RESEARCH-ARTICLE

Mitigation of Cyber-physical Attacks in Industry 4.0 using Secure Function Blocks

STEPH WU, University of Auckland, Auckland, AUK, New Zealand

NATHAN ALLEN, Auckland University of Technology, Auckland, AUK, New Zealand

ALEX BAIRD, University of Auckland, Auckland, AUK, New Zealand

HAMMOND A PEARCE, UNSW Sydney, Sydney, NSW, Australia

PARTHA S ROOP, University of Auckland, Auckland, AUK, New Zealand

Open Access Support provided by:

UNSW Sydney

University of Auckland

Auckland University of Technology

Published: 28 September 2025

Citation in BibTeX format

MEMOCODE '25: International Symposium on Formal Methods and Models for System Design
September 28 - October 3, 2025
Taipei, Taiwan

Conference Sponsors:
SIGDA
SIGBED

https://doi.org/10.1145/3742875.3754685
ISBN: 9798400719943

Mitigation of Cyber-physical Attacks in Industry 4.0 using Secure Function Blocks

Steph Wu
qwu536@aucklanduni.ac.nz
University of Auckland
Auckland, New Zealand

Nathan Allen
nathan.allen@aut.ac.nz
Auckland University of Technology
Auckland, New Zealand

Alex Baird
alex.baird@auckland.ac.nz
University of Auckland
Auckland, New Zealand

Hammond Pearce
hammond.pearce@unsw.edu.au
University of New South Wales
Sydney, Australia

Partha Roop
p.roop@auckland.ac.nz
University of Auckland
Auckland, New Zealand

Abstract

As Industry 4.0 drives the Fourth Industrial Revolution, Cyber-Physical Systems (CPSs) have become central to industrial automation. These systems integrate software with physical processes, significantly improving the efficiency and adaptability. However, this integration also expands the attack surface, exposing systems to Cyber-Physical-attacks (CP-attacks) that can target either the computational components, physical devices, or both. The impact of such attacks can be catastrophic, ranging from system disruption to physical damage. Although numerous techniques have been developed to detect and mitigate these threats, industrial standards are often not incorporated into the design of these methods. This limits their deployment within the Industry 4.0 systems, where standard compliance is critical.

To this end, we extend IEC 61499, an emerging standard being considered in Industry 4.0, that uses reusable artefacts called function blocks. We formalise the mitigation of CP-attacks using a novel method based on Bi-directional Runtime Enforcement (Bi-RE) using a standards compliant approach called Secure Function Blocks (SFBs). This approach automatically generates the necessary enforcers from a timed specification language called Valued Discrete Timed Automaton (VDTA). We illustrate our approach using the case study of a water treatment system, and highlight the low overhead associated with it. This approach allows for runtime techniques to be applied with minimal changes to many Industry 4.0 applications.

CCS Concepts

• **Computer systems organization → Embedded and cyber-physical systems**; • **Security and privacy → Formal methods and theory of security**; **Embedded systems security**.

MEMOCODE '25, Taipei, Taiwan

ACM ISBN 979-8-4007-1994-3/2025/09
https://doi.org/10.1145/3742875.3754685

Keywords

IEC61499, Cyber-Physical security, Function Blocks, Runtime Enforcement

ACM Reference Format:
Steph Wu, Nathan Allen, Alex Baird, Hammond Pearce, and Partha Roop. 2025. Mitigation of Cyber-physical Attacks in Industry 4.0 using Secure Function Blocks. In *International Symposium on Formal Methods and Models for System Design (MEMOCODE '25), September 28-October 3, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3742875.3754685

1 Introduction

Cyber-Physical-attacks (CP-attacks) are a class of incidents where the attack results in consequences to the physical assets within a Cyber-Physical System (CPS). Examples include the Stuxnet attack on Nuclear infrastructure [16] which destroyed centrifuges, an attack on a German steel mill [18] which caused significant damage to a blast furnace, and the recent attack by "predatory sparrow" on an Iranian steel mill [24]. Given the close relationship between the physical assets and their digital controllers, classical cybersecurity defenses are incapable of detecting and mitigating such attacks.

Formal methods [36] utilise a set of well known mathematical techniques for the specification, verification, and design of safety-critical systems. As security and safety are related and sometimes intertwined, there has been considerable interest in the use of formal methods for security [14]. However, there are limited attempts at formal techniques for CP-attacks in industrial automation, except some prominent exceptions [17, 27]. Both these approaches are inspired by run-time based formal methods for industrial automation. The former approach [17] develops the concept of *secure proxies* that act as a invisible mediator (to the attacker), which is placed between the controller (the Programmable Logic Controller (PLC)) and the physical process (the plant). The proxy can determine if the attacker is changing the controller output, based on pre-defined conditions, and can take evasive action to mitigate the attack at run-time. This approach is also compliant to the IEC 16631-3 standard, and hence is the closet to solving this problem. Figure 1 presents an overview of the secure proxy concept, where each PLC in a distributed control environment has a secure proxy for attack mitigation.

However, this method [17] has some limitations. First, the secure proxy is unidirectional meaning that, like a shield [4], it can only

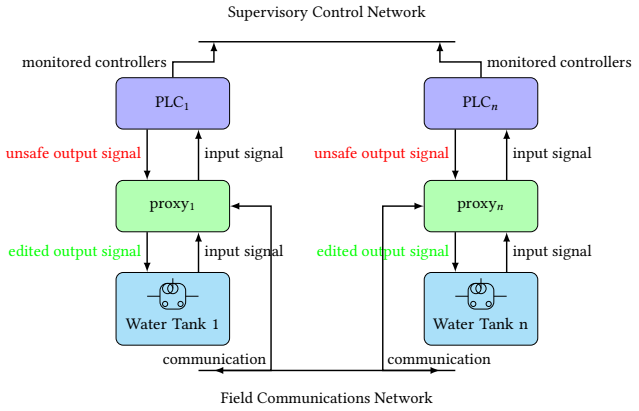

Figure 1: A secure proxy architecture for distributed automation, recreated from [17]

enforce attacks that alter the controller outputs (that influence the actuators) but are not able to deal with attacks that alter the inputs from the plant (attack on sensors). This gap has been remedied in bi-directional run-time enforcement [28], as shown in Figure 2, where an *enforcer* is placed between the plant and controller of the CPS. This enforcer, which is automatically derived from a security policy described as a Valued Discrete Timed Automaton (VDTA) [27], monitors and modifies the input and output signals to ensure that the policy is always satisfied, i.e. the system is kept safe. This framework secures both input and output directions, strengthening CPS security.

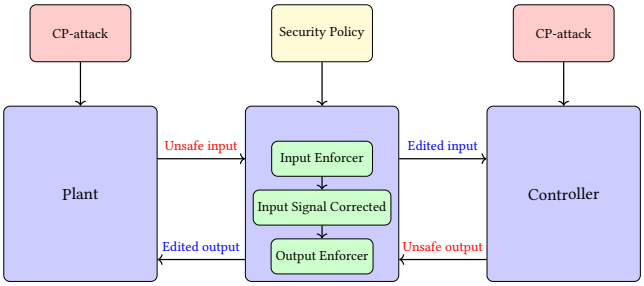

Figure 2: Bi-directional Runtime Enforcement

However, this approach is yet to be applied to any industrial control systems due to a lack of integration of the developed method with any automation standards. A second limitation of Lanotte et al. [17] is that the approach is developed for older PLC standards, which are non-ideal for Industry 4.0, unlike the more recent IEC 61499 standard.

1.1 Industry 4.0 and IEC 61499

Industry 4.0, often referred to as the Fourth Industrial Revolution, represents the ongoing transformation of traditional industries through automation and digitalisation. This shift leverages advanced technologies, including CPSs, to improve production efficiency. CPSs integrate hardware and software within a typically closed-loop structure, linking embedded systems (controllers), and physical processes (plants). However, this integration introduces the challenge of increased system complexity.

To manage this complexity in Industry 4.0, the IEC 61499 Function Blocks standard was introduced [34] to facilitate the modelling and implementation of distributed automation systems. While the earlier IEC 61131-3 [12] focused more on the programming of individual PLCs, IEC 61499 is designed with distributed systems and interoperability in mind. In the IEC 61499 context, systems are composed of various Function Blocks (FBs). Basic Function Blocks (BFBs) capture the logic of a system through Execution Control Charts (ECCs) and algorithms, while Composite Function Blocks (CFBs) allow for hierarchical composition of other FBs.

While IEC 61499 has seen many industrial applications [5, 35] and has been extensively investigated for formalising the semantics [34, 37], verification [3] , and relationships to other standards [11], the issue of securing systems designed using the standard has received scant attention. This is a serious problem.

1.2 Proposed solution

To address this, we propose a technique named Secure Function Blocks (SFBs) which is based on the principles of Bi-directional Runtime Enforcement (Bi-RE) [28] and is standards compliant with the IEC 61499 standard. Security policies are described using a formal policy language called VDTA [27], an extension of Discrete Timed Automaton (DTA) [28] that accommodates valued signals, internal variables, and complex guard conditions. In this work, each policy is compiled individually and multiple policies, if required, must be composed by the designer at runtime. This enhancement ensures compatibility with real-world CPS and industrial systems.

The overall approach for the SFB technique is illustrated in Figure 3. Firstly, security policies are defined as VDTAs using an existing language known as *easy-rte* [27]. These policies are then automatically compiled into our SFBs in IEC 61499 using a developed compiler. The generated SFBs can then be combined with the industrial system (also described in IEC 61499) to create a secured system in IEC 61499 that is now resilient to CP-attacks. Finally, the secured system can be compiled or executed using any IEC 61499 toolchain to allow for the SFBs to be used on a real system.

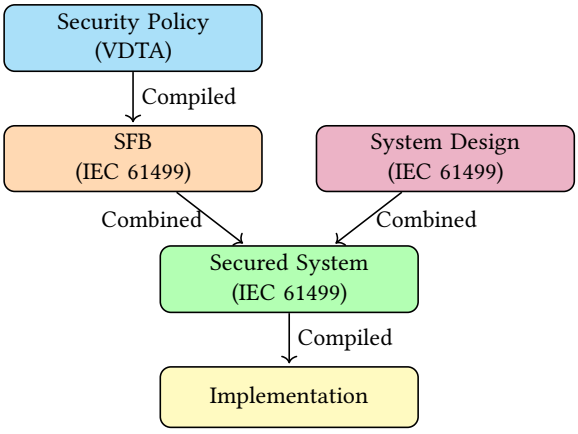

Figure 3: Overview of the approach

As a result, this work has *two key contributions*: (1) Firstly, we propose, for the first time, a technique to define Bi-RE algorithms within IEC 61499 without any modifications to the standard. (2) Secondly, we extend an existing industrial automation benchmark [17] with CP-attacks to include bi-directional attacks, and show that our proposed approach is able to detect and correct such instances. We provide the significance of our findings in Table 1.

The remainder of this paper is organised as follows. In Section 2 we will introduce some preliminary information around the case study (previously proposed in [17]) and formal definitions that are used in this paper. Subsequently, in Section 3, we will use these definitions to describe our SFB approach, showing how Runtime Enforcement (RE) techniques are captured in IEC 61499. Section 4 will then define the attacks and policies used with the case study, while Section 5 will provide their results to illustrate the effectiveness of this approach. Section 6 then covers some of the related work in this area in comparison to our work, while Section 7 provides concluding remarks.

2 Case Study and Preliminaries

To illustrate the suitability of our SFB approach to a realistic industrial application, we build upon the water treatment system introduced by Lanotte et al. for their uni-directional enforcement [17]. Figure 4 shows an overview of this system, which aims to create and store clean water in a tank (T_3) from raw untreated water through a series of tanks, pumps, and valves.

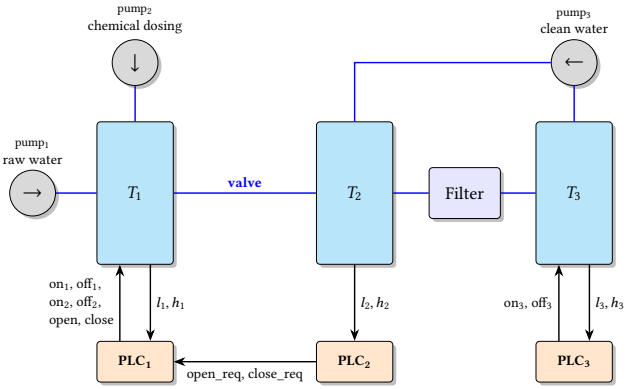

Figure 4: Simplified water treatment system overview, recreated from [17]

The system consists of three separate tanks — T_1, T_2, and T_3. T_1 has two input pumps ($pump_1$ and $pump_2$) to add water into the system, and an outlet valve ($valve$) to feed water into T_2. T_2 is purely passive with no actuators, and serves as a "buffer" for the filtration which passively filters the water and moves it into T_3. If the level in T_2 ever gets too low, it is able to request the valve from T_1 to be opened or closed ($open\_req$ or $close\_req$) to control the inflow. T_3 has a single pump ($pump_3$) that pumps water back to T_2. There are water level sensors for each tank that provide low and high level signals (l_1 and h_1, l_2 and h_2, and l_3 and h_3) respectively. Each tank has an associated controller (a PLC) which receives information about the water level inside its respective tank (l and h) and is able to control various pumps (through on and off commands) and valves ($open$ or $close$) to maintain the functionality of the system.

An implementation of the case study in the IEC 61499 standard is shown in Figure 5. The basic building blocks in this standard are known as BFBs. Here, each tank's dynamics are captured as BFBs with connections between them — red lines for events and blue/grey lines for data ports — representing water inflow/outflow. A controller is created for each tank, with its water level sensor as input and its various control actions as outputs. Additionally, our SFBs are shown between the tanks and their controllers, as will be discussed in this paper.

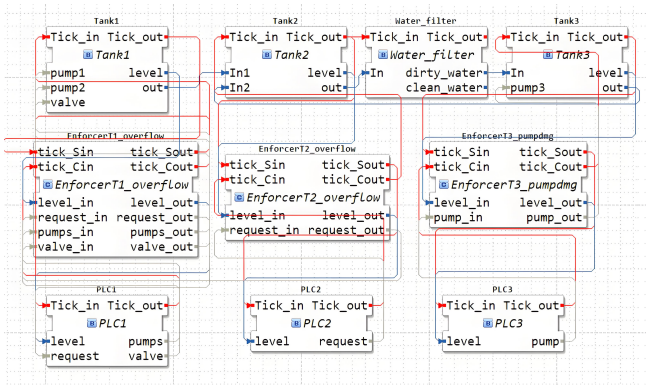

Figure 5: IEC 61499 water treatment system enforcement overview

In order to formalise our approach, we first need to introduce a few preliminary definitions covering the IEC 61499 specification and the policy model for our RE techniques — VDTAs. These definitions are heavily based on previously published definitions for IEC 61499 [31] and RE [27].

Firstly, we will formally define the components of an IEC 61499 design. Each FB has an identical interface which contains its input and output events and variables, along with their associations.

Definition 2.1 (Function Block Interface). A Function Block Interface is a type $\mathcal{I} = (E_I, V_I, \alpha_I, E_O, V_O, \alpha_O)$, where:

- E_I and E_O are the sets of input and output events,
- V_I and V_O are the sets of input and output variables, and
- $\alpha_I \subseteq E_I \times V_I$ and $\alpha_O \subseteq E_O \times V_O$ are the event and variable associations.

Referring back to the water treatment system implementation in Figure 5, let's look at the interface for PLC_2, $\mathcal{I}_{PLC_2}$. Here we have $E_I = \{Tick\_in\}$, $E_O = \{Tick\_out\}$, $V_I = \{level\}$, and $V_O = \{request\}$. The event associations are $\alpha_I = \{(Tick\_in, level)\}$ and $\alpha_O = \{(Tick\_out, request)\}$.

Next, we formally define BFBs, which consist of the interface mentioned above, along with an ECC which contains the execution logic of the block.

Definition 2.2 (Execution Control Chart). An Execution Control Chart is a tuple $\mathcal{E} = (S, s_0, E_I, E_O, V, A, \lambda, T)$, where:

- S denotes a finite set of states,
- $s_0 \in S$ is the initial state,

- E_I and E_O are the set of input and output events,
- V is the set of variables,
- A is a set of algorithms,
- $\lambda : S \rightarrow (A \cup E_O)*$ is the action function which maps states to finite sequences of algorithm executions and/or output event emissions, and
- $T : S \times E_I \times \mathcal{B}(V) \rightarrow S$ denotes the transition function, where $\mathcal{B}(V)$ is a boolean expression over the set of variables V.

Definition 2.3 (Basic Function Block). A Basic Function Block is a tuple $\mathcal{BFB} = (\mathcal{I}, V_L, A, \mathcal{E})$, where:

- $\mathcal{I}$ is an interface according to Definition 2.1,
- V_L denotes a set of local variables,
- A denotes a set of algorithms, and
- $\mathcal{E}$ is an ECC according to Definition 2.2 with the sets of events E_I and E_O coming directly from $\mathcal{I}$, and the set of variables $V = V_I \cup V_O \cup V_L$.

Each of the FBs for the tanks and PLCs in Figure 5 are BFBs, with their interface $\mathcal{I}$ defined like the previous example. For the tanks, the current level is stored as a local variable $V_L = \{v\}$, and their ECCs $\mathcal{E}$ are simple single-state entities ($S = \{s\}$, $s_0 = s$) where the physics is encoded as an algorithm $a \in A$ which updates each time step, i.e. $\lambda(s) = \langle a, Tick_{out} \rangle$.

Finally, for the IEC 61499 definitions, we define CFBs which includes a network made up of other FBs (either BFBs or other CFBs) along with the connections between them.

Definition 2.4 (Composite Function Block). A Composite Function Block is a tuple $\mathcal{CFB} = (\mathcal{I}, F, C_E, C_X)$, where:

- $\mathcal{I}$ is an interface according to Definition 2.1,
- F is a set of FB instances which include an interface according to Definition 2.1, and
- $C_E \subseteq E_S \times E_D$ is a set which contains the event connections, where:
  - $E_S = \left(\bigcup_{f \in F} f.E_O\right) \cup E_I$ is the set of possible source events, and
  - $E_D = \left(\bigcup_{f \in F} f.E_I\right) \cup E_O$ is the set of possible destination events.
- $C_X \subseteq X_S \times X_D$ is a set which contains the variable connections, where:
  - $X_S = \left(\bigcup_{f \in F} f.X_O\right) \cup X_I$ is the set of possible source variables,
  - $X_D = \left(\bigcup_{f \in F} f.X_I\right) \cup X_O$ is the set of possible destination variables, and
  - for any two connections $(s_1, d_1), (s_2, d_2) \in C_X, d_1 \neq d_2$.

The overall system shown in Figure 5 shows the internal logic of a CFB. The interface $\mathcal{I}$ of this CFB would contain any external inputs/outputs, while each FB is declared as part of F. Connections between events (C_E) and variables (C_X) are declared, with examples being $(T_2.Tick\_out, filter.Tick\_in)$ for events and $(T_2.out, filter.in)$ for variables.

In terms of RE preliminaries, we also need to introduce the formal model that is used for the policy definitions. Here, we will define our policies as VDTAs.

Definition 2.5 (Valued Discrete Timed Automaton). A Valued Discrete Timed Automaton is a tuple $\mathcal{A} = (L, l_0, X, V, C, \Sigma, \Theta, F, \Delta)$, where:

- L is a finite non-empty set of locations,
- $l_0 \in L$ is the initial location,
- X is a finite set of discrete clocks,
- V is a set of typed internal variables,
- C is a set of external variables where $C = C_I \cup C_O$, where C_I is the set of input channels and C_O is the set of output channels,
- Σ is a non-empty finite set of actions where an action $a \in \Sigma$ has a signature $sig(a) = (t_0, t_1, ..., t_k)$ which is a set of types of the external variables,
- $\Theta \subseteq \mathcal{D}_V$ is an initial condition which is a computable predicate over V,
- $F \subseteq L$ is the set of accepting locations, and
- Δ is a finite set of transitions with each transition $t \in \Delta$ being a tuple (l, a, c, G, A, l') (also written $l \xrightarrow{a(c), G(V,c), V' := A(V,c)} l'$), where:
  - $l, l' \in L$ are respectively the origin and target locations of the transition,
  - $a \in \Sigma$ is the action and $c = (c_1, ..., c_k)$ is a set of external variables local to the transition,
  - $G = G^D \wedge G^X$ is the transition guard, where:
    - $G^D \subseteq \mathcal{D}_V \times \mathcal{D}_{sig(a)}$ is a computable predicate over internal variables and external variables in $V \cup c$, and
    - G^X is a clock constraint over X defined as a Boolean combinations of constraints of the form $x \bowtie f(V \cup C)$, where $x \in X$, $f(V \cup C)$ is a computable function over internal and external variables, and $\bowtie \in \{<, \leq, =, \geq, >\}$.
  - $A = (A^D, A^X)$ is the assignment of the transition where
    - $A^D : \mathcal{D}_V \times \mathcal{D}_{sig(a)} \rightarrow \mathcal{D}_V$ defines the evolution of internal variables, and
    - $A^X \subseteq X$ is the set of clocks to be reset.

Here, we present an example policy (φ) named `AB5` for a system with an input A and an output B. This policy ensures that *A and B cannot happen simultaneously, A and B alternate starting with an A. B should be true within 5 ticks after A occurs.* As a VDTA this is shown in Figure 6, which we call $\mathcal{A}_\varphi$.

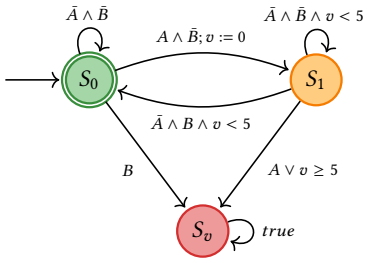

Figure 6: AB5 policy as a Valued Discrete Timed Automaton $\mathcal{A}_\varphi$

Using definition 2.5, we can formalise this property. First, there are three locations $L = \{S_0, S_1, S_v\}$, with the initial location $l_0 = S_0$ and the accepting locations $F = \{S_0\}$. There is a single discrete

clock $X = \{v\}$, and no internal variables $V = \emptyset$. In terms of external variables, $C_I = \{A\}$ and $C_O = \{B\}$.

The transition relation Δ captures the transitions visible in Figure 6. In some cases clocks are reset to 0 (A^X) when transitions are taken, such as when going from S_0 to S_1, which is formally encoded as $A^X = \{v\}$. In other cases, such as the transition from S_1 to S_0, the clocks are used as part of the guard condition. In this case, it can be formally captured by saying $G^X = v < 5$.

The Bi-RE input property, denoted φ_I, is the projection of φ solely over inputs. As a VDTA, the input automaton $\mathcal{A}_{\varphi_I}$ can be obtained from $\mathcal{A}_\varphi$ by ignoring all outputs C_O on each transition. Therefore, for every transition in $\mathcal{A}_\varphi$ there exists a transition in $\mathcal{A}_{\varphi_I}$. Figure 7 depicts the Input VDTA for the AB5 example after this projection over inputs.

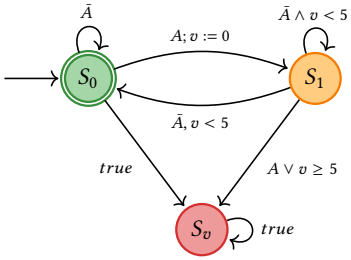

Figure 7: Input Valued Discrete Timed Automaton for $\mathcal{A}_\varphi$

We also need to define the semantics of a VDTA. Here, to capture the valuation of clocks we use $\chi \in \mathbb{N}^X$, with $\chi + 1$ capturing the value after all clocks have ticked, and $\chi[X' \leftarrow 0]$ the case where clocks $X' \in X$ are being reset. The valuation of all variables in V is denoted as $v \in \mathcal{D}_V$.

Definition 2.6 (Semantics of a Valued Discrete Timed Automaton). Given a VDTA $\mathcal{A}$, its semantics is a *timed transition system* $[[\mathcal{A}]] = (Q, q_0, \Gamma, \rightarrow, Q_F, Q_v)$, where:

- $Q = L \times \mathbb{N}^X \times \mathcal{D}_V$ is the set of states of the form $q = (l, \chi, v)$ with $l \in L$ being the location, $\chi \in \mathbb{N}^X$ being the clock valuations, and $v \in \mathcal{D}_V$ being the variable valuations,
- $q_0 = \{(l_0, \chi[X \leftarrow 0], v) \mid \Theta(v) = true\}$ is the set of initial states,
- $\Gamma = \{a(\eta) \mid a \in \Sigma \wedge \eta \in \mathcal{D}_{sig(a)}\}$ is the set of transition labels,
- $\rightarrow \subseteq Q \times \Gamma \times Q$ is the transition relation of the form $(l, \chi, v) \xrightarrow{a(\eta)} (l', \chi', v')$ with $\chi' = (\chi + 1)[A^X \leftarrow 0]$ and $v' = A^D(v, \eta)$ whenever there exists a transition $(l, a, c, G, A, l') \in \Delta$ such that $G^X(\chi + 1) \wedge G^D(v, \eta)$ is satisfied,
- $Q_F = F \times \mathbb{N}^X \times \mathcal{D}_V$ is the set of accepting states, and
- $Q_v \subseteq Q \setminus Q_F$ is the set of violation states where there is no possible transition (or sequence of transitions) back to a state in Q_F.

3 Secure Function Blocks for mitigating Cyber Physical Attacks

Our approach, SFBs, integrates Bi-RE design under the IEC 61499 standard. We will illustrate the concept of SFBs and the methodology using the previously mentioned AB5 example.

3.1 Secure Function Block Architecture

Algorithm 1 presents how a VDTA-based policy can be implemented in IEC 61499 using our SFB approach. An SFB can be considered as a CFB that contains two other BFBs — an input enforcer (F_I) and an output enforcer (F_O).

Algorithm 1 Converting a VDTA policy to an SFB

Input: VDTA policy $\mathcal{A}_\varphi = (L, l_0, X, V, C, \Theta, F, \Delta)$
Output: SFB $\mathcal{SFB} = (\mathcal{I}, F, C_E, C_X)$

$\mathcal{I}.E_I \leftarrow \{tick_{Sin}, tick_{Cin}\}$
$\mathcal{I}.E_O \leftarrow \{tick_{Sout}, tick_{Cout}\}$
$\mathcal{I}.V_I \leftarrow \bigcup_{c \in C} c_{in}$
$\mathcal{I}.V_O \leftarrow \bigcup_{c \in C} c_{out}$
$\mathcal{I}.\alpha_I \leftarrow \bigcup_{c \in C_I} (tick_{Sin}, c_{in}) \cup \bigcup_{c \in C_O} (tick_{Cin}, c_{in})$
$\mathcal{I}.\alpha_O \leftarrow \bigcup_{c \in C_I} (tick_{Sout}, c_{out}) \cup \bigcup_{c \in C_O} (tick_{Cout}, c_{out})$

F_I = generate_enforcer($\mathcal{A}_{\varphi_I}$) ▷*See Section 3.2*
F_O = generate_enforcer($\mathcal{A}_\varphi$) ▷*See Section 3.2*
$F = \{F_I, F_O\}$

$C_E = \{(tick_{Sin}, F_I.tick_{Sin}), (F_I.tick_{Sout}, tick_{Sout})\}$
$C_E = C_E \cup \{(tick_{Cin}, F_O.tick_{Cin}), (F_O.tick_{Cout}, tick_{Cin})\}$

$C_X = \{(F_I.state_{out}, F_O.state_{in})\} \cup \{(F_O.state_{out}, F_I.state_{in})\}$
$C_X = C_X \cup \bigcup_{x \in X} \{(F_I.x_{out}, F_O.x_{in}), (F_O.x_{out}, F_I.x_{in})\}$
$C_X = C_X \cup \bigcup_{c \in C_I} \{(c_{in}, F_I.c_{in})\} \cup \bigcup_{c \in C_I} \{(F_I.c_{out}, c_{out})\}$
$C_X = C_X \cup \bigcup_{c \in C_O} \{(c_{in}, F_O.c_{in})\} \cup \bigcup_{c \in C_O} \{(F_O.c_{out}, c_{out})\}$
$C_X = C_X \cup \bigcup_{c \in C_I} \{(F_I.c_{out}, F_O.c_{in})\}$

return $\mathcal{SFB}$

The algorithm can be seen in four separate parts. Firstly, the interface $\mathcal{I}$ of the SFB is created by looking through the set of external variables C in the VDTA. Copies of each variable are present on both the input (V_I) and output (V_O) sides of the SFB, and these variables are associated with their respective synchronous tick events.

Secondly, enforcer BFBs are created for both the input-projected policy and the original. These BFBs are termed F_I and F_O respectively. Event mappings are straightforward due to the synchronous composition, as the only required mappings are from the external events (E_I and E_O) to their respective internal BFBs.

Finally, variable mappings are performed starting with the sharing of state and clock valuations between the two enforcer blocks. Next, the external input and output variables (C_I and C_O) are mapped to/from the input and output enforcers (F_I and F_O), respectively. Input variables (C_I) are also forwarded from input enforcer (F_I) to output enforcer (F_O).

For the AB5 example, the resulting interface and network of this algorithm are shown in Figure 8 and Figure 9 respectively. Here, in the interface, we can see the synchronous events, the corresponding input and output variables for A and B, and the association between the events and variables.

Figure 9 shows the creation of the two separate input and output enforcers, named InputEF and OutputEF respectively. Additionally, note the mapping of events and variables between the two enforcers, as well as with the overall SFB, and how they correspond to the

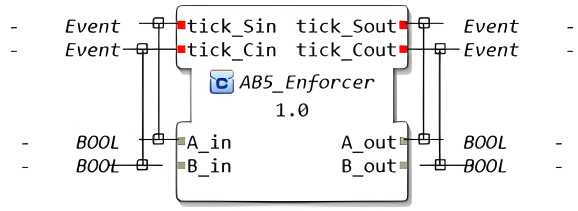

Figure 8: AB5 Secure Function Block Interface

process outlined in Algorithm 1. In particular, the `state` variable is passed between the two enforcers, and is encoded as type `STRING`. The clock variable `v` goes through a similar process to be shared between both enforcers, with its type being `INT`.

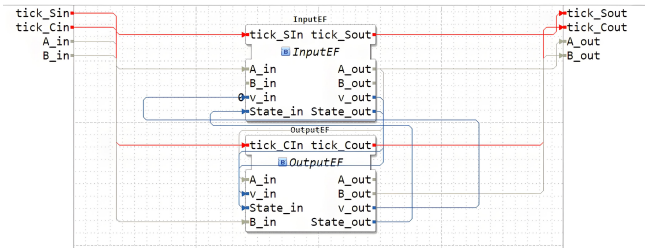

Figure 9: AB5 Secure Function Block Network

3.2 Input and Output Enforcers

As shown in Algorithm 1, two BFBs are generated from the VDTA policy to capture the input and output enforcers. Like with all BFBs, their logic is implemented through the use of algorithms and ECCs. The input enforcer is generated based on the input VDTA ($\mathcal{A}_{\varphi_I}$), while the output enforcer uses the complete policy. In both cases, their interfaces are similar and contain all the external variables of C, the synchronous tick events, and the shared data for state and clock variables.

The states within the ECC for an enforcer (S) consists of the same set of locations L in the policy, excluding any violation locations. This exclusion is due to the fact that in a correct enforcer, violations should never occur. At each of these states ($s \in S$) in the ECC, a corresponding enforcement algorithm is defined (in $a_s \in A$) and executed, along with the associated event ($\lambda(s) = \langle a_s, tick_{out} \rangle$). An additional initial state (i.e. s_0) named `START` is created which facilitates the sharing of state between the two enforcers. As an example, the resulting ECC for the input enforcer (`InputEF`) is shown in Figure 10.

The set of ECC transitions T are exclusively established between the `START` state and each other state in S. The shared state variable between the two enforcers is an input to each enforcer (`State_in`) and is then used to determine its operation. Outgoing transitions from `START` are guarded by the value of this state, while the return transitions are automatically taken (represented by a guard of `1`). It is important to note that the output value of the state (`State_out`) will be set inside the algorithm associated with the state.

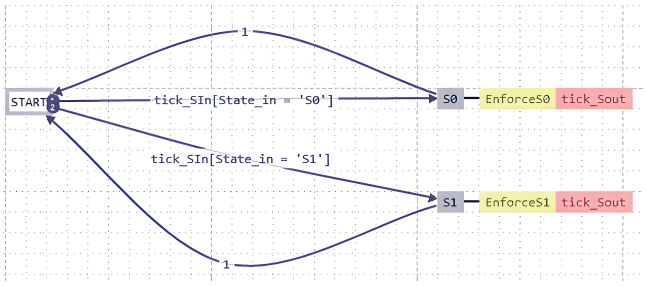

Figure 10: AB5 Input Enforcer Execution Control Chart

Figure 10 refers back to the `AB5` example. Starting from the `START` state, If the value of `State_in` was `S0` then the ECC would transition to `S0`, execute its associated algorithm (`EnforceS0`), emit the `tick_Sout` event, and then return back to `START` to await the next cycle. The same approach happens for the state `S1`.

The BFB enforcement algorithms executing on each state are similar to those previously proposed in the literature [27, 28], using the semantics described earlier. Algorithms 2 and 3 show this logic for the input and enforcer respectively.In both cases, the state q and time t are stored at the BFB level, rather than the algorithm level, due to the way the IEC 61499 standard handles algorithm execution.

Algorithm 2 SFB Input Enforcement Algorithm

Parameter: Input policy $[[\mathcal{A}_{\varphi_I}]] = (Q, q_0, \Gamma_I, \rightarrow_I, Q_F, Q_v)$

1: **if** $\exists \sigma' \in \Sigma^* : q \xrightarrow{x_t \cdot \sigma'}_I q' \wedge q' \in Q_F$ **then**
2: $\quad x'_t \leftarrow x_t$
3: **else**
4: $\quad x'_t \leftarrow \text{sel-edit}_{\mathcal{A}_{\varphi_I}}(q)$
5: **end if**
6: **return** x'_t

Algorithm 3 SFB Output Enforcement Algorithm

Parameter: Output policy $[[\mathcal{A}_{\varphi}]] = (Q, q_0, \Gamma, \rightarrow, Q_F, Q_v)$

1: **if** $\exists \sigma' \in \Sigma^* : q \xrightarrow{(x_t, y_t) \cdot \sigma'} q' \wedge q' \in Q_F$ **then**
2: $\quad y'_t \leftarrow y_t$
3: **else**
4: $\quad y'_t \leftarrow \text{sel-edit}_{\mathcal{A}}(q)$
5: **end if**
6: $q \leftarrow q''$ where $q \xrightarrow{(x_t, y'_t)} q'' \wedge q'' \notin Q_v$
7: $t \leftarrow t + 1$
8: **return** y'_t

The operation of the enforcers is quite similar. Firstly, they check if the given input (x_t) or output (y_t) is admissible based on if there is any possible path ($\sigma' \in \Sigma^*$) to an accepting state in the future ($q' \in Q_F$). If true, the input or output values can be used as they are and do not need modifying. Otherwise, the value needs to be modified through a *select* function, which returns the appropriate

edit action for the current state. Finally, the output enforcer is in charge of updating the state (q') and time (t') of the system.

Appendix A shows the generated code for these enforcers in the AB5 example.

4 Attack Scenarios and Mitigation Policies

Using the case study introduced in Section 2, we will introduce five potential attacks on the system as an example for the SFB approach.

- *T1 overflow*: The attacker tries to overflow T1 by activating the incoming pumps (pump1 or pump2) when the tank is full.
- *T2 overflow*: The attacker tries to overflow T2 by opening the incoming valve (valve) when the tank is full.
- *T3 pump damage*: The attacker tries to damage the pump in T3 by activating it (pump3) when the tank is empty.
- *T2 combination attack*: The attacker simultaneously targets the water level sensor (level2) of T2 and the incoming valve (valve) to overflow the tank.
- *T3 pump burnout*: The attacker aims to damage the pump by quickly turning the pump on and off.

To capture and mitigate each of these attacks, we define policies (P_1 through P_5) which will be enforced by our SFBs. These attacks and their associated policies are summarised in Table 1. Our objective here is to ensure the system's actions prioritize safety, even if it means delaying the water treatment process, in order to safeguard the system from potential real-world damage. We will now expand on each of these properties in more detail.

4.1 Policy P_1: Mitigating T1 overflow

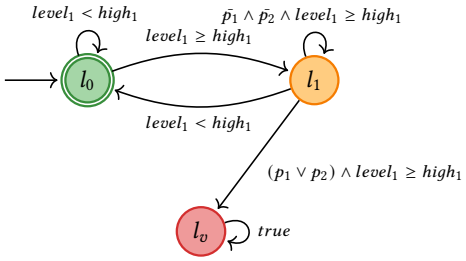

Figure 11: Policy P_1 captured as a VDTA

Through an attack, T1 could be caused to overflow by enabling the inflow pumps while the water is already at a high level. Figure 11 shows a policy to prevent this, captured as a VDTA, which switches between two active locations (l_0 and l_1) which correspond to the water level being below and above the high marker. If either of the incoming pumps (denoted p_1 and p_2) are enabled while in location l_1 then this is a violation because the tank would be at risk of overflowing, and so control switches to location l_v.

4.2 Policy P_2: Mitigating T2 overflow

A similar attack could be performed on T2 in an attempt to cause it to overflow. However, in this case the tank does not have any actuators directly attached to it and instead its inflow is controlled through a request signal to PLC1, the controller for T1. Figure 12 shows how a policy to prevent this can be captured as a VDTA, in a similar architecture to P_1, which ensures that the request signal should never be set when the water level is already high.

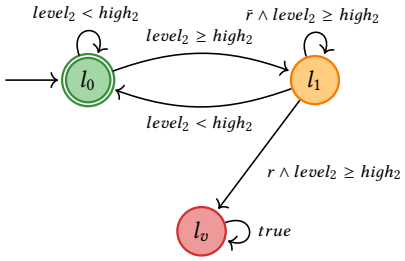

Figure 12: Policy P_2 captured as a VDTA

4.3 Policy P_3: Mitigating T3 pump damage

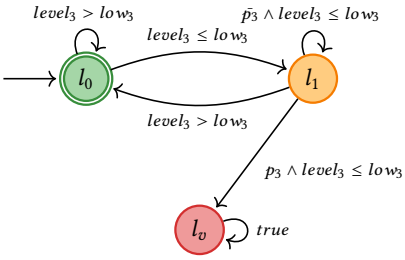

Figure 13: Policy P_3 captured as a VDTA

Conversely to the previous attacks, which tried to cause overflow, an attack on T3 could be performed which aims to enable the output pump even if the tank is empty, which would cause device failure. The VDTA to capture a policy which detects this attack is shown in Figure 13 which compares the tank level ($level_3$) to its low reference (low_3), rather than the high reference that was previously used. Here, if the pump (p_3) was ever requested to be enabled while the water level was below the low threshold, then the VDTA would transition to the violation location.

4.4 Policy P_4: Mitigating T2 combination attack

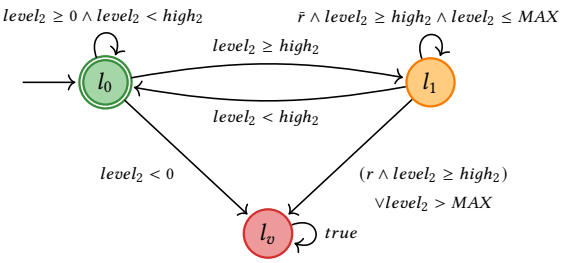

Figure 14: Policy P_4 captured as a VDTA

The advantage of the Bi-RE is that is not only able to prevent attacks that impact the controller outputs, but also detect and prevent attacks on the sensor side of the system. We introduce an

Table 1: Attack and Policy Summary

ID	Attack Description	Policy Description	Attack Focus	Policy Type	Source
P_1	`T1` overflow: Turning on `pump1` or `pump2` despite the tank being full	If `T1` is full, `pump1` and `pump2` should be off	Plant	Uni-directional	Re-created from [17]
P_2	`T2` overflow: Opening `valve` when `T2` is full	If `T2` is full, `valve` should be closed	Plant	Uni-directional	Re-created from [17]
P_3	`T3` pump damage: Activating `pump3` when `T3` is empty	If `T3` is empty, `pump3` should be off	Plant	Uni-directional	Re-created from [17]
P_4	`T2` combination attack: Targeting `level2` and `valve` simultaneously	Out-of-range level2 $\Rightarrow$ tank considered full; If full, `valve` should be closed	Plant + Controller	Bi-directional	Proposed
P_5	`T3` pump burnout: Turning pump off while it is activating	If `T3` is not empty, pumping should last at least 5 ticks	Plant	Time-based	Proposed

attack which is bi-directional and combines multiple of our earlier attacks, simultaneously attacking both the plant and the controller. The policy P_4, shown in Figure 14, is checking both for a water level sensor attack, along with the overflow attack of P_2. In the case where we detect a violation on an input signal, the question becomes what is the correct action to take? Therefore, we consider the water level signal to its maximum value in order to elminate the risk of overflow — a critical failure — at the risk of system performance degradation.

There are two things to note with this policy. Firstly, the policy involves enforcement actions that would need to be applied to both the input and output sides of the problem — a true bi-directional enforcement. Secondly, it shows a more complex example that can be used for comparison against the previously introduced examples and provides the highest level of security against attack.

4.5 Policy P_5: Avoiding `T3` pump burnout

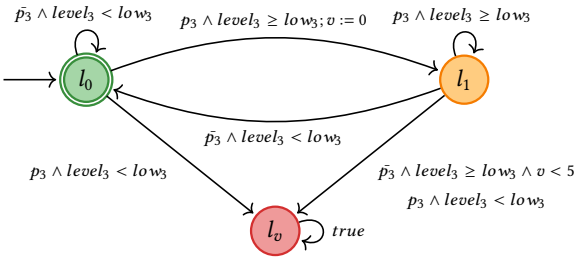

Figure 15: Policy P_5 captured as a VDTA

Finally, we present an attack designed to quickly turn off the pump during operation, potentially damaging it through frequent on-off cycling. To mitigate this attack, in Figure 15, we introduce a clock, v, which resets to 0 whenever the pump is activated. Once the pump is running, it operates for no fewer than 5 ticks, provided the water level in `T3` is not empty.

5 Results

In order to evaluate the proposed approach, we ran the system shown in Figure 5 under the range of attack scenarios presented in Section 4 both with and without the SFBs. Modifications were made to the *easy-rte* tool to add support for automatic SFB generation in IEC 61499 from VDTA-based policies.

For each test case, the simulation was run for 100,000 steps of execution. We present the same simulation results for *T1 overflow*, *T2 overflow*, and *T3 pump damage* (P_1 through P_3) as shown in Lanotte et al. [17], where the attacks focus on the controller side and are mitigated solely by the output enforcer, along with two additional properties (P_4 and P_5) that show bi-directional and timing behaviours. Finally, we also conducted performance tests to assess and report on the overhead of our proposed approach.

5.1 Uni-directional Enforcement

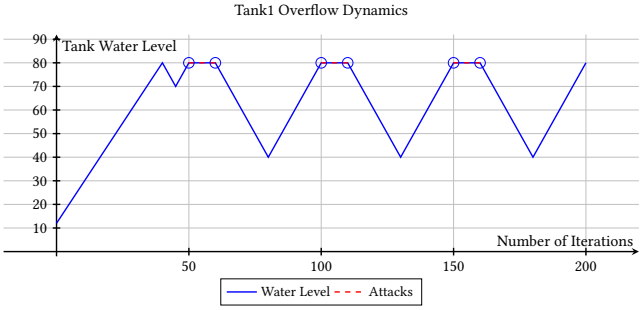

Figure 16: `T1` overflow

For our first attack vector, `T1` overflow, Figure 16 shows the water level in `T1`, measured over the first 200 ticks in combination with the SFB for P_1. Initially, the water level rises gradually as the pumps operate, while the water level decreases when the valve is opened. The highlighted regions in red indicate attacks on the controller, where the attacker attempts to activate the pumps despite the tank being full. Table 2 shows this in more detail, where the pump control signal $pump_1$ has been erroneously set to 1 due to an attack. However, the enforcer for P_1 has corrected this output, causing the pump to remain off and the water level to remain constant.

A similar behaviour is seen for the second attack vector, where an attacker attempts to overflow `T2` by opening the incoming valve. Table 3 shows some key datapoints from this simulation, where the request signal (r) has been corrected by the enforcer P_2 to avoid an

Table 2: Signal valuations under T1 overflow (P_1)

Tick	$level_1$	$level'_1$	$pump_1$	$pump'_1$
0	13	13	1	1
...	...	...	...	...
50	80	80	1	0
51	80	80	1	0
...	...	...	...	...
61	76	76	0	0
...	...	...	...	...

overflow event. Subsequently, once the attack ends and the signal is safe, the enforcer takes no protective action.

Table 3: Signal valuations under T2 overflow (P_2)

Tick	$level_2$	$level'_2$	r	r′
0	13	13	0	0
...	...	...	...	...
14	20	20	1	0
15	20	20	1	0
16	20	20	1	0
17	19	19	0	0
...	...	...	...	...

The last of our uni-directional enforcement examples is T3 pump damage. Here, an attacker has tried to turn on $pump_3$ while the water level inside T3 ($level_3$) is empty. However, as shown in Table 4 the enforcer P_3 is able to correct this and avoid pump damage.

Table 4: Signal valuations under T3 pump damage (P_3)

Tick	$level_3$	$level'_3$	$pump_3$	$pump'_3$
0	0	0	1	0
1	0	0	1	0
...	...	...	...	...
10	0	0	0	0
11	2	2	0	0
...	...	...	...	...
30	0	0	1	0
31	0	0	1	0
...	...	...	...	...

5.2 T2 Combination Attack

The next example requires bi-directional enforcement as there are simultaneous attacks on the plant, by modifying the T2 level sensor, and the controller, by modifying the request signal. Table 5 shows the system behaviour for the first 10 ticks during this attack. Here, we observe unsafe signals on both the input and output sides, sometimes even occurring within the same iteration, indicating the attacker's attempt to compromise the system through both vectors. However, the input enforcer is correcting the level sensor signal whenever an invalid input is received, while the output enforcer is correcting the request signal. Thus, despite the CP-attack targeting both the plant and controller simultaneously, the SFB effectively mitigates these attacks and prevents potential system failure.

Table 5: Signal valuations under T2 combination attack (P_4)

Tick	$level_2$	$level'_2$	r	r′
0	13	13	1	1
1	14	14	1	1
2	−81	20	1	0
3	372	20	0	0
4	−33	20	0	0
5	12	12	0	0
...	...	...	...	...

5.3 T3 pump burnout

Finally, we look at the case when an attacker attempts to quickly turn $pump_3$ on and off. Table 6 shows a trace under these conditions with P_5 using a timer v to detect this behaviour. The SFB is able to prevent rapid switching of the pump by forcing the signal to remain high at tick 7. Additionally, P_5 also contains logic to prevent running the pump when the tank is empty, as shown at tick 10.

Table 6: Signal valuations under T3 pump burnout (P_5)

Tick	$level_3$	$pump_3$	$pump'_3$	v
0	10	0	0	0
...	...	...	...	...
6	8	1	1	1
7	6	0	1	2
8	4	0	1	3
9	2	0	1	4
10	0	0	0	0
...	...	...	...	...

5.4 Performance Evaluation

To assess the impact of enforcement on execution time, we conducted performance evaluations using an Intel(R) Core(TM) i7-7700HQ CPU at 3.8 GHz with 16 GB of RAM running a 64-bit Windows operating system. Each test case was executed for 100,000 time steps and repeated for 10 independent trials calculating the mean execution time to obtain a reliable performance estimate.

Table 7 presents the results for each of our test cases, including a base case with no enforcers ("None"). In this table, we record the Average Execution Time (AET) over all trials. In scenarios where attacks occur on one side (either the controller or the plant side), the overhead ranges from 0.27 % to 0.45 %. In cases that require Bi-RE, the overhead of our SFB approach increases to 1.37 %.

5.5 Discussion

These results demonstrate that our approach facilitates the application of RE techniques to existing industrial automation designs

Table 7: Performance Analysis Results

Test Case	AET (ms)	Overhead
None	4689.7	—
T1 Overflow	4710.8	0.45 %
T2 Overflow	4707.1	0.37 %
T3 Pump Damage	4703.3	0.29 %
T2 Combination Attack	4753.5	1.37 %
T3 Pumping Burnout	4702.6	0.27 %

without significantly compromising the original system performance. Interestingly, attacks which involve T1, whether on the input or output side, seem to exhibit larger execution overhead compared to other tanks. This can be attributed to the increased complexity of PLC1 which contains two input and three output signals. Additionally, the overhead for the combination attack on T2 (1.37 %) is higher than other scenarios. This illustrates the state-space explosion problem, where increased complexity does not scale in a linear fashion. However, even for this most complex example we consider it a minimal performance loss. A key limitation of this approach lies in its inability to enforce multiple policies within a single SFB. Each SFB is restricted to enforce one policy only. Consequently, future work will need to address mechanisms for merging or composing multiple policies within this framework.

6 Related Work

A large amount of research into attack vectors for CP-attacks continues to be investigated, including Denial of Service (DoS) attacks [6, 38], false data injection attacks [21, 26], deception attacks [19], integrity attacks [22, 23], replay attacks [39], zero-dynamics attacks [33], and ε stealthy attacks [2, 15]. While these have been extensively analysed from a control-theoretic standpoint there remains a pressing need for effective security approaches to prevent and mitigate these kinds of attacks, especially using scalable methods, at run-time.

In the industrial sector, Runtime Verification (RV) is commonly employed to detect malicious activities. The Argus defence framework [1] has been proposed to address complex multicomponent deception attacks in industrial settings. Schellenberger et al. [29] explore the creation of a secure channel between the plant and the controller to mitigate communication-based attacks. Similar investigations have been reported in [20] and [9]. Eslami et al. [8] investigate time-varying dynamics, introducing external states with dynamics unknown to the attacker. Taheri et al. [32] have designed a combination of plant-side filters and observers, along with control-side filters, which attempt to mitigate attack vectors even if the attacker fully knows all system parameters. Jhunjhunwala et al. [13] introduced a new RV method that supervises adapter connections in IEC 61499 applications, using state machines to define and monitor specific properties which trigger an event when breached. However, RV only *monitors* system behaviour without actively enforcing corrections. As a result violations may still occur, potentially causing system damage or failure.

To address this, attempts have been made in recent years to support the RE approach, a paradigm which aims to not only *detect* violations of provided policies but also *correct* them [28]. This technique has been used to guarantee the safety for a range of CPSs such as a cardiac pacemaker [28], the over-current protection circuitry of a power system [27], and for the control of a swam of drones [25]. However, its use in industrial automation is minimal.

Do Tran et al. [7] suggest the incorporation of *contract monitors* into IEC 61499 applications which can uphold certain predetermined properties during the application's runtime. However, it is important to note that this method primarily ensures the accuracy of individual components rather than the entire application.

Lanotte et al. [17] introduced the case study of a water treatment system in an industrial setting, aligning with the IEC 61131 standard. Secure proxies were used to restrict the outputs of the controller to ensure safety, similar to RE, using the formal language Timed Calculus of Monitored Controllers (TCMC) which extends Timed Process Language (TPL) [10]. However, the proxy design is only unidirectional and focuses solely on the controller, which may lead to potential attacks on the plant side being neglected. This concern is amplified in industrial environments where control architectures typically operate at fixed rates [30], as these settings are more prone to risks from threats not monitored on the plant side.

In summary, despite extensive research into the safety of CPSs and industrial systems, existing approaches exhibit notable limitations. RV techniques are designed to monitor against predefined safety properties, but do nothing to prevent violations. RE provides an avenue to address this, however recent research on this in the industrial domain largely adopts a unidirectional perspective. Furthermore, most methods focus on ensuring the correctness of individual components, lacking scalability for IEC 61499 environments. As such, ensuring the correctness and safety of systems designed using the IEC 61499 standard, which has become a cornerstone of Industry 4.0, remains a significant concern. Integrating Bi-RE into the IEC 61499 framework as a general solution is crucial.

7 Conclusion

Industry 4.0 continues to evolve and CPSs are increasingly becoming more integrated. The IEC 61499 standard has evolved as the leading language for such applications due to its ability to capture complex and distributed systems using familiar languages, however until now it lacks any ability to provide formal security assurances. The approach proposed in this paper, SFBs, integrates existing Bi-RE techniques into the IEC 61499 standard natively, allowing for formal VDTA-based properties to be guaranteed across an industrial application.

To illustrate the effectiveness of this approach, we implemented SFBs in the context of a water treatment system with the aim of preventing CP-attacks that could occur. Our experimental results show this approach in action across a range of attacks that target both the plant and controller signals. Additionally, performance analysis shows that the overhead associated with SFBs is minimal in the context of the overall system, indicating its suitability for widespread adoption in embedded CPS applications where performance is often constrained.

Future research include advancing the compositionality of SFBs to enhance scalability on large-scale problems, as well as interoperability with the IEC 61499 framework to better leverage its features and integrate with existing IEC 61499 tooling.

References

[1] Sridhar Adepu, Siddhant Shrivastava, and Aditya Mathur. 2016. Argus: An orthogonal defense framework to protect public infrastructure against cyber-physical attacks. *IEEE Internet Computing* 20, 5 (2016), 38–45.

[2] Cheng-Zong Bai, Fabio Pasqualetti, and Vijay Gupta. 2017. Data-injection attacks in stochastic control systems: Detectability and performance tradeoffs. *Automatica* 82 (2017), 251–260.

[3] Jan Olaf Blech, Per Lindgren, David Pereira, Valeriy Vyatkin, and Alois Zoitl. 2016. A comparison of formal verification approaches for IEC 61499. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 1–4.

[4] Roderick Bloem, Bettina Könighofer, Robert Könighofer, and Chao Wang. 2015. Shield synthesis: Runtime enforcement for reactive systems. In *International conference on tools and algorithms for the construction and analysis of systems*. Springer, 533–548.

[5] John Conway. 2020. IEC 61499: The Industrial Automation Standard for Portability that Unleashes Industry 4.0.

[6] Kemi Ding, Yuzhe Li, Daniel E Quevedo, Subhrakanti Dey, and Ling Shi. 2017. A multi-channel transmission schedule for remote state estimation under DoS attacks. *Automatica* 78 (2017), 194–201.

[7] Duc Do Tran, Jörg Walter, Kim Grüttner, and Frank Oppenheimer. 2020. Towards time-sensitive behavioral contract monitors for iec 61499 function blocks. In *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, Vol. 1. IEEE, 27–34.

[8] Ali Eslami and Khashayar Khorasani. 2023. Detection of Event-Based Covert Attacks in Cyber-Physical Systems. In *2023 9th International Conference on Control, Decision and Information Technologies (CoDIT)*. IEEE, 920–925.

[9] Paul Griffioen, Sean Weerakkody, and Bruno Sinopoli. 2019. An optimal design of a moving target defense for attack detection in control systems. In *2019 American Control Conference (ACC)*. IEEE, 4527–4534.

[10] Matthew Hennessy and Tim Regan. 1995. A process algebra for timed systems. *Information and computation* 117, 2 (1995), 221–239.

[11] Neil Higgins, Valeriy Vyatkin, Nirmal-Kumar C Nair, and Karlheinz Schwarz. 2010. Distributed power system automation with IEC 61850, IEC 61499, and intelligent control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 41, 1 (2010), 81–92.

[12] IEC Iec. 2003. 61131-3: Programmable controllers–part 3: Programming languages. *International Standard, Second Edition, International Electrotechnical Commission, Geneva* 1 (2003), 2003.

[13] Pranay Jhunjhunwala, Jan Olaf Blech, Alois Zoitl, Udayanto Dwi Atmojo, and Valeriy Vyatkin. 2021. A design pattern for monitoring adapter connections in IEC 61499. In *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*, Vol. 1. IEEE, 967–972.

[14] Tomas Kulik, Brijesh Dongol, Peter Gorm Larsen, Hugo Daniel Macedo, Steve Schneider, Peter WV Tran-Jørgensen, and James Woodcock. 2022. A survey of practical formal methods for security. *Formal aspects of computing* 34, 1 (2022), 1–39.

[15] Enoch Kung, Subhrakanti Dey, and Ling Shi. 2016. The Performance and Limitations of ε-Stealthy Attacks on Higher Order Systems. *IEEE Trans. Automat. Control* 62, 2 (2016), 941–947.

[16] Ralph Langner. 2013. To kill a centrifuge: A technical analysis of what stuxnet's creators tried to achieve. *The Langner Group* 37 (2013).

[17] Ruggero Lanotte, Massimo Merro, and Andrei Munteanu. 2022. Industrial control systems security via runtime enforcement. *ACM Transactions on Privacy and Security* 26, 1 (2022), 1–41.

[18] RM Lee, MJ Assante, and T Conway. 2014. German steel mill cyber attack. Industrial Control Systems.

[19] Yuzhe Li, Ling Shi, and Tongwen Chen. 2017. Detection against linear deception attacks on multi-sensor remote state estimation. *IEEE Transactions on Control of Network Systems* 5, 3 (2017), 846–856.

[20] Hao Liu, Shaodong Wang, and Yuzhe Li. 2022. Event-triggered control and proactive defense for cyber–physical systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 52, 10 (2022), 6305–6313.

[21] Yilin Mo and Bruno Sinopoli. 2010. False data injection attacks in control systems. In *Preprints of the 1st workshop on Secure Control Systems*, Vol. 1.

[22] Yilin Mo and Bruno Sinopoli. 2014. Secure estimation in the presence of integrity attacks. *IEEE Trans. Automat. Control* 60, 4 (2014), 1145–1151.

[23] Yilin Mo and Bruno Sinopoli. 2015. On the performance degradation of cyber-physical systems under stealthy integrity attacks. *IEEE Trans. Automat. Control* 61, 9 (2015), 2618–2624.

[24] Patrick O'Connor. 2022. Turning Up the Heat: 2022's Biggest Hacks. *ITNOW* 64, 4 (2022).

[25] Abhinandan Panda, Alex Baird, Srinivas Pinisetty, and Partha Roop. 2023. Incremental security enforcement for cyber-physical systems. *IEEE Access* 11 (2023), 18475–18498.

[26] Zhong-Hua Pang, Guo-Ping Liu, Donghua Zhou, Fangyuan Hou, and Dehui Sun. 2016. Two-channel false data injection attacks against output tracking control of networked systems. *IEEE Transactions on Industrial Electronics* 63, 5 (2016), 3242–3251.

[27] Hammond Pearce, Srinivas Pinisetty, Partha S Roop, Matthew MY Kuo, and Abhisek Ukil. 2019. Smart I/O modules for mitigating cyber-physical attacks on industrial control systems. *IEEE Transactions on Industrial Informatics* 16, 7 (2019), 4659–4669.

[28] Srinivas Pinisetty, Partha S Roop, Steven Smyth, Nathan Allen, Stavros Tripakis, and Reinhard Von Hanxleden. 2017. Runtime enforcement of cyber-physical systems. *ACM Transactions on Embedded Computing Systems (TECS)* 16, 5s (2017), 1–25.

[29] Christian Schellenberger and Ping Zhang. 2017. Detection of covert attacks on cyber-physical systems by extending the system dynamics with an auxiliary system. In *2017 IEEE 56th annual conference on decision and control (CDC)*. IEEE, 1374–1379.

[30] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. 2009. Mobile robots. *Robotics: Modelling, Planning and Control* (2009), 469–521.

[31] Roopak Sinha, Partha S Roop, Gareth Shaw, Zoran Salcic, and Matthew MY Kuo. 2015. Hierarchical and concurrent ECCs for IEC 61499 function blocks. *IEEE Transactions on Industrial Informatics* 12, 1 (2015), 59–68.

[32] Mahdi Taheri, Khashayar Khorasani, Iman Shames, and Nader Meskin. 2023. Cyberattack and Machine-Induced Fault Detection and Isolation Methodologies for Cyber-Physical Systems. *IEEE Transactions on Control Systems Technology* (2023).

[33] André Teixeira, Iman Shames, Henrik Sandberg, and Karl Henrik Johansson. 2015. A secure control framework for resource-limited adversaries. *Automatica* 51 (2015), 135–148.

[34] Valeriy Vyatkin. 2009. The IEC 61499 standard and its semantics. *IEEE Industrial Electronics Magazine* 3, 4 (2009), 40–48.

[35] Valeriy Vyatkin, Zoran Salcic, Partha S Roop, and John Fitzgerald. 2007. Now that's smart! *IEEE Industrial Electronics Magazine* 1, 4 (2007), 17–29.

[36] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. 2009. Formal methods: Practice and experience. *ACM computing surveys (CSUR)* 41, 4 (2009), 1–36.

[37] Li Hsien Yoong, Partha S Roop, Valeriy Vyatkin, and Zoran Salcic. 2009. A synchronous approach for IEC 61499 function block implementation. *IEEE Trans. Comput.* 58, 12 (2009), 1599–1614.

[38] Heng Zhang, Peng Cheng, Ling Shi, and Jiming Chen. 2015. Optimal DoS attack scheduling in wireless networked control system. *IEEE Transactions on Control Systems Technology* 24, 3 (2015), 843–852.

[39] Minghui Zhu and Sonia Martinez. 2013. On the performance analysis of resilient networked control systems under replay attacks. *IEEE Trans. Automat. Control* 59, 3 (2013), 804–808.

A Enforcer Examples

```
// Map input variables to the internal variables
v = v_in;
A = A_in;
State = State_in;

if (A) {
    // Violation
    A = 0;
}

if (!A) {
    // Do nothing
}

// Map internal variables to output variables
v_out = v;
A_out = A;
State_out = State;
```

Listing 1: Input Enforcer example `EnforceS1`

```
// Map input variables to the internal variables
v = v_in;
State = State_in;
A = A_in;
B = B_in;
v++;

if (!A and !B and v < 5) {
    // Do nothing
}

if (!A and B) {
    // Go to S0
    State = "S0";
}

if ((A and B) or (A and !B)) {
    A = 0;
    if (!A and !B and v < 5) {
        // Do nothing
    }
    if (!A and B) {
        // Go to S0
        State = "S0";
    }
}

if (v >= 5) {
    // Violation
    B = 1;
    // Go to S0
    State = "S0";
}

// Map internal variables to output variables
A_out = A;
B_out = B;
v_out = v;
State_out = State;
```

Listing 2: Output Enforcer example `EnforceS1`