, 1997.

[56] Christian Collberg, Clark Thomborson, and Douglas Low. "Breaking abstractions and unstructuring data structures". In: *Proceedings of the 1998 International Conference on Computer Languages (Cat. No. 98CB36225)*. IEEE. 1998, pp. 28–38.

[57] Christian Collberg, Clark Thomborson, and Douglas Low. "Manufacturing cheap, resilient, and stealthy opaque constructs". In: *Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 1998, pp. 184–196.

[58] Christian Sven Collberg, Clark David Thomborson, and Douglas Wai Kok Low. *Obfuscation techniques for enhancing software security*. US Patent 6,668,325. Dec. 2003.

[59] International Electrotechnical Commission. *Industrial communication networks – Fieldbus specifications*. https://webstore.iec.ch/publication/59890. 2016.

[60] Kelong Cong et al. "SortingHat: Efficient Private Decision Tree Evaluation via Homomorphic Encryption and Transciphering". In: *Cryptology ePrint Archive* (2022).

[61] Stamford Conn. *Gartner Predicts 30% of Critical Infrastructure Organizations Will Experience a Security Breach by 2025*. https://www.gartner.com/en/newsroom/press-releases/2021-12-2-gartner-predicts-30--of-critical-infrastructure-organi. 2021.

[62] Bart Coppens, Bjorn De Sutter, and Jonas Maebe. "Feedback-driven binary code diversification". In: *ACM Transactions on Architecture and Code Optimization (TACO)* 9.4 (2013), pp. 1–26.

[63] Giovanni Di Crescenzo. "Cryptographic formula obfuscation". In: *International Symposium on Foundations and Practice of Security*. Springer. 2018, pp. 208–224.

[64] Tiago Cruz et al. "Improving network security monitoring for industrial control systems". In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE. 2015, pp. 878–881.

[65] Cybersecurity and Infrastructure Security Agency. *Cyber-Attack Against Ukrainian Critical Infrastructure*. https://www.cisa.gov/uscert/ics/alerts/IR-ALERT-H-16-056-01.

[66] ATT Cybersecurity. *AlienVault HIDS*. https://cybersecurity.att.com/documentation/usm-appliance/ids-configuration/about-alienvault-hids.htm.

[67] Joan Daemen and Vincent Rijmen. "AES proposal: Rijndael". In: (1999).

[68] Robert I Davidson and Nathan Myhrvold. *Method and system for generating and auditing a signature for a computer program*. US Patent 5,559,884. Sept. 1996.

[69] Martine De Cock et al. "Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation". In: *IEEE Transactions on Dependable and Secure Computing* 16.2 (2017), pp. 217–230.

[70] Christine Decaestecker et al. "Methodological aspects of using decision trees to characterise leiomyomatous tumors". In: *Cytometry: The Journal of the International Society for Analytical Cytology* 24.1 (1996), pp. 83–92.

[71] Dragos. *CHERNOVITE's PIPEDREAM Malware Targeting Industrial Control Systems (ICS)*. https://www.dragos.com/blog/industry-news/chernovite-pipedream-malware-targeting-industrial-control-systems/.

[72] Dragos. *Rise in threat groups, vulnerabilities, and ransomware as ICS/OT systems digitally transform*. https://www.theee.ai/2022/02/23/16430-rise-in-threat-groups-vulnerabilities-and-ransomware-as-ics-ot-systems-digitally-transform/.

[73] Stephen Drape. "Generalising the array split obfuscation". In: *Information Sciences* 177.1 (2007), pp. 202–219.

[74] Stephen Drape. "Obfuscation of Abstract Data- Types". In: (2004).

[75] LINEAR SOLUTIONS MADE EASY. "ETHERNET/IP PROGRAMMER'S GUIDE". In: ().

[76] Thomas Eisenbarth et al. "A survey of lightweight-cryptography implementations". In: *IEEE Design & Test of Computers* 24.6 (2007), pp. 522–533.

