Resource Aware Synthesis of Automotive Security Primitives

Soumyajit Dey, Ipsita Koley, and Sunandan Adhikary

1 Introduction

With the evolution of transportation systems, modern-day vehicles are no more mere mechanical systems. Contemporary automotive architectures are designed as a collection of cyber-physical control loops with an aim to provide energy-efficient performance, safety, comfort, and connected mobility features. These software-governed automotive controllers supervise a plethora of functionalities like engine control, power management, regenerative braking, lane-keeping, comfort features, etc. Examples from domains like safety would be features like Vehicle Stability Control (VSC), Anti-lock Braking System (ABS), Roll Stability Control (RSC), etc. Convenience features like Adaptive Cruise Control (ACC) are also ubiquitous in most vehicles nowadays.

Features are implemented in the form of control programs mapped to Electronic Control Units (ECUs) as real-time tasks. *Sensing* tasks process sensor measurements, *communication tasks* interface with communication hardware and send such measurements over the communication channels and *control* tasks compute the desired control input for respective actuators. An ECU may host multiple control tasks. Moreover, some control functionalities may require tasks spanning over multiple ECUs to attain some global control objective. For example, on identifying a life-threatening situation, the Central Locking System (CLS) that controls the power door locking mechanism works alongside the crash detection system to ensure occupants' safety.

With time, the number of ECUs in modern vehicles has been increasing with the quest for more features that make transportation safer and more convenient.

S. Dey (✉) · I. Koley · S. Adhikary
Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Kharagpur, West Bengal, India
e-mail: soumya@cse.iitkgp.ac.in; ipsitakoley@iitkgp.ac.in; mesunandan@kgpian.iitkgp.ac.in

V. K. Kukkala, S. Pasricha (eds.), *Machine Learning and Optimization Techniques for Automotive Cyber-Physical Systems*, https://doi.org/10.1007/978-3-031-28016-0_6

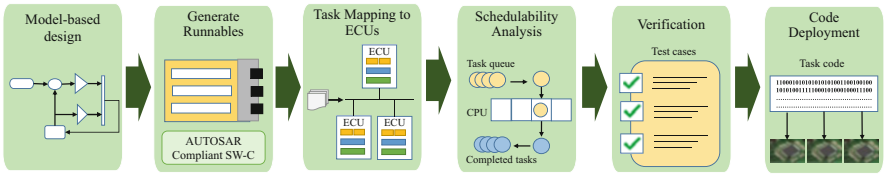

Fig. 1 Automotive software development flow

For example, BMW 7-series models have as many as 150 ECUs. Generalized bus-oriented architectures have come into the picture to realize the real-time collaboration of such a large number of ECUs. The electrical/electronic (E/E) [1] architecture of modern vehicles is divided into different functional domains (power-train, body control, infotainment, etc). Based on the bit rate, fault-tolerance, and soft/hard real-time requirements of each domain, the intra-vehicular network [2] of a contemporary car comprises network elements with multiple lightweight protocols making it heterogeneous in nature. Example protocols include Controller Area Network (CAN) [3], FlexRay [4], Local Interconnect Network (LIN) [5], Media oriented systems transport (MOST) [6]. The E/E architecture of modern vehicles ensures the inter-operability of such heterogeneous network protocols.

The current design flow of automotive architectures (Fig. 1) in the industry is compartmentalized into model-based design, development, and standardized implementation on target platforms, each followed by testing and verification against certain specifications. The implementations of control software in ECUs are generalized by certain standard guidelines set by AUTOmotive Open System ARchitecture also known as AUTOSAR [7], which is a predominant entity built as a worldwide development partnership among automotive industries. Such standards encourage the model-based development of modularized and reusable control functionalities for automotive subsystems. This is followed by an AUTOSAR compliant conversion of these model-based designs of controllers into runnable programs. Then the control programs are implemented on the ECUs and invoked from the system level as control tasks based on the task allocations in ECUs.

Today, the design specification and implementation of automotive controllers are mostly carried out using Synchronous Reactive (SR) models such as those modeled in the Simulink and Stateflow tools [8]. TIER 1 suppliers organize the control functionalities as a hierarchy of subsystems and define them as a network of blocks in Simulink. Code implementation of each subsystem is generated as a set of functions. These sets of runnables are then standardized with AUTOSAR-guided specifications as Autosar Software Components (SWC) [8, 9]. The AUTOSAR model specifies the data, execution, and call dependencies for all the functions. All the SWCs from the TIER 1 suppliers are collected and connected to a system-level model by the OEMs and Carmakers. Using AUTOSAR tools, they map the runnables or the functions into tasks. Schedulability analysis is performed using platform-specific utilities (eg. symtavision for Infineon ECUs). Accordingly, tasks are allocated to the processors. Formal tests are conducted on the initial designs using some verification tool, like

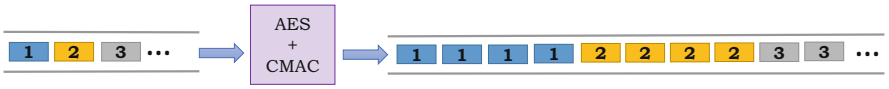

Fig. 2 Effect of CMAC/AES on CAN Traffic (numerals denote message id-s)

Simulink Test. Simulink Test facilitates test case and test suite definitions along with automation of test harness generation. Following the simulation tests, the C-code of the model is generated using Embedded Coder or Simulink Coder and implemented on the processor.

Security requirements have been an afterthought in the automotive software development flow discussed above. AUTOSAR mandates cryptographic mechanisms, like MAC (Message Authentication Code) to authenticate communications through the intra-vehicular network [10]. However, such cryptographic methods incur computation and communication overload. For example, securing an 8 byte CAN message using AES encryption and SHA-2 MAC algorithm will generate 6 CAN frames for a single CAN frame [11]. To deal with such bus load issues, AUTOSAR has suggested using truncated MAC [10]. The problem with MAC is that it can detect an external cyber attack, but fails to detect insider attacks and denial-of-service attacks like bus-off [12]. The embedded platforms where the automotive controllers are implemented are mostly of low computation power. On the other hand, cryptographic security algorithms, like MAC, incur significant computation and communication load. For example, on a 96 MHz ARM Cortex-M3-based Electronic Control Unit (ECU), some of the well-known control law computations take approximately 5 μs while a 128-bit MAC computation for a single message takes 100 μs [13]. On the other hand, if CMAC hash and AES-128 encryption algorithm are used to secure CAN frames, each CAN frame will be replaced by 4 CAN frames (Fig. 2) [11]. Imagine the load on CAN traffic if every CAN packet is secured this way. As most automotive CPSs are safety-critical with hard real-time deadlines, it naturally raises the question of how practical it is to implement cryptographic security algorithms on such embedded platforms. As an alternate solution, a number of researchers have proposed to use control-theoretic light-weight attack detectors[14] in place of periodic cryptographic security checks. These detectors are designed by exploiting the control theoretic properties of automotive CPSs [14, 15].

This naturally leads us to the problem of evolving automotive software design flows, which must consider the smooth integration of lightweight security primitives along with software controllers while maintaining verifiability, schedulability, and other platform constraints. The above also brings up the question of how such detection systems can be of practical use and in what way such existing approaches may be improved, more specifically in the automotive context. In this regard, we now discuss the major contributions to this chapter.

1. In control-theoretic light-weight attack detectors, the residue i.e., the difference between the actual sensor measurements and the estimated sensor measurements

is compared with a threshold. An alarm is raised when the residue surpasses the threshold value. The performance of the detector depends on the value of the threshold. If a lower valued threshold is selected, even noise can be considered an attack. This will lead to false alarms. On the other hand, a smartly crafted stealthy attack, like a zero-dynamic attack [15], can easily bypass a higher-valued threshold. So, the question is how to wisely compute the threshold value. In Sect. 3, we discuss some methodologies to synthesize such fixed threshold-based detectors.

2. Next, we consider a more informed attack scenario. An external attacker can snoop into intra-vehicular networks through OBD port and telematic units [16]. Widely used intra-vehicular network protocols like CAN transmits data in broadcast mode. Therefore, an attacker who has access to the intra-vehicular network can analyze transmitted data packets and design optimal attacks. We discuss in Sect. 4, how adaptive and intelligent threshold-based detectors can be designed to thwart such attack attempts.
3. While light-weight detection is an important task in the context of security-aware automotive CPS design, another important feature is what to do when an attack is detected. A number of researchers have proposed robust controller design methods to make the system robust against attacks. In this chapter, we discuss an alternative approach. In Sect. 5, we present how an intermittent MAC along with additional control logic can diminish the effect of the attack on the system.
4. Finally, in Sect. 6, we have presented how to realize some of these security-aware automotive CPS design methods in a Hardware-in-Loop (HIL) experimental setup.