[77] ESCHWEIGERT. *SCADA Security Basics: Why are PLCs so Insecure?* https://www.tofinosecurity.com/blog/scada-security-basics-why-are-plcs-so-insecure. 2012.

[78] *EtherNet/IP focuses on digitization and process industry applications*. https://iebmedia.com/technology/industrial-ethernet/ethernet-ip-focus-on-digitization-and-process-industry-applications/. 2022.

[79] Albert Falcione and Bruce H Krogh. "Design recovery for relay ladder logic". In: *IEEE Control Systems Magazine* 13.2 (1993), pp. 90–98.

[80] Allie Fick. *Critical infrastructure is more vulnerable than ever—your industry could be a prime target*. https://www.lacework.com/fr/blog/critical-infrastructure-is-more-vulnerable-than-ever-your-industry-could-be-a-prime-target/. 2022.

[81] Th Filkorn et al. "Formal verification of PLC-programs". In: *IFAC Proceedings Volumes* 32.2 (1999), pp. 1513–1518.

[82] Silvan Fischer and Hans Dermot Doran. "Embedding Real Time Ethernet: EtherNet/IP on resource constricted platforms". In: *ETFA2011*. IEEE. 2011, pp. 1–4.

[83] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. "Model inversion attacks that exploit confidence information and basic countermeasures". In: *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 2015, pp. 1322–1333.

[84] Radek Fujdiak et al. "Communication model of smart substation for cyber-detection systems". In: *International Conference on Computer Networks*. Springer. 2019, pp. 256–271.

[85] Steven D Galbraith and Lukas Zobernig. "Obfuscated fuzzy hamming distance and conjunctions from subset product problems". In: *Theory of Cryptography Conference*. Springer. 2019, pp. 81–110.

[86] Shafi Goldwasser and Guy N Rothblum. "On best-possible obfuscation". In: *Theory of Cryptography Conference*. Springer. 2007, pp. 194–213.

[87] Vincent Gourcuff, Olivier De Smet, and J-M Faure. "Efficient representation for formal verification of PLC programs". In: *2006 8th International Workshop on Discrete Event Systems*. IEEE. 2006, pp. 182–187.

[88] Rishab Goyal, Venkata Koppula, and Brent Waters. "Lockable obfuscation". In: *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2017, pp. 612–621.

[89] Rishab Goyal, Venkata Koppula, and Brent Waters. "Separating semantic and circular security for symmetric-key bit encryption from the learning with errors assumption". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2017, pp. 528–557.

[90] Hadeli Hadeli et al. "Leveraging determinism in industrial control systems for advanced anomaly detection and reliable security configuration". In: *2009 IEEE Conference on Emerging Technologies & Factory Automation*. IEEE. 2009, pp. 1–8.

[91] Dina Hadžiosmanović et al. "Through the eye of the PLC: semantic security monitoring for industrial processes". In: *Proceedings of the 30th Annual Computer Security Applications Conference*. 2014, pp. 126–135.

[92] Dag H Hanssen. *Programmable logic controllers: a practical approach to IEC 61131-3 using CODESYS*. John Wiley & Sons, 2015.

[93] *hashlib — Secure hashes and message digests*. https://docs.python.org/3/library/hashlib.html.

[94] Kelly Heffner and Christian Collberg. "The obfuscation executive". In: *International Conference on Information Security*. Springer. 2004, pp. 428–440.

[95] Simon Heron. "Advanced encryption standard (AES)". In: *Network Security* 2009.12 (2009), pp. 8–12.

[96] Edgar F. Hilton. *Sample rates for your industrial applications?* https://control.com/forums/threads/sample-rates-for-your-industrial-applications.2677/. 2022.

[97] Michael Hoffman. "Vulnerabilities on the wire: Mitigations for insecure ICS device communication". In: *Cyber Security: A Peer-Reviewed Journal* 4.2 (2020), pp. 160–181.

[98] Yan Hu et al. "A survey of intrusion detection on industrial control systems". In: *International Journal of Distributed Sensor Networks* 14.8 (2018), p. 1550147718794615.

[99] IEC Iec. "61131-3: Programmable controllers–part 3: Programming languages". In: *International Standard, Second Edition, International Electrotechnical Commission, Geneva* 1 (2003), p. 2003.

[100] Dragos Inc. *TRISIS Malware*. https://www.dragos.com/wp-content/uploads/TRISIS-01.pdf.

[101] *Industry 4.0 and OT security*. https://www.cgi.com/sites/default/files/2020-08/industry-4.0-cybersecurity-methodology-en.pdf.

[102] ControlNet International and Open DeviceNet Vendor Association. *EtherNet/IP Adaptation of CIP Specification*. https://silo.tips/download/ethernet-ip-adaptation-of-cip-specification.

[103] ISSUU. *Industrial edge computing on rise: special report*. https://issuu.com/iebmedia/docs/ieb131-final. 2022.

[104] William Jardine et al. "Senami: Selective non-invasive active monitoring for ics intrusion detection". In: *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*. 2016, pp. 23–34.

[105] Blake Johnson et al. "Attackers deploy new ICS attack framework "TRITON" and cause operational disruption to critical infrastructure". In: *Threat Research Blog* 14 (2017).

[106] Stamatis Karnouskos. "Stuxnet worm impact on industrial cyber-physical system security". In: *IECON 2011-37th Annual Conference of the IEEE Industrial Electronics Society*. IEEE. 2011, pp. 4490–4494.

[107] Masanobu Katagi, Shiho Moriai, et al. "Lightweight cryptography for the internet of things". In: *sony corporation* 2008 (2008), pp. 7–10.

[108] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2020.

[109] Anastasis Keliris and Michail Maniatakos. "ICSREF: A framework for automated reverse engineering of industrial control systems binaries". In: *arXiv preprint arXiv:1812.03478* (2018).

[110] Manish Kesarwani et al. "Model extraction warning in mlaas paradigm". In: *Proceedings of the 34th Annual Computer Security Applications Conference*. 2018, pp. 371–380.

[111] Ilan Komargodski and Eylon Yogev. "Another step towards realizing random oracles: non-malleable point obfuscation". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2018, pp. 259–279.

[112] Oliver Kosut et al. "Malicious data attacks on smart grid state estimation: Attack strategies and countermeasures". In: *2010 first IEEE international conference on smart grid communications*. IEEE. 2010, pp. 220–225.

[113] Aleksandrina Kovacheva. "Efficient code obfuscation for Android". In: *International Conference on Advances in Information Technology*. Springer. 2013, pp. 104–119.

[114] Eduard Kovacs. *Hundreds of ICS Vulnerabilities Disclosed in First Half of 2022*. https://www.securityweek.com/hundreds-ics-vulnerabilities-disclosed-first-half-2022.

[115] Maryna Krotofil and Dieter Gollmann. "Industrial control systems security: What is happening?" In: *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE. 2013, pp. 670–675.

[116] Robert M Lee, Michael J Assante, and Tim Conway. "German steel mill cyber attack". In: *Industrial Control Systems* 30.62 (2014), pp. 1–15.

[117] Taesung Lee et al. "Defending against model stealing attacks using deceptive perturbations". In: *arXiv preprint arXiv:1806.00054* (2018).

[118] Jinwen Liang et al. "Efficient and privacy-preserving decision tree classification for health monitoring systems". In: *IEEE Internet of Things Journal* 8.16 (2021), pp. 12528–12539.

[119] Hui Lin et al. "Adapting bro into scada: building a specification-based intrusion detection system for the dnp3 protocol". In: *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*. 2013, pp. 1–4.

[120] Ondrej Linda, Todd Vollmer, and Milos Manic. "Neural network based intrusion detection system for critical infrastructures". In: *2009 international joint conference on neural networks*. IEEE. 2009, pp. 1827–1834.

[121] Maria B Line et al. "Targeted attacks against industrial control systems: Is the power industry prepared?" In: *Proceedings of the 2nd Workshop on Smart Energy Grid Security*. 2014, pp. 13–22.

[122] Lin Liu et al. “Towards practical privacy-preserving decision tree training and evaluation in the cloud”. In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 2914–2929.

[123] Yao Liu, Peng Ning, and Michael K Reiter. “False data injection attacks against state estimation in electric power grids”. In: *ACM Transactions on Information and System Security (TISSEC)* 14.1 (2011), pp. 1–33.

[124] Oscar Ljungkrantz et al. “Formal specification and verification of industrial control logic components”. In: *IEEE Transactions on Automation Science and Engineering* 7.3 (2009), pp. 538–548.

[125] LogRythm. *What is ICS Security? How to Defend Against Attacks*. https://logrhythm.com/blog/what-is-ics-security-how-to-defend-against-attacks/. 2022.

[126] Benjamin Lynn, Manoj Prabhakaran, and Amit Sahai. “Positive results and techniques for obfuscation”. In: *International conference on the theory and applications of cryptographic techniques*. Springer. 2004, pp. 20–39.

[127] Munir Majdalawieh, Francesco Parisi-Presicce, and Duminda Wijesekera. “DNPSec: Distributed network protocol version 3 (DNP3) security framework”. In: *Advances in Computer, Information, and Systems Sciences, and Engineering*. Springer, 2007, pp. 227–234.

[128] Cyntia Vargas Martinez and Birgit Vogel-Heuser. “A host intrusion detection system architecture for embedded industrial devices”. In: *Journal of the Franklin Institute* 358.1 (2021), pp. 210–236.

[129] Aditya P Mathur and Nils Ole Tippenhauer. “SWaT: A water treatment testbed for research and training on ICS security”. In: *2016 international workshop on cyber-physical systems for smart water networks (CySWater)*. IEEE. 2016, pp. 31–36.

[130] Yusuke Matsuoka and Atsuko Miyaji. “Revisited Diffusion Analysis of Salsa and ChaCha”. In: *2018 International Symposium on Information Theory and Its Applications (ISITA)*. IEEE. 2018, pp. 452–456.

[131] Kerry McKay et al. *Report on lightweight cryptography*. Tech. rep. National Institute of Standards and Technology, 2016.

[132] Stephen McLaughlin. “On dynamic malware payloads aimed at programmable logic controllers”. In: *6th USENIX Workshop on Hot Topics in Security (HotSec 11)*. 2011.

[133] Stephen McLaughlin and Patrick McDaniel. "SABOT: specification-based payload generation for programmable logic controllers". In: *Proceedings of the 2012 ACM conference on Computer and communications security*. 2012, pp. 439–449.

[134] Gordon McMillan. *Socket Programming HOW TO*. https://docs.python.org/3/howto/sockets.html. [Online; accessed 15-February-2023]. 2023.

[135] Machine Metrics. *Enabling Industrial Automation with Ethernet/IP*. https://www.machinemetrics.com/connectivity/protocols/ethernet-ip.

[136] Stevan A Milinković and Ljubomir R Lazić. "Industrial PLC security issues". In: *2012 20th Telecommunications Forum (TELFOR)*. IEEE. 2012, pp. 1536–1539.

[137] Robert Mitchell and Ray Chen. "Behavior rule specification-based intrusion detection for safety critical medical cyber physical systems". In: *IEEE Transactions on Dependable and Secure Computing* 12.1 (2014), pp. 16–30.

[138] Akito Monden et al. "A practical method for watermarking java programs". In: *Proceedings 24th Annual International Computer Software and Applications Conference. COMPSAC2000*. IEEE. 2000, pp. 191–197.

[139] Mohamad-Houssein Monzer et al. "Model-based IDS design for ICSs". In: *Reliability Engineering & System Safety* (2022), p. 108571.

[140] Thomas Morris, Rayford Vaughn, and Yoginder Dandass. "A retrofit network intrusion detection system for MODBUS RTU and ASCII industrial control systems". In: *2012 45th Hawaii International Conference on System Sciences*. IEEE. 2012, pp. 2338–2345.

[141] Andreas Moser, Christopher Kruegel, and Engin Kirda. "Exploring multiple execution paths for malware analysis". In: *2007 IEEE Symposium on Security and Privacy (SP'07)*. IEEE. 2007, pp. 231–245.

[142] Jasvir Nagra and Christian Collberg. *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Pearson Education, 2009.

[143] Jasvir Nagra and Christian Collberg. *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Pearson Education, 2009. Chap. 1, pp. 26–27.

[144] Edgar Namoca. *Targeted Attacks on Industrial Control Systems*. https://westoahu.hawaii.edu/cyber/ics-cybersecurity/ics-weekly-summaries/targeted-attacks-on-industrial-control-systems/.

[145] txOne Networks. *Cybersecurity Report 2021: TXOne Networks Publishes In-Depth Analysis of Vulnerabilities Affecting Industrial Control Systems*. https://www.txone.com/news/cybersecurity-report-2021-txone-networks-publishes-in-depth-analysis-of-vulnerabilities-affecting-industrial-control-systems/. 2022.

[146] Nicolas Nicolaou et al. "Reducing vulnerability to cyber-physical attacks in water distribution networks". In: *2018 international workshop on cyber-physical systems for smart water networks (CySWater)*. IEEE. 2018, pp. 16–19.

[147] Yoav Nir and Adam Langley. *ChaCha20 and Poly1305 for IETF Protocols*. Tech. rep. 2018.

[148] Ryo Nishimaki. "How to watermark cryptographic functions". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2013, pp. 111–125.

[149] Patrick Nohe. *TLS 1.3: A Complete Overview*. https://www.thesslstore.com/blog/tls-1-3-everything-possibly-needed-know/. 2019.

[150] Philip O'Kane, Sakir Sezer, and Kieran McLaughlin. "Obfuscation: The hidden malware". In: *IEEE Security & Privacy* 9.5 (2011), pp. 41–47.

[151] Eric Olson. *What is the most popular industrial network protocol?* https://insights.globalspec.com/article/11936/what-is-the-most-popular-industrial-network-protocol. 2019.

[152] Soham Pal et al. "A framework for the extraction of deep neural networks by leveraging public data". In: *arXiv preprint arXiv:1905.09165* (2019).

[153] Dean Parsons. *The State of ICS/OT Cybersecurity in 2022 and Beyond*. https://www.nozominetworks.com/downloads/US/SANS-Survey-2022-OT-ICS-Cybersecurity-Nozomi-Networks.pdf. 2022.

[154] Vern Paxson. "Bro: a system for detecting network intruders in real-time". In: *Computer networks* 31.23-24 (1999), pp. 2435–2463.

[155] Barak Perelman. *ICS Environments: Insecure by Design*. https://www.securityweek.com/ics-environments-insecure-design.

[156] Dale Peterson. *PLC's: Insecure By Design v. Vulnerabilities*. https://dale-peterson.com/2011/08/02/plcs-insecure-by-design-v-vulnerabilities/. 2011.

[157] Dale Peterson. *Will CISA recommend securing industrial control systems?* https://www.industrialcybersecuritypulse.com/regulations/will-cisa-recommend-securing-industrial-control-systems/. 2022.

[158] Frank Petruzella. *Programmable logic controllers*. McGraw-Hill, Inc., 2004.

[159] Sasanka Potluri, Christian Diedrich, and Girish Kumar Reddy Sangala. "Identifying false data injection attacks in industrial control systems using artificial neural networks". In: *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE. 2017, pp. 1–8.

[160] Nick Powers. *Industrial Protocols Comparison: Fieldbus vs Ethernet More*. https://www.arrow.com/en/research-and-events/articles/industrial-connectivity-protocols. 2016.

[161] Herbert Prähofer, Christian Wirth, and Richard Berger. "Reverse engineering and visualization of the reactive behavior of plc applications". In: *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE. 2013, pp. 564–571.

[162] Mila Dalla Preda et al. "Opaque predicates detection by abstract interpretation". In: *International Conference on Algebraic Methodology and Software Technology*. Springer. 2006, pp. 81–95.

[163] *Programming Introduction With Arduino PLC IDE*. https://docs.arduino.cc/software/plc-ide/tutorials/plc-programming-introduction.

[164] *Python 3.8.2*. https://www.python.org/downloads/release/python-382/. 2020.

[165] Syed Ali Qasim, Juan Lopez, and Irfan Ahmed. "Automated reconstruction of control logic for programmable logic controller forensics". In: *International Conference on Information Security*. Springer. 2019, pp. 402–422.

[166] Erwin Quiring, Daniel Arp, and Konrad Rieck. "Forgotten siblings: Unifying attacks on machine learning and digital watermarking". In: *2018 IEEE European symposium on security and privacy (EuroS&P)*. IEEE. 2018, pp. 488–502.

[167] Muhammad Haris Rais et al. "JTAG-based PLC memory acquisition framework for industrial control systems". In: *Forensic Science International: Digital Investigation* 37 (2021), p. 301196.

[168] R Nirmala Devi Rajesh and C Selvalakshmi. "NIDS: network intrusion detection system with deceptive virtual hosts for industrial control networks". In: *Scientific Engineering and Applied Science* 1 (2015), pp. 255–263.

[169] *Raspberry Pi 3 Model B*. https://www.raspberrypi.com/products/raspberry-pi-3-model-b/.

[170] *Raspberry Pi 4 Tech Specs*. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.

[171] Kapersky Lab Global Research and Analysis Team. *Energetic Bear — Crouching Yeti*. https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/08080817/EB-YetiJuly2014-Public.pdf.

[172] Martin Roesch et al. "Snort: Lightweight intrusion detection for networks." In: *Lisa*. Vol. 99. 1. 1999, pp. 229–238.

[173] Alexander Romanovsky and Fuyuki Ishikawa. *Trustworthy cyber-physical systems engineering*. CRC Press, 2016.

[174] Ron Ross et al. "Managing risk from information systems an organizational perspective". In: (2008).

[175] Kevin A Roundy and Barton P Miller. "Binary-code obfuscations in prevalent packer tools". In: *ACM Computing Surveys (CSUR)* 46.1 (2013), pp. 1–32.

[176] Steven Salzberg et al. "Decision trees for automated identification of cosmic-ray hits in Hubble Space Telescope images". In: *Publications of the Astronomical Society of the Pacific* 107.709 (1995), p. 279.

[177] GPH Sandaruwan, PS Ranaweera, and Vladimir A Oleshchuk. "PLC security and critical infrastructure protection". In: *2013 IEEE 8th International Conference on Industrial and Information Systems*. IEEE. 2013, pp. 81–85.

[178] Sebastian Schrittwieser et al. "Protecting software through obfuscation: Can it keep pace with progress in code analysis?" In: *ACM Computing Surveys (CSUR)* 49.1 (2016), pp. 1–37.

[179] Abraham Serhane et al. "Programmable logic controllers based systems (PLC-BS): Vulnerabilities and threats". In: *SN Applied Sciences* 1.8 (2019), pp. 1–12.

[180] Monirul I Sharif et al. "Impeding Malware analysis using conditional code obfuscation." In: *NDSS*. 2008.

[181] BK Sharma, RP Agarwal, and Raghuraj Singh. "An efficient software watermark by equation reordering and fdos". In: *Proceedings of the International Conference on Soft Computing for Problem Solving (SocProS 2011) December 20-22, 2011*. Springer. 2012, pp. 735–745.

[182] KLS Sharma. *Overview of industrial process automation*. Elsevier, 2016.

[183] Craig Silverstein and Stuart M Shieber. "Predicting individual book use for off-site storage using decision trees". In: *The Library Quarterly* 66.3 (1996), pp. 266–293.

[184] Joseph Slowik. "Evolution of ICS attacks and the prospects for future disruptive events". In: *Threat Intelligence Centre Dragos Inc* (2019).

[185] Murugiah Souppaya and Karen Scarfone. *Guide to data-centric system threat modeling*. Tech. rep. National Institute of Standards and Technology, 2016.

[186] Keith Stouffer, Joe Falco, Karen Scarfone, et al. "Guide to industrial control systems (ICS) security". In: *NIST special publication* 800.82 (2011), pp. 16–16.

[187] Bjorn De Sutter et al. "Instruction set limitation in support of software diversity". In: *International Conference on Information Security and Cryptology*. Springer. 2008, pp. 152–165.

[188] Raymond KH Tai et al. "Privacy-preserving decision trees evaluation via linear functions". In: *European Symposium on Research in Computer Security*. Springer. 2017, pp. 494–512.

[189] Sadik Tamboli et al. "Implementation of Modbus RTU and Modbus TCP communication using Siemens S7-1200 PLC for batch process". In: *2015 international conference on smart technologies and management for computing, communication, controls, energy and materials (ICSTM)*. IEEE. 2015, pp. 258–263.

[190] André Teixeira et al. "Cyber-secure and resilient architectures for industrial control systems". In: *Smart Grid Security*. Elsevier, 2015, pp. 149–183.

[191] Vishal A Thakor, Mohammad Abdur Razzaque, and Muhammad RA Khandaker. "Lightweight cryptography algorithms for resource-constrained IoT devices: A review, comparison and research opportunities". In: *IEEE Access* 9 (2021), pp. 28177–28193.

[192] Olivier Thonnard et al. “Industrial espionage and targeted attacks: Understanding the characteristics of an escalating threat”. In: *International workshop on recent advances in intrusion detection*. Springer. 2012, pp. 64–85.

[193] Florian Tramèr et al. “Stealing Machine Learning Models via Prediction {APIs}”. In: *25th USENIX security symposium (USENIX Security 16)*. 2016, pp. 601–618.

[194] Patrick P Tsang and Sean W Smith. “YASIR: A low-latency, high-integrity security retrofit for legacy SCADA systems”. In: *IFIP International Information Security Conference*. Springer. 2008, pp. 445–459.

[195] Anselme Tueno, Yordan Boev, and Florian Kerschbaum. “Non-interactive private decision tree evaluation”. In: *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer. 2020, pp. 174–194.

[196] David Urbina et al. “Attacking fieldbus communications in ICS: Applications to the SWaT testbed”. In: *Proceedings of the Singapore Cyber-Security Conference (SG-CRC) 2016*. IOS Press. 2016, pp. 75–89.

[197] Maxime Fabian Veit. *ICS protocol dissectors for signature-based NIDS*. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/ICS/Masterarbeit_ICS_protocol_dissectors_for_signature_based_NIDS.pdf?__blob=publicationFile&v=7.

[198] Chenxi Wang et al. “Protection of software-based survivability mechanisms”. In: *2001 International Conference on Dependable Systems and Networks*. IEEE. 2001, pp. 193–202.

[199] Haroon Wardak, Sami Zhioua, and Ahmad Almulhem. “PLC access control: a security analysis”. In: *2016 World Congress on Industrial Control Systems Security (WCICSS)*. IEEE. 2016, pp. 1–6.

[200] Hoeteck Wee. “On obfuscating point functions”. In: *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*. 2005, pp. 523–532.

[201] Joe Weiss. “Industrial Control System (ICS) cyber security for water and wastewater systems”. In: *Securing Water and Wastewater Systems*. Springer, 2014, pp. 87–105.

[202] Daniel Wichs and Giorgos Zirdelis. "Obfuscating compute-and-compare programs under LWE". In: *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2017, pp. 600–611.

[203] Wikipedia. *CANopen*. https://en.wikipedia.org/wiki/DeviceNet.

[204] Wikipedia. *ControlNet*. https://en.wikipedia.org/wiki/ControlNet.

[205] Wikipedia. *DeviceNet*. https://en.wikipedia.org/wiki/DeviceNet.

[206] Wikipedia. *Duqu*. https://en.wikipedia.org/wiki/Duqu.

[207] Wikipedia. *EtherCAT*. https://en.wikipedia.org/wiki/EtherCAT.

[208] Wikipedia. *EtherNet/IP*. https://en.wikipedia.org/wiki/EtherNet/IP.

[209] Wikipedia. *Modbus*. https://en.wikipedia.org/wiki/Modbus.

[210] Wikipedia. *Profibus*. https://en.wikipedia.org/wiki/Profibus.

[211] Wikipedia. *Profinet*. https://en.wikipedia.org/wiki/Profinet.

[212] Andrew K Wright, John A Kinast, and Joe McCarty. "Low-latency cryptographic protection for SCADA communications". In: *International Conference on Applied Cryptography and Network Security*. Springer. 2004, pp. 263–277.

[213] David J Wu et al. "Privately Evaluating Decision Trees and Random Forests." In: *Proc. Priv. Enhancing Technol.* 2016.4 (2016), pp. 335–355.

[214] Le Xie, Yilin Mo, and Bruno Sinopoli. "False data injection attacks in electricity markets". In: *2010 First IEEE International Conference on Smart Grid Communications*. IEEE. 2010, pp. 226–231.

[215] Dongpeng Xu, Jiang Ming, and Dinghao Wu. "Generalized dynamic opaque predicates: A new control flow obfuscation method". In: *International Conference on Information Security*. Springer. 2016, pp. 323–342.

[216] W Yew. "PLC Device Security–Tailoring Needs". In: *White Paper, SANS Institute, Bethesda, Maryland (sansorg. egnyte. com/dl/aN9oVirLPG)* (2021).

[217] Maki Yoshida and Toru Fujiwara. "Toward digital watermarking for cryptographic data". In: *IEICE transactions on fundamentals of electronics, communications and computer sciences* 94.1 (2011), pp. 270–272.

[218] Maki Yoshida, Shigeo Mitsunari, and Toru Fujiwara. "The vector decomposition problem". In: *IEICE transactions on fundamentals of electronics, communications and computer sciences* 93.1 (2010), pp. 188–193.

[219] Ilsun You and Kangbin Yim. "Malware obfuscation techniques: A brief survey". In: *2010 International conference on broadband, wireless computing, communication and applications*. IEEE. 2010, pp. 297–300.

[220] Mohammed Bani Younis and Georg Frey. "Formalization and Visualization of Non-binary PLC Programs". In: *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE. 2005, pp. 8367–8372.

[221] Muhammed Ali Yurdagul and Husrev Taha Sencar. "BLEKeeper: Response Time Behavior Based Man-In-The-Middle Attack Detection". In: *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2021, pp. 214–220.

[222] Kim Zetter. *Meet 'Flame,' The Massive Spy Malware Infiltrating Iranian Computers*. https://www.wired.com/2012/05/flame/.

[223] Xiaolu Zhang et al. "Android application forensics: A survey of obfuscation, obfuscation detection and deobfuscation techniques and their impact on investigations". In: *Forensic Science International: Digital Investigation* 39 (2021), p. 301285.

[224] Huadi Zheng et al. "Bdpl: A boundary differentially private layer against machine learning model extraction attacks". In: *European Symposium on Research in Computer Security*. Springer. 2019, pp. 66–83.

[225] Yifeng Zheng, Huayi Duan, and Cong Wang. "Towards secure and efficient outsourcing of machine learning classification". In: *European Symposium on Research in Computer Security*. Springer. 2019, pp. 22–40.

[226] William Zhu, Clark Thomborson, and Fei-Yue Wang. "A survey of software watermarking". In: *International Conference on Intelligence and Security Informatics*. Springer. 2005, pp. 454–458.

[227] Xiaotong Zhuang et al. "Hardware assisted control flow obfuscation for embedded processors". In: *Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems*. 2004, pp. 292–302.

[228] Lukas Zobernig, Steven D Galbraith, and Giovanni Russello. "When are opaque predicates useful?" In: *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE. 2019, pp. 168–175.