2 Background and Related Work

2.1 System Model for Secure CPS

Similar to other model-based CPSs, the automotive software design life-cycle also conceptualizes modular subsystems for certain desired operating regions. For efficient and real-time control computation, a nonlinear plant $\dot{x}(t) = f(x)$ is usually linearized around such an operating point in the form of a linear time-invariant (LTI) system expressed as follows.

$$\dot{x}(t) = \Phi x(t) + \Gamma u(t) + w(t),\ y(t) = Cx(t) + v(t) \tag{1}$$

Here $x(t) \in \mathbb{R}^n$ is system state, $u(t)$ is output of the controller, $y(t) \in \mathbb{R}^m$ is system output under the influence of physical process noise $w(t) \in \mathbb{R}^n \sim \mathcal{N}(0, \Sigma_w)$ and measurement noise $v(t) \in \mathbb{R}^m \sim \mathcal{N}(0, \Sigma_v)$ at time t (w and v are independent Gaussian random variables with Σ_w and Σ_v as variance parameters). Also, Φ,Γ, and C are transition matrices, derived from the physical plant equations. Φ is known

as the state transition matrix, Γ is known as the input-to-state transition matrix and C is the output matrix. The output of the plant is sensed and used for control input generation.

Depending on the plant-state characteristics and the available sampling periods in the ECU, the plant outputs are sampled periodically. For this, the control program uses the discretized versions of the above state-space equations, i.e.

$$x[k+1] = Ax[k] + Bu[k] + w[k],\ y[k+1] = Cx[k+1] + v[k] \tag{2}$$

Above equations express the k-th sampling iteration of the discretized system i.e.,$t \in [hk, h(k+1)]$, where h is the chosen sampling interval. Therefore, the new transition matrices become $A = e^{\Phi h},\ B = \int_{hk}^{h(k+1)} e^{\Gamma s}\Gamma ds$ [17]. At the controller side, this sensed plant output $y[k+1]$ is received once every sampling period. The controller needs to estimate the actual plant states using this output in order to calculate a suitable control input u to control the plant dynamics. To estimate the plant states from the sensed outputs, typically an *observer* is used.

$$\begin{aligned}\hat{x}[k+1] &= A\hat{x}[k] + Bu[k] + Lr[k],\ r[k] = y[k] - C\hat{x}[k],\ u[k+1] \\ &= -K\hat{x}[k+1]\end{aligned} \tag{3}$$

As shown in Eq. 3 the estimated state at $(k+1)$-th iteration is denoted using $\hat{x}[k+1] \in \mathbb{R}^n$ and it is derived using a similar state-space equation like Eq. 2 along with a suitable correction $Lr[k]$ in order to track the actual state. The quantity $r[k] = y[k] - C\hat{x}[k]$ is known as system residue and it signifies the error between estimated and actual outputs. The observer gain L is designed in such a way that minimizes the residue [17]. The feedback control input $u[k+1]$ at the $(k+1)$-th sampling iteration is calculated based on the current estimated state $\hat{x}[k+1]$. We consider K as a pre-calculated optimal control gain. The control input thus calculated is then used to actuate the plant and stabilize it around the target operating point. As an example, consider Fig. 3 which demonstrates such closed-loop interaction between plant(s) and controller(s). We represent as a high-level view of a system under control where different subsystems with corresponding dynamics are modeled as plants (denoted as P_i for the i-th plant) and for each of them, suitable measurements are obtained using a set of sensors (denoted as S_i for the i-th plant). For each P_i, we have a corresponding control (denoted as c_i) and estimation task (denoted as e_i) implemented in the ECU. The communicated data and control outputs are suffixed with the plant names and tasks are suffixed with the plant indices for better understanding.

Like standard information processing systems, there are three fundamental security properties of any computer-controlled system and the information it deals with, i.e., *confidentiality, integrity and availability* (CIA). Now, there are several kinds of Man-in-the-Middle attacks that can observe and then utilize some system-specific knowledge to corrupt the communicated data to hamper its *integrity*. Insider attacks of this kind do not target *confidentiality* and can not be stopped by the

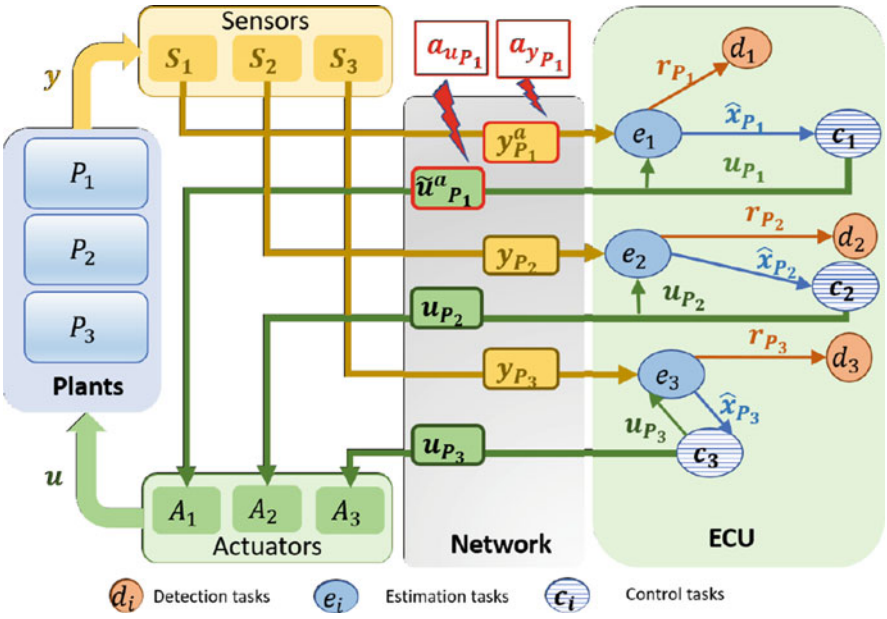

Fig. 3 Component-level overview of secure CPS under false data injection (FDI)

encryption policies mandated as per the AUTOSAR standards (truncated CMAC with 128 bit AES). Such attacks can be generalized as *False Data Injection Attacks* which corrupt the actual sensor and actuator data with a certain amount of false data. Our discussion in this chapter will primarily focus on the effects and countermeasures of False Data Injection (FDI)-type Man-in-the-middle attacks on control loops. In Fig. 3, such attacks on measurement and actuation are denoted by the variables $a_{y_{P_i}}$ and $a_{u_{P_i}}$ for a plant P_i. We assume that the ECU also runs a detection task d_i for every i-th loop in the system. We start with an example of simple threshold-based detection. Threshold-based detection tasks (Fig. 3) are designed to monitor the transmitted sensor data and flag an attack or anomaly in system output whenever the residue (or some derived statistics from it) surpasses some pre-fixed constant detector threshold Th i.e.,

$$\|r[k]\|_p > Th, \ (\|r\|_p = (\sum \|r\|^p)^{1/p}) \tag{4}$$

where p is the chosen norm. A suitable threshold value on the residue statistics can be chosen to constrain the estimation error. For a control loop, the state progression under FDI attacks can be expressed as follows.

$$x^a[k+1] = Ax^a[k] + B\tilde{u}^a[k] + w[k], \tag{5}$$

$$y^a[k+1] = Cx^a[k+1] + v[k] + a_y[k], \tag{6}$$

$$\hat{x}^a[k+1] = A\hat{x}^a[k] + Bu^a[k] + Lr^a[k],\ r^a[k] = y^a[k] - C\hat{x}^a[k] \tag{7}$$

$$u^a[k+1] = -K\hat{x}^a[k+1],\ \tilde{u}^a[k+1] = -K\hat{x}^a[k+1] + a_u[k] \tag{8}$$

Here $x^a[k]$, $\tilde{u}^a[k]$, $y^a[k]$ and $\hat{x}^a[k]$ are the attacked variants of the state, control input, output and estimated state vectors (from Eqs. 2 and 3) respectively at k-th iteration. By *attacked*, we mean to say under the influence of additive false data $a_u[k]$ injected on actuation, and $a_y[k]$ injected on sensor data at k-th sampling iteration. Also, $r^a[k]$ denotes the system residue under attack scenario (from Eq. 3) at k-th sampling instance. Note the difference between $\tilde{u}^a[k]$ and $u^a[k]$. The first one is the control input at k-th iteration under the influence of an additive FDI attack on actuation ($a_u[k]$) and the second one is not affected by the actuation attack but is calculated using the estimated states derived from the FDI affected ($a_y[k]$) sensor readings y^a. The estimated states are calculated in the controller side itself (i.e., not transmitted via the network under attack or not actuated via the actuator under attack). Therefore, unlike the actual plant state calculation, (where $\tilde{u}^a[k]$ is used to calculate $x^a[k+1]$), $u^a[k]$ is used for the calculation of estimated state $\hat{x}^a[k+1]$. In Fig. 3, we demonstrate such a data falsification attack on an automotive communication network. The attack vector at k-th sampling iteration is symbolically represented as $\mathcal{A}[k]^T = [a_u[k]^T, a_y[k]^T]^T$. If the attacker continues the false data injection for l sampling iterations, then the l length attack vector is expressed as follows.

$$\mathcal{A}_l = [\mathcal{A}[1] \cdots \mathcal{A}[l]] = \begin{bmatrix} a_u[1] \cdots a_u[l] \\ a_y[1] \cdots a_y[l] \end{bmatrix}$$

Since automotive control loops are highly *safety-critical*, an intelligent attacker can design the FDIs while utilizing system model knowledge with an aim to make the system states *unsafe*. To achieve this, the attacker has to compromise the sensors/actuators or the intra-vehicular communication networks (e.g. the CAN bus). In this process, the attacker might get detected as the residue-based detection tasks are always running in the ECU looking for anomalies where the residue-statistic changes undesirably or beyond a certain threshold. Therefore, the attacker also needs to design the false data in a way such that it can maintain its *stealth* while making the system eventually unsafe [15]. Considering n as the dimension of the system and $X_S \subset \mathbb{R}^n$ being the safe region of system states, the following is the criteria that an N-length *stealthy and successful FDI* attack vector $\mathcal{A}_N$ has to satisfy.

$$\|r^a[k]\|_p \leq Th\ \forall k \in [1, N] \textbf{ and } x^a[N+1] \notin X_S \tag{9}$$

Figure 4 demonstrates such a successful attack injected into the sensor and actuation data of an Automatic Cruise Control (ACC) system. The states of the system are deviation (D) from the reference trajectory and the velocity (V) of the vehicle. The velocity (V) is considered as system output and is controlled using acceleration

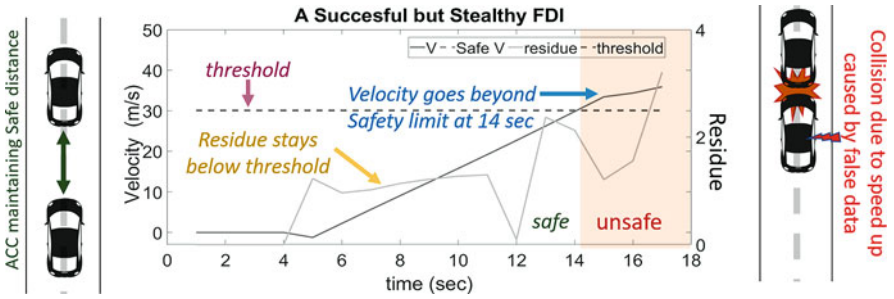

Fig. 4 A successful yet stealthy FDI

control input. The safety boundary for V is considered to be 30 m/s to maintain a safe distance from the preceding vehicle. Following are the states and transition matrices of the ACC plant sampled at every $T_s = 0.1$ s, $A = \begin{bmatrix} 1 & 0.1 \\ 0 & 1 \end{bmatrix}$, $B = \begin{bmatrix} 0.005 \\ 0.1 \end{bmatrix}$, $C = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $D = [0]$, $x^a = \hat{x}^a = \begin{bmatrix} D \\ V \end{bmatrix}$. The controller and observer gains used to achieve the closed-loop functionalities are respectively $K = [0.9171\ 1.6356]$ and $L = [0.8327\ 2.5029]^T$. The system is equipped with a residue-based detector with $threshold = 2.5$. The stealthy attacker can inject falsified velocity and acceleration data communicated between the plant and controller. Figure 4 is a plot of velocity and residue under a 13 length false data injection attack vector launched on this Automatic Cruise Control system (ACC). We can see that the false data is successfully pushing the velocity of the follower vehicle beyond the safety limit, i.e., 30 m/s at 14 sec time instant but the residue still remains below $threshold$. Hence, this is a successful 13-length attack vector as per Eq. 9 as it successfully makes the system unsafe before the alarm is raised by the detector. In this chapter, we consider this generalised set of false data injection type attack, and discuss a security-aware design that can protect the system against them.

2.2 Automotive Software Tools and Standards

Automotive controllers are developed in a modularized fashion with different initial models representing different subsystems aimed to handle specific control operations. The high-level modeling is typically done using a formalism that supports hybrid specification of continuous dynamics and discrete switching logic together. In this development process, Model-Based Design (MBD) methodology is used in early stages by the TIER-I suppliers. MBD tools enable design, testing and verification to be performed in a single design platform. Stateflow/Simulink is widely used by control system designers for this purpose. Controller specifications are defined as networks of Simulink components or state-flow models that are

developed and validated separately as part of a hierarchical system model [18]. Along with simulation-based testing in the system state-space, the design verifiers associated with these platforms verify the high-level design as a hybrid automaton using formal engines. This enables an integrated verification and correction of the developed control system model in the development stage itself. The code generation for the target platform is another feature integrated into these model-based design tools. The Simulink Coder is one such popularly used tool that modularizes the functionalities of each subsystem and provides the binaries for integration to a system-level model. An AUTOSAR compliant conversion of these model-based designs of controllers into runnable programs is done thereafter. These modular tasks are then cluster-wise mapped to the ECUs so that they can be invoked from the system level as control tasks based on the task allocations following the AUTOSAR standards [8, 9]. Thereafter, a thorough schedulability analysis of the collection of tasks in a given ECU core is done in order to validate this. Given the safety-critical and real-time nature of tasks, such automotive performance analysis tools need to be correct with high confidence (even if conservative). Tools like Symtavision SymTA suite (for Infineon ECUs), Inchron chronSUITE are popularly used for this purpose. These tools help in calculating the end-to-end response time for a given task mapping and verifying simulated system response given certain safety, performance criteria, and resource budget. Standard protocols like CAN, Flexray, etc., are also supported in order to analyze the communication busloads and optimize them. After rounds of tests and required design updates, the code for the final design is generated for the target ECUs. After analysing and verifying the generated code using integrated code verifiers (e.g., Polyspace [19]), the binaries are implemented in the ECU following the mapping strategy. Modern automotive ECUs follow a layered software architecture with the AUTOSAR runtime environment (RTE) interfacing with the AUTOSAR software components (SWCs). A service layer follows this application layer that interacts with ECU and Microcontroller abstraction layer (MCAL), which is equipped with complex and low-level device drivers. This facilitates multiple control features to be executed as real-time tasks while sharing the same physical platform.

The crypto stack of AUTOSAR provides an interface for Message Authentication Codes (MACs), Secure Hash Algorithms (SHA), and key-based authentication methods. Crypto service manager (CSM) [20] is the service layer module that interacts with the crypto interface (CryIf) in the ECU abstraction layer and enables communication with the cryptographic software or hardware via the crypto driver module in MCAL. Data packets are transmitted as Protocol Data Units (PDUs) and unpacked into Service Data Units (SDUs) at the receiver's end following the protocol control information (PCI). Standard AUTOSAR guidelines for Secure Onboard communication mandate the use of 128-bit AES with Cipher-based MACs while transmitting PDUs through the communication buses. This prevents unauthorized tampering of data communication but it does not ensure protection against false-data injection type insider attacks. To thwart one such powerful attack i.e., Record and Replay Attack, freshness value (FV) is introduced along with the MACs. But the use of these cryptographic authentications increases the processing and communication

overheads. This is why the AUTOSAR Secure onboard communication (SecOC) directive suggests the use of truncated MACs. This might result in a lower security level, but the use of a 128-bit key size with more than 64-bit MAC is considered to provide significant security against unauthorized intrusions. The security profile for CAN communication suggests the use of 28 most significant bits from MAC (calculated using 128-bit AES with CMAC) and 4 least significant bits from the freshness value.

2.3 Related Studies

There exists a significant amount of work that had shown how an adversary can gain access to the intra-vehicular network physically or remotely [16, 21–24]. Once the access to the intra-vehicular network is gained, any ECU with safety-critical tasks can be compromised and the attack will pose like an *insider attack* effort. Since CAN is a protocol using which most of the safety-critical control messages are broadcasted, it is an ideal attack surface for an FDI attacker. The authors in [12] exploit the in-built error-handling protocol of CAN to send a victim ECU to bus-off mode using a compromised ECU. Authors in [25] take this attack strategy further by extending the bus-off period. They choose an optimal victim message ID, observe when the ECU recovers from bus-off, and re-transmit that ID to target the preceding error transmission frames, thus pushing the ECU back to bus-off. Now that the victim ECU is compromised repeatedly, the attacker can inject fabricated data packets in the CAN bus in the disguise of this victim ECU for a long enough period to make a control loop unsafe. A *denial-of-service* type attack is demonstrated in [21]. Authors show how individual brakes of a real car can be locked and communication with the engine control module, body control module can be disabled by injecting random data packets into the CAN bus. A *false data injection* attack can be inflicted in this way by crafting data packets with false speed information and injecting them into the CAN bus. A *replay attack* methodology is discussed in [26] on the keyless entry system of a vehicle. There are various other automotive attacks in the literature [27, 28]. Such intrusions can cause serious damage to the system but are hard to catch.

To combat such attacks, the integration of cryptographic schemes is proposed by researchers in the automotive domain. The use of Cipher-based Message Authentication Codes (CMAC) based on symmetric key ciphers like AES was chosen as part of Secure Hardware Extension (SHE) for automotives [29]. Since these are computationally simpler than the asymmetric approaches, they are ideal for real-time use with less computational power. But sharing of secret keys among all participating ECUs makes the intra-vehicular network prone to insider attacks. Keys being pre-programmed into the ECUs have been exploited in [21]. To prevent this, the use of Cyclic Redundancy Codes (CRC) along with CMAC suggested in [30] ensures the integrity of intra-vehicular communication.

Control-theoretic monitoring systems are also proposed to deal with power-hungry cryptographic algorithms [13]. These mechanisms offer basic safety checks on a CPS while it operates. There are statistical change detection methods like χ^2-test, Cumulative Sum (CUSUM) [31, 32], that are implemented to detect whether the system output or the system states are anomalous. The residue of the system is monitored for this purpose. If the residue statistic goes beyond a certain pre-calculated threshold, the system is found to be anomalous. Though such lightweight control-theoretic security primitives can limit the attacks, they can also be fooled [14, 15].

Since the standalone use of cryptographic algorithms to secure a CPS is not resource-friendly and control-theoretic anomaly/attack detection units are not sufficient for security either, combining both is usually suggested and is a good choice to build a resource-aware *Intrusion Detection System* (IDS) for CPSs. Authors in [33] proposed such an IDS for securing plant controller communication with reduced resources by sporadically using the cryptographic schemes with attack-resilient control-theoretic detection tasks running in the background. Such intermittent activation of cryptographic schemes is made further resource-aware by utilizing the weakly-hard design constraints of a CPS in [34]. They also explore formal methodologies to ensure that resource awareness would not compromise the safety and security of the CPS.

Another approach is to design the detection task adaptively enough to detect attacks based on the current state of the systems. Authors of [35, 36] have proposed such anomaly detectors that vary their detection thresholds. The work in [35] proposes two greedy algorithms based on formal methods to generate a set of monotonically decreasing thresholds in off-line mode. On the other hand, the authors of [36] formulate an attacker-defender game to solve the adaptive threshold selection problem. In [37], the authors show that the using windowed residue statistic with an optimally chosen threshold, one can have a better idea about the history of the states which can be useful in terms of better attack detection. The work in [38] takes this statistical analysis further toward guided learning of attacked state detection using reinforcement learning (RL) and model knowledge.

In the context of attack mitigation, [39] presented a secure state estimation problem which is further leveraged to compute attack-mitigating robust control inputs using RL. A recent work [40] presented an online attack recovery method by estimating the current system state from the latest trusted data using the checkpoint method from [41] followed by which they synthesized recovery control inputs using a linear program (LP) and formal methods. In [42], trusted hardware components are used as a high-assurance unit to increase the security of the system. As the decider unit, they proposed a side-channel analysis-based intrusion detection system. In the case of connected and automated vehicles, as discussed in [43], on detection of an attack, the system is switched to adaptive cruise control from cooperative adaptive cruise control.

There are several works that address the overall security-aware co-design perspective for automotives. Authors in [44] propose a Lightweight Authentication for Secure Automotive Networks (LASAN), which suggests optimization of the

cryptographic protocols with asymmetric key encryption based on available power, compute, and communication resources. The work in [45] suggests cross-layer co-design of security framework, keeping the performance in mind along with a schedulable solution. This design space exploration solution was applied to an automotive case study to achieve a refined co-design.

3 Lightweight Attack Detection

As we discuss the security-aware design of automotive CPSs, we must keep in mind that the design should ensure the real-time requirements of the safety-critical systems. Hundreds of ECUs collaboratively work together to attain global objectives. Additional communication load due to security primitives must not hamper the real-time aspects of automotive networks. Being light-weight and handling real-time communication has been the primary motivation for designing intra-vehicular network protocols like CAN, FlexRay, etc. Most safety-critical CPSs are connected via CAN but CAN does not have any authentication scheme. It does contain a cyclic redundancy check (CRC) field, however, it can be broken via simple reverse engineering [46]. To ensure utmost security, securing every data packet using some cryptographic method seems the most promising strategy. To date, the traditional cryptographic techniques (for example, message authentication code also known as MAC along with some encryption techniques like RSA, AES, etc.) are known to provide the best security against false data injection (FDI) attacks. But, they incur computational and communication overheads which may lead some of the safety-critical tasks to miss their deadlines (refer to Fig. 2 and corresponding discussion in Sect. 1).

An alternative solution that has been widely suggested by a number of researchers in the literature to deal with the above limitations in the context of security-aware automotive CPSs is to use residue-based attack detectors [14, 15, 32]. A residue is computed as the difference between actual and estimated sensor measurements $r^a[k] = y^a[k] - C\hat{x}^a[k]$ (see Eq. 7). As explained in Sect. 2.1 either some norm of the residue or some statistical derivation of the residue [37] is compared with a threshold value Th. The detector's efficiency depends highly on the value of the threshold. The following two measures are used to quantify the detector's performance. The first one is *true positive rate* (TPR) i.e., the probability at which the detector raises an alarm when an FDI attack is taking place. The second one is *false alarm rate* (FAR) i.e., the probability at which the detector raises an alarm when no attack is taking place. An efficient detector will have higher TPR and lower FAR.

Let us consider an example of a zero-dynamic attack demonstrated on a trajectory tracking control (TTC) system (see Fig. 5). The states of TTC are deviation (D) from the reference trajectory and the velocity (V) of the vehicle, $x^a = [D\ V]^T$. The system matrices are $A = \begin{bmatrix} 1 & 0.1 \\ 0 & 1 \end{bmatrix}$, $B = [0.005\ 0.1]^T$. The system is equipped

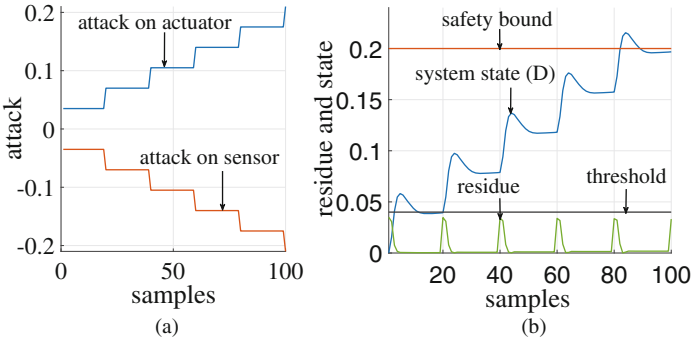

Fig. 5 Zero dynamic attack: (**a**) Attack vectors (**b**) Effect of attack on system and detector

with only distance sensor i.e. $C = [1\ 0]$. Here, the control input is the acceleration of the vehicle. The controller and observer gains of this TTC system are $K = [16.03\ 5.66]$ and $L = [1.87\ 9.65]^T$ respectively. The attacker can modify both distance measurement and acceleration data. A residue-based attack detector is also in place with a constant threshold $Th = 0.4$. The safety limit for state D has been set as 0.2 unit. Consider an FDI attacker crafted stair-case-like attack vector as given in Fig. 5a. Here, by *attack vector* we mean a sequence of false data to be injected to sensor data or actuator signal as mentioned earlier in Sect. 2.1. The intensity of the attack values is constant for a certain number of consecutive samples and then it is increased. While the actuation attacks $a_u[k]$ are positive, attacks on sensor measurements $a_y[k]$ are negative. This is because the attack on the control signal accelerates the changes in plant states and drives the system towards an unsafe region. On the other hand, the attack on sensor measurement hides the reflection of the system's drastic change in the measurements (see Eq. 9). Thus, the detector task that can only see the measurements, not the actual system states, is hoodwinked into thinking that the system is operating as desired. This smartly crafted attack can successfully make the system unsafe as can be seen in Fig. 5b. We are saying the attack is stealthy because following the successful attack criteria mentioned in Eq. 9, the residue remains below the threshold value all the time. It will never trigger an alarm to notify that an attack has taken place. This reduces the detector's TPR. One can reduce the threshold further to improve the detection rate. But, in that case, the detector will consider even small process and/or measurement noises as attacks. This increases FAR and thus reduces the detector's efficacy. Therefore, it is necessary to determine an optimal threshold for which the detector's performance is enhanced. We now discuss 2 state-of-the-art approaches that can be found in the literature for determining the optimal threshold to improve detectability as well as to reduce false alarms.

3.1 Optimal Static Threshold-Based Detector

In [37, 47], the authors considered a *stateful* [32] detection system and provided a theoretical base on how to correlate the characteristics of the detector with system dynamics. A *stateful* detector in the context of residue-based light-weight detection mechanism means the decision of the attack detector does not rely only on the current measurement; rather, a number of past measurements are also considered. Examples of such detectors include CUmulative SUm (CUSUM), windowed χ^2-test statistics-based detectors, etc. Let us summarize the idea by considering a CUSUM detector. Consider we have m sensors to measure that plant state i.e.,$y[k],\ y^a[k] \in \mathbb{R}^m$. Therefore, the residue $r[k],\ r^a[k] \in \mathbb{R}^m$. When no attack is taking place, the mean and covariance of residue are respectively $E[r_k] = 0$ and $E[r_k r_k^T] = \Sigma$. Using the subscript i, we denote the i-th sensor as $y_i[k]$ where $i \in \{1, 2, \ldots, m\}$. Consequently, we have $r_i[k] \sim \mathcal{N}(0, \sigma_i^2)$. Here, σ_i is the i-th diagonal entry of the covariance matrix Σ. The condition for detecting a false data injection attack using CUSUM detector [48] is:

$$S_i[k] = max(0, S_i[k-1] + \mid r_i^a[k] \mid -b_i) \quad \text{if } S_i[k-1] \leq Th_i \tag{10}$$

$$= 0 \quad \text{if } S_i[k-1] > Th_i \tag{11}$$

The test sequence S_i is initialized with 0 for all $i \in \{1, 2, \ldots, m\}$. Th_i and b_i are the threshold and bias selected for the i-th sensor. So, basically, CUSUM detector checks whether a certain sensor is under attack. When the cumulative sum sequence S_i exceeds Th_i, an alarm is triggered to raise an attack situation.

The efficacy of this detector depends on the bias b_i and the threshold Th_i. Since $\mid r_i^a[k] \mid$ is non-negative, if a sufficiently large value is not selected for b_i, the test sequence S_i may grow unboundedly. This inherent unboundedness of CUSUM may lead to false alarms. Because, due to some measurement and process noise, the value of $\mid r_i^a[k] \mid$ can be greater than 0 even if an attack is not taking place. Therefore, first, the value of b_i must be selected wisely relative to the characteristics of the residue r_i. Following this, a suitable Th_i needs to be computed to achieve a desired FAR.

The authors of [37] established a lower bound on b_i as $b_i > \bar{b} = \sigma_i\sqrt{2/\pi}$ in Theorem 1 in [37]. Once, b_i is determined, the value of the threshold Th_i has to be computed such that the false alarm rate never crosses a desired value. To do so, we define *run length* κ_i of CUSUM (Eq. 11) as the number of iterations needed to reach $S_i[k] > Th_i$ i.e.

$$\kappa_i = inf\{k \geq 1 : S_i[k] > Th_i\} \tag{12}$$

The average run length (ARL_i) is the expected value κ_i which is related to FAR as $ARL = 1/FAR$. Considering the desired FAR as FAR^*, we need to find out Th_i such that $ARL_i = 1/FAR^*$ provided $b_i > \bar{b_i}$. Authors of [37] presented a Markov chain approach for approximating ARL_i to determine the pair $< b_i, Th_i >$ such that Eq. 12 holds.

3.2 *Variable Threshold-Based Attack Detector*

A different line of approach was presented in [35] to synthesize the threshold for residue-based detectors to enhance TPR and reduce FAR. As a potential example of a targeted performance degrading attack, they consider the situation when the reference point of a controller changes due to the occurrence of some event. For example, if the driver rotates the steering wheel, the yaw rate of the vehicle needs to be changed to maintain the lateral dynamics of the vehicle. For such kinds of systems, an attacker can obstruct the vehicle from reaching the proximity of the new reference by injecting even smaller faults at the later stage of the system dynamics (when nearing the reference). From the perspective of designing a security-aware system, this brings in an interesting trade-off. Assume we want to design a static threshold-based detector where a constant threshold will be used throughout. We look into two cases. First, a lower-valued threshold is determined considering the required false data to be injected at the later phase of settling time. In this case, any process or measurement noise induced by the environmental disturbances in the system will be considered an attack. This will lead to false alarms. Second, a higher-valued threshold is selected considering the required attack amount at the earlier phase of settling time. This will help an attacker easily bypass the detector. The attacker can inject a sequence of small false data to make the system unsafe (as demonstrated in Fig. 5). Such scenarios have motivated the authors of [35] to design a variable threshold-based detector that may ensure reduced FAR while identifying even small attack efforts that may lead to potential performance degradation.

As a motivating example, we again consider the same example trajectory tracking control (TTC) system. We can see in Fig. 6a, that due to the process noise $w[k]$ and measurement noise $v[k]$ (Eq. 2), the violation in system's desired performance is negligible. This is due to the intrinsic robustness of the controller. On the other hand, we can see the system gradually becomes unstable when the system is under the influence of a smart attacker (Fig. 6a). Consider $three$ such possible residue based detectors: with the smaller threshold th, the bigger threshold Th and the variable threshold curve v_{th} in Fig. 6b. The detector considers even the harmless noise as an attack when th is used, while the actual attacker could bypass the detector when

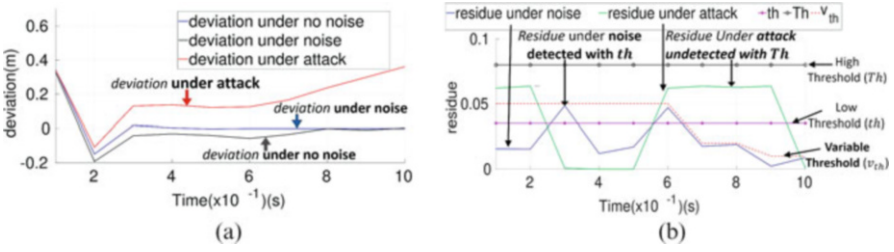

Fig. 6 Noise and attack simulation on trajectory tracking system. (**a**) Effect of noise and attack. (**b**) Static vs dynamic threshold

Th is used. However, using the variable threshold curve v_{th} (dotted black line in Fig. 6b), the attack does not remain stealthy while harmless noise is allowed to pass, reducing the false alarm rate. With this motivation, the authors of [35] presented two greedy approaches for synthesizing variable thresholds by leveraging formal methods like Satisfiability Modulo Theory (SMT) [49]. The following paragraph contains detailed explanations of those methods.

Given the closed-loop system dynamics in Eq. 2, and the reference point x_{des}, the target performance criteria pfc is to reach some n-dimensional closed ball (polytope) $\mathcal{B}_\epsilon(x_{des})$ with radius $\epsilon > 0$ around x_{des} (i.e., the closed region $\{x \in \mathcal{R}^n \mid\mid x - x_{des} \mid\mid \leq \epsilon\}$) within a finite number of iterations, starting from an initial state $x[0] \in \mathcal{I} \subset \mathbb{R}^n$. Hence,

$$pfc : x[l] \in \mathcal{B}_\epsilon(x_{des}), \text{ where } l > 0 \text{ is the finite number of iterations.}$$

The attacker's objective would be $x[l] \notin \mathcal{B}_\epsilon(x_{des})$ after l closed-loop iterations (Eqs. 5–8). The property pfc captures both control performance and stability criteria. Assume the system already has some rudimentary monitoring scheme, like a range monitor for the sensor measurements, in place. Let us denote such monitoring rules as mdc. Authors of [35] present two counter-example guided methods to synthesize variable thresholds. They generate a stealthy attack vector i.e., a sequence of attacks that can ensure violation of pfc while mdc fails to detect it. Using this attack vector, they include a new threshold to the variable threshold set and again generate another attack vector. This step is continued until no attack vector can be generated with the current set of thresholds. We first explain the attack vector generation method using SMT [35].

The following are fed to Algorithm 1 as input: i) dynamics of the plant P, ii) the controller gain K to control the plant P, iii) estimator gain L, iv) desired performance criteria pfc of the closed-loop system, v) specification of existing attack monitor mdc, vi) set of thresholds Th (this is initially a null set), and vii) finite duration T for satisfying pfc. The system states, estimated states, and control inputs are initialized in line 2. Note that, u^a differs from $\tilde{u}^a$ by the fact that u^a is the control input before being communicated to the plant, and $\tilde{u}^a$ is the control input which is modified by the attacker and received by the plant (see Eq. 8). Consider attack is taking place at every iteration in $\{1, 2, \ldots, T\}$. At every iteration $k \in \{1, \ldots, T\}$, the variable $a_y[k]$ and $a_u[k]$ signifying false data are assigned a value non-deterministically (line 4). Following Eq. 6, the false data $a_y[k]$ is added to the measurement $y^a[k]$ which is transmitted from plant to controller (line 5). The controller computes estimated measurement $\hat{y}^a[k]$ and thereby the residue $r^a[k]$. System states $x^a[k+1]$ and estimated states $\hat{x}^a[k+1]$ are updated in lines 8–9. Note that since estimator and controller reside in the same embedded platform and we are only considering network-level attack on CPS, x^a is updated with $\tilde{u}^a$ while $\hat{x}^a$ is updated with u^a. Finally, control input $u^a[k+1]$ is computed in line 10 and modified control input $\tilde{u}^a[k+1]$ is calculated by introducing $a_u[k]$ to $u^a[k+1]$ in line 11. This way, the closed-loop system progression for T iterations is unrolled and symbolically represented. We say that an attack is stealthy but successful

Algorithm 1 Attack vector synthesis [35]

Require: Plant P, controller K, observer L, Control property pfc, existing monitoring constraint mdc, computed threshold vector Th, attack duration T
Ensure: Attack vector $\mathcal{A}$(if it exists, otherwise NULL)

1: **function** ATTVECSYN(P, K, L, Th, pfc, mdc, T)
2: $x^a[1] \leftarrow \mathcal{I}$; $\hat{x}^a[1] \leftarrow 0$; $u^a[1], \tilde{u}^a[1] \leftarrow -K\hat{x}^a[1]$; ▷ Initialization
3: **for** $k = 1$ to T **do**
4: $a_y[k], a_u[k] \leftarrow non - deterministic\_choice$;
5: $y^a[k] \leftarrow Cx^a[k] + D\tilde{u}_a[k] + a_y[k]$;
6: $\hat{y}^a[k] \leftarrow C\hat{x}^a[k] + Du^a[k]$;
7: $r^a[k] \leftarrow y^a[k] - \hat{y}^a[k]$;
8: $x^a[k+1] \leftarrow Ax^a[k] + B\tilde{u}^a[k]$;
9: $\hat{x}^a[k+1] \leftarrow A\hat{x}^a[k] + Bu^a[k] + Lr^a[k]$;
10: $u^a[k+1] \leftarrow -K\hat{x}[k+1]$;
11: $\tilde{u}^a[k+1] \leftarrow u^a[k+1] + a_u[k]$;
12: **end for**
13: $\mathbb{A} \leftarrow$**assert**$((\forall Th[p] \in Th, \|r^a[p]\| < Th[p]$ && $mdc) \rightarrow pfc)$
14: **if** $\mathbb{A}$ is *violated* **then**
15: **return** $\mathcal{A} \leftarrow \begin{bmatrix} a_u[1] \cdots a_u[T] \\ a_y[1] \cdots a_y[T] \end{bmatrix}$;
16: **else**
17: **return** NULL;
18: **end if**
19: **end function**

when predicates $||\ r^a[k]\ ||< Th[k]$ and mdc are satisfied, but pfc is violated. Negation of this is modeled by assertion $\mathbb{A}$ in line 13. The function ATTVECSYN() in Algorithm 1 thus non-deterministically models all possible T consecutive closed-loop executions under stealthy attacks. After this, the assertion on the system states and residue is given as input to an SMT tool with the assert clause. If the assertion is violated, the algorithm gives as output a successful stealthy attack vector (line 15). Else, it returns NULL (line 18) which signifies that the performance criteria pfc of the system can not be violated by any stealthy attack of duration T samples. Using this algorithm, the authors of [35] presented two greedy algorithms to synthesize a set of variable thresholds.

3.2.1 Pivot-Based Threshold Synthesis Method

This method generates a new threshold at every iteration. The steps of the method are demonstrated in Figs. 7, 8, 9, and 10 and discussed in detail below.

Step 1: Initially, the threshold set is considered to be empty. The function ATTVECSYN() in Algorithm 1 is called with the empty threshold set and other parameters. If an attack vector $\mathcal{A}$ is returned, it implies that the existing monitor mdc fails to detect the successful attack and a new threshold for the residue-based detector is needed. The maximum residue generated by the current attack vector

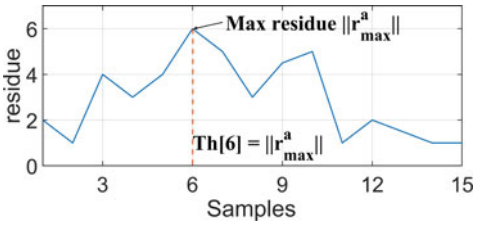

Fig. 7 Step 1 (pivot-based threshold synthesis): from the 1-st attack vector, add 1st threshold to Th that can detect maximum residue

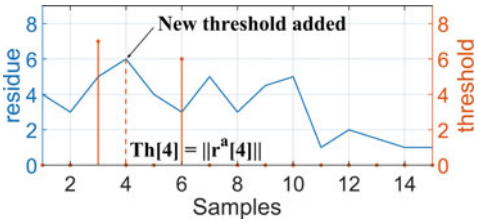

Fig. 8 Step 2 (pivot-based threshold synthesis): check if attack vector exists with new Th, look for new threshold on LHS of existing ones keeping monotonic decreasing order intact

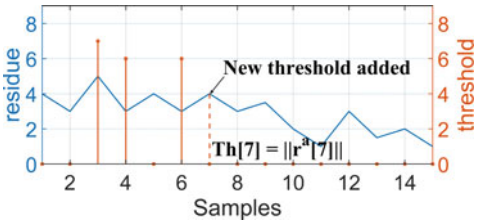

Fig. 9 Step 3 (pivot-based threshold synthesis): check if attack vector exists with new Th and step 2 fails, look for new threshold on RHS of existing ones keeping monotonic decreasing order intact

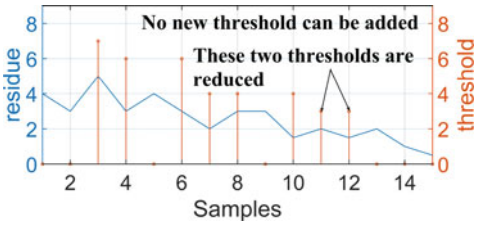

Fig. 10 Step 4 (pivot-based threshold synthesis): check if attack vector exists with new Th and step 3 fails, modify an existing threshold keeping monotonic decreasing order intact

is selected as the first threshold (Fig. 7). This ensures that the new threshold will be able to detect the current attack vector.

Step 2: The function AttVecSyn() is called again with the updated threshold set to check if any attack vector exists. If so, it implies that the current threshold set is not enough to detect all attacks, and a new threshold must be included to Th. As we aim to generate a monotonic decreasing set of thresholds, first we see if we can add a new threshold on the left-hand side of the existing ones such that the monotonic decreasing order is maintained. It is demonstrated in Fig. 8. For any of the existing thresholds $Th[p] \in Th$, we try to find out whether the current attack has produced any residue $|| r^a[k] || \geq Th[p]$ for $k \leq p$. Multiple such candidate residues may exist. The maximum of them is considered to be the new threshold. This new threshold ensures the current attack will be detected.

Step 3: In step 2, if any higher valued threshold can not be found on the left-hand side of any existing threshold, the method checks if a new threshold can be added on the right-hand side of some existing threshold (Fig. 9). For any of the thresholds $Th[p] \in Th$, we try to find out whether the current attack has produced any residue $|| r^a[k] ||\geq Th[p]$ after p-th instance, i.e., $k > p$. A new threshold is added at i if only $r^a[i]$ is at least as much as $Th[k]$ for all $k > i$. This ensures the monotonic decreasing order.

Step 4: If no new threshold can be added following the rules in Steps 2 and 3, then one of the existing thresholds needs to be modified to detect the current attack vector. To do so, the proposed approach computes the difference between existing thresholds $Th[p] \in Th$ and the corresponding residue $|| r^a[p] ||$. The threshold $Th[i]$ is selected as a candidate if $Th[i] - || r^a[i] ||$ is minimum among all $Th[i] \in Th$ and the value of $Th[i]$ is comparatively reduced than earlier. If this modification violates the monotonic decreasing property of Th, all the $Th[p] \in Th$ for $p > i$ are reduced. This is demonstrated in Fig. 10.

Steps 2–4 are repeated until the function AttVecSyn() returns no attack vector with the modified threshold set. This returned threshold set Th is the final one. Since this approach may take a longer time to converge, the authors of [35] proposed another greedy approach that we discuss next.

3.2.2 Step-Wise Threshold Synthesis Method

While the previous approach computes a single threshold at each iteration, this method computes a sequence of thresholds together at each step. The steps of this method are pictorially presented in Fig. 11. Let us discuss the steps in detail.

Step 1: Here as well, the threshold set is initialized to be empty. The function AttVecSyn() in Algorithm 1 is called with the empty set. If it returns an attack vector, it implies that there is a need for a threshold to detect this attack vector. For introducing the first sequence of thresholds, the maximum value among the $|| r^a[i] ||$'s where $1 \leq i \leq T$ is selected, say $|| r^a[j] ||$. The first sequence of thresholds is computed as $Th[p] = || r^a[j] ||$, for all $p \in \{1, .., j\}$. This is demonstrated in Fig. 11a. The name of the method is justified by the fact that the threshold set computed using this method will always looks like steps.

Step 2: With the updated threshold set Th, the function AttVecSyn() is again called to check if the new threshold is enough to detect every attack vector. If the function returns an attack vector, it indicates the need for new thresholds. Let $Th[i]$ be the last non-zero threshold value. To create a new step, this method finds out maximum $|| r^a[k] ||$ for $k > i$ such that $|| r^a[k] ||\leq Th[i]$. Say, the maximum is $|| r^a[j] ||$. The threshold set is then updated as $Th[p] = || r^a[j] ||$ for all $i < p \leq j$ (Fig. 11b). This ensures the desired monotonic decreasing order property of Th.

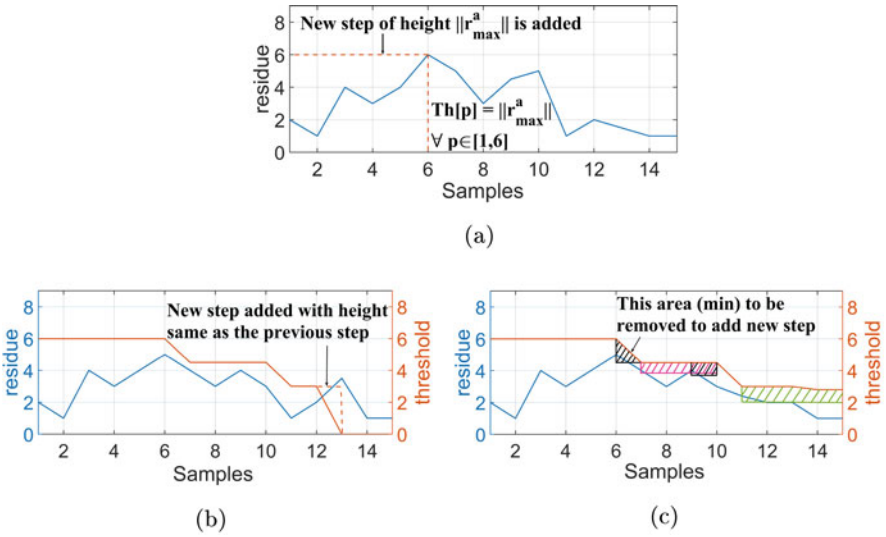

Fig. 11 Step-wise threshold synthesis. (**a**) Step 1: from 1-st attack vector, add 1st step of thresholds in Th. (**b**) Step 2: check if attack vector exists with new threshold Th, look for new step downwards to maintain the monotonic decreasing order. (**c**) Step 3: check if attack vector exists with new threshold Th, and step 2 fails, create new steps out of the old ones by keeping the monotonic decreasing order intact

Step 3: A situation may occur when no new step can be generated in the threshold set. This can happen when there is no zero element in Th. In such cases, the height of some existing steps needs to be modified to ensure that the current attack will be detected with the modified threshold set. Instead of reducing the height of an entire step, we break a portion or the whole step whichever involves minimum effort i.e., the minimum area under the threshold curve that can be removed to detect the current attack. From Fig. 11, it can be seen that the thresholds in Th create an area under the threshold curve. At each sampling instant i, an area $Area_i$ is computed as follows. Find $p, i < p \leq T$ such that for all $k > p$, $Th[k] \leq || r^a[i] ||$ but for all $k \leq p$, $Th[k] > || r^a[i] ||$. $Area_i$ is the segment under threshold curve Th from i-th to p-th sample. The sampling instant for which this area is minimum is selected, say that is the j-th instance. By removing $Area_j$, new step is generated as $Th[l] = || r^a[j] ||$ for all $l \in (j, p]$. This is demonstrated in Fig. 11c.

Steps 2–3 are repeated until no new attack vector can be found upon calling AttVecSyn() every time Th is updated. In [35], authors analyzed that the step-wise method converges must faster than the pivot-based one. Also, the step-wise method performs better in terms of FAR than the pivot-based method.

4 AI-Based Adaptive Attack Detection

The previous section brings forth some algorithmic and heuristic-based approaches to improve the detection tasks in CPSs. It introduces the concept of an adaptive detection technique. These heuristic-based approaches definitely improve the detection when compared to a fixed threshold-based detector, but it does not formalize or quantify the improvements. In an attempt to achieve so, the authors of [36] presented an attacker-defender game to guide the adaptive threshold selection problem though the proposed detector is not evaluated on any closed-loop CPS. Moreover, the optimization problem for threshold selection is solved in real-time and may cause computation overhead. Reinforcement learning is already been successfully used in the domain of estimation [50], energy efficiency [51] of safety-critical CPSs with real-time requirements. An RL-based adaptive threshold-based attack detector learns from the affected system dynamics and adaptively tunes the threshold of the residue-based anomaly detectors. The main motivation behind considering an RL-based strategy is the following challenge. The false data injected into sensors and actuators by the stealthy attacker are highly system-specific, random, and do not follow any statistical distribution. Therefore, the parameters of the adaptive attack detector cannot be directly derived from the injected false data signature. The performance of the proposed detector depends on how well it is trained against the optimal attack vectors. Thus, we also design an RL agent for mimicking the attacker's behavior during the training phase.

Let, for the discrete LTI system shown earlier in Eq. 2, the estimation error $e[k]$ be defined as $e[k] = (x[k] - \hat{x}[k])$. The Gaussian assumptions of noise and initial states ($x[0] \sim \mathcal{N}(0, \Sigma_{x[0]})$) ensure that $e[k] \sim \mathcal{N}(0, \Sigma_e)$ (steady state covariance matrix of this estimation error is Σ_e). Therefore, the system residue $r[k] = Ce[k] + v[k]$. Being a linear function of two other independent gaussian random variables estimation error and measurement noise, the residue is also normally distributed i.e., $r \sim \mathcal{N}(0, \Sigma_r)$, where, $\Sigma_r = E[r[k]r[k]^T] - E[r[k]]E[r[k]]^T = E[(Ce[k])(Ce_{=}[k])^T] + E[v[k]v[k]^T] = C\Sigma_e C^T + \Sigma_v$.

As a popular detection scheme, the χ^2-test can be used on $r[k]$ to find out how anomalous $x[k]$ is i.e., whether it is affected by injected false data. This helps one understand how *bad* the estimated output is compared to the actual controlled-plant output according to the χ^2-test. Let $g[k]$ denote the χ^2-test result at k-th sample and $g[k] = \sum_{i=k-l[k]+1}^{k} r_i^T \Sigma_r^{-1} r_i$. A window size of $l[k]$ is considered during the χ^2-test at k-th sampling instance since taking historical data into account produces a more accurate estimation compared to only considering instantaneous data. Consider that there are m available sensors to sense different plant outputs (i.e., $m \leq n$, n being the dimension of the system). The degree of freedom (DOF) for this test is $m \times l[k]$. When there is no FDI attack, $g[k]$ follows χ^2 distribution with mean $ml[k]$ (Fig. 12) since $r[k]$ and $e[k]$ follows 0 mean Gaussian distribution as discussed above. If $Th[k]$ is the threshold that is chosen at k-th sampling instance, $g[k]$'s probability density function (PDF) along with its cumulative distribution function (CDF) w.r.t. $Th[k]$ can be defined as,

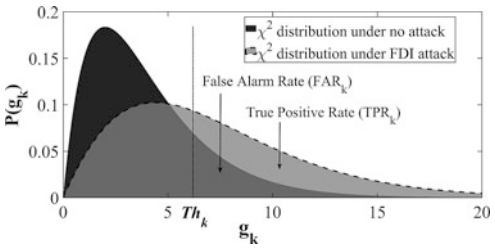

Fig. 12 χ^2-distribution

$$P(g[k]) = \frac{g[k]^{\frac{ml[k]}{2}-1} e^{-\frac{g[k]}{2}}}{2^{\frac{ml[k]}{2}} \Gamma(\frac{ml[k]}{2})} \tag{13}$$

$$P(g[k] \leq Th[k]) = \frac{\gamma(\frac{ml[k]}{2}, \frac{Th[k]}{2})}{\Gamma(\frac{ml[k]}{2})} \tag{14}$$

Here, Γ and γ are ordinary and lower incomplete gamma functions respectively [52, 53]. It is considered to be a *false alarm* when $g[k] > Th[k]$ even in the absence of an attacker. We quantify this with the *false alarm rate* (FAR), calculated with the ratio of the number of times a false alarm is raised falsely and the total number of alarms raised. We denote the FAR at k-th sample where the χ^2-test result $g[k]$ is compared with the threshold $Th[k]$ as $FAR[k]$. In Fig. 12, the black area under the solid curve and the grey area under the dashed curve represent the distribution of $g[k]$ under no attack and attack respectively. Therefore, $FAR[k]$ should be the fraction of area under the probability distribution curve of un-attacked $g[k]$ that is constrained by $g[k] > Th[k]$; thus computed as $FAR[k] = 1 - P(g[k] \leq Th[k])$.

As proven in Theorem 1 in [38], the spurious data $a^y[k]$ and $a^u[k]$ added by the attacker to the sensor, and the actuator transmissions respectively introduce non-centrality to the actual χ^2 distribution of system residue. The gray area under the dashed curve in Fig. 12 is the distribution of $g^a[k]$ obtained from the residue $r^a[k]$ (Eq. 7). The resulting $g^a[k]$ is compared to $Th[k]$ in order to flag an attack. Following Corollary 1 in [38], the variance of $g[k]$ is $\sigma[k] = 2ml[k]$ and variance of $g^a[k]$ is $\sigma^a[k] = 2(ml[k] + 2\lambda[k])$, where $\lambda[k] > 0 \implies \sigma^a[k] > \sigma[k]$. From the Theorem 1 in [38] we can see that the expected deviation of $g^a[k]$ from its mean is more than the expected deviation of $g[k]$ from $ml[k]$ which makes the distribution of P($g^a[k]$) wider and thereby flatter (since the area under both curves is unity). Therefore, P($g^a[k] > Th[k]$) > P($g[k] > Th[k]$) as shown in Fig. 12. As the window size $l[k]$ increases, the PDF of $g^a[k]$ becomes even flatter and hence more distinguishable from the PDF of $g[k]$. So, intuitively speaking, the non-centrality of χ^2 distribution improves $TPR[k]$ i.e., attacks are more detectable for a properly chosen window size $l[k]$ parameter.

For an optimally chosen $l[k]$, the non-centrality of $g^a[k]$ is more evident and hence produces better TPR for a certain threshold $Th[k]$. We can also optimally choose a $Th[k]$ to attain the minimum possible $FAR[k]$ (for a certain window

size) during the absence of an attack (i.e., when only noise is present, in PDF of $g[k]$) and change it during the attack to attain the maximum possible TPR. The main trick here is to understand that the system is under attack more often in the true positive case and reduce false alarms. Therefore, the pertinent problem becomes *how to achieve the above by suitable choice of threshold* $Th[k]$ *and the test parameter* $l[k]$ *in order to identify the attack as quickly as possible without too many false alarms.* So, given a closed-loop CPS, one needs to learn when is the system under some stealthy FDI attack and when it is running normally. The work in [38] leverages the non-centrality property of $g^a[k]$ and learns when the system is becoming affected by a successful and stealthy FDI attack. The problem of synthesizing an optimal detector at every k-th simulation step can thus be formulated as the following optimization problem.

$$J_t = \max_{l[k], Th[k]} \; w_1 \times TPR[k] - w_2 \times FAR[k] \text{ s.t. } FAR[k] < \epsilon,\; l[k] < l_{max} \tag{15}$$

The cost function J_t aims to minimize $FAR[k]$ and maximize $TPR[k]$ at every simulation step. Here, w_1, w_2 are respective non-negative weights assigned to TPR and FAR depending on attacked (TPR increment gets more importance) and non-attacked (FAR reduction gets more importance) situations. ϵ is the maximum allowable FAR and l_{max} is the maximum allowed χ^2 window length. At each k-th step, given the current measurement $y[k]^a$, the solution of the above optimization problem is a pair $< l[k]^*, Th[k]^* >$, where $l[k]^*$ and $Th[k]^*$ are the optimal χ^2 window length and threshold respectively that lead to maximum $TPR[k]$ and minimum $FAR[k]$ w.r.t. current measurement of the system states. But this formulation has to work for all possible FDI attacks within the sensor and actuation limits. The authors in [38] take a nice approach to ensure that the detection works even in the worst case. They learn the optimal attack possible at k-th iteration that maintains its stealth but imparts the most significant damage to the system safety. The following subsection explains how such an attacking policy can be learned.

4.1 Optimal Attack Policy Design

As we discussed in Sect. 2, the attacker's motive is to steer the system beyond the safe set X_S while trying to remain stealthy by *reducing the TPR* i.e., fooling the detector. Given the sensor measurement $y^a[k-1]$, we present this attack estimation problem as the following optimization problem.

$$\begin{aligned} J_a = \max_{a^y[k], a^u[k]} \quad & -w_1 \times TPR[k] + w_2 \times FAR[k] + \\ & \sum_{i=0}^{\infty} (\mid x^a[i+1] \mid - X_S \mid)^T W_3 (\mid x^a[i+1] \mid - \mid X_S \mid) \end{aligned} \tag{16}$$

$$\text{s.t.} \quad x_0^a,\ \hat{x}_0^a \in X_R \tag{17}$$

$$u^a[k] = -K\hat{x}^a[k],\ \tilde{u}^a[k] = u^a[k] + a^u[k],\ |\ u^a[k]\ |,$$
$$|\ \tilde{u}^a[k]\ | \leq \epsilon_u\ \forall k \in [0,\ \infty] \tag{18}$$

$$y^a[k] = Cx^a[k] + D\tilde{u}^a[k] + v[k] + a^y[k],$$
$$|\ y^a[k]\ | \leq \epsilon_y\ \forall k \in [0,\ \infty] \tag{19}$$

$$r^a[k] = Cx^a[k] - C\hat{x}^a[k],\ g^a[k] \leq Th[k]\ \forall k \in [0,\ \infty] \tag{20}$$

$$\hat{x}^a[k+1] = A\hat{x}^a[k] + Bu^a[k] + L(Cx^a[k] - C\hat{x}^a[k]),\ \forall k \in [0,\ \infty] \tag{21}$$

$$x^a[k+1] = Ax^a[k] + B\tilde{u}^a[k],\ \forall k \in [0,\ \infty] \tag{22}$$

Here, w_1 and w_2 are weights that denote the relative priorities of the attack initiative similar to the optimal threshold cost function J_t (Eq. 15). This is because our intention is to design an optimal and stealthy FDI attack for a system equipped with the adaptive detector designed above. While an attacker tries to decrease $TPR[k]$ (and increase $FAR[k]$ simultaneously as a by-product), the detector's objective is to increase $TPR[k]$ and decrease $FAR[k]$ based on the value of $\lambda[k]$ (Eq. 15). The last component of J_a is important to establish it as the worst-case attack. It accounts for the deviation of the current system state from the safety boundary X_S using a quadratic weighted distance metric where W_3 is a diagonal matrix with relative weights signifying the safety-criticality of each dimension. The constraints in 18 and 19 ensure that the attack efforts are practical, within the allowable ranges and utilize the LTI system properties. In case of an invalid or beyond the range sensor data and control signal, their effects will be trimmed by the saturation limit and won't produce a desirable effect of the attack. An intelligent adversary's another aim is to remain stealthy, thus bypassing the detector. This is taken into account in constraint 20 while estimating the optimal attack. The constraints 21 and 22 ensure system progression following Eqs. 5–8.

4.2 The MARL Based Framework

This section discusses the Multi-Agent Reinforcement Learning (MARL) based implementation that is the methodology to build an adaptive threshold-based detection module as discussed in [38]. The goal of the adaptive detector is to detect a stealthy FDI attack before it is successful in making the system unsafe. In the introduction of Sect. 4, we have explained how changing the detection thresholds by leveraging the non-centrality of the system residue can be useful to increase TPR and reduce FAR. We utilize that notion here. The detection and attacker modules implicitly learn how the system model behaves normally and under FDI attacks by analyzing system outputs, states, residue, etc. The smart attacker module should challenge the adaptive detection module by posing the most stealthy yet effective

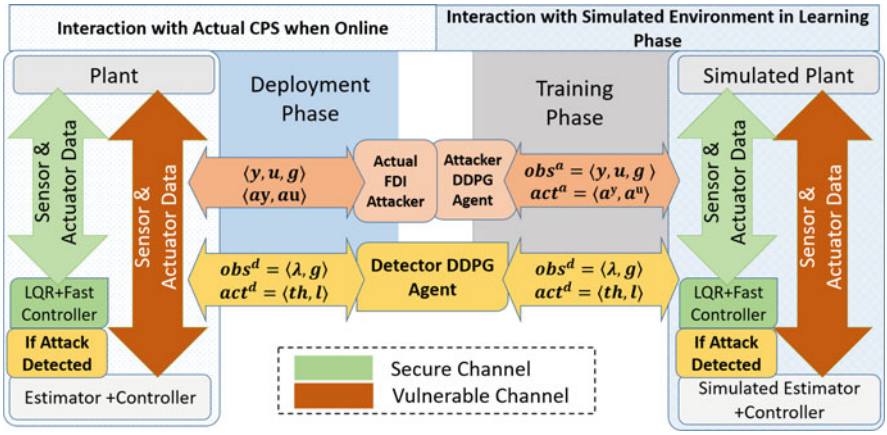

Fig. 13 The RL-based methodology [38]

FDI attacks depending on current system behavior. On the other hand, the intelligent detector module learns the best possible FDI attack on the CPS by observing the non-centrality of the χ^2-distribution of system residue and it adaptively changes the threshold to expose the attacker with a promise of increased TPR and reduced FAR. RL algorithms are capable of automatically updating their strategy by learning from prior experiences. Moreover, since labeled data for falsified system states are not available, a simulation environment for the targeted system model can be useful. This explains how integrating an MARL framework like this can be useful in the context of a security-aware CPS design. In the context of the timing overhead, it is reasonably low when we use a trained RL agent for inferencing at run time as seen in previous literature [51].

A plant-controller closed-loop system equipped with a χ^2-based detector (as shown in Fig. 3) is modeled similar to the real-world system under test. This is then used as the environment for the RL agents (both under attack and without attack situations). Individual RL agents are built as part of our methodology to act as the FDI attacker and the adaptive threshold-based detector that run simultaneously in a closed-loop with the system environment (see Fig. 13). These agents (Λ) interact with the environment by observing certain states from the environment (obs) and learn how intelligent choice of action (act) values can influence the environment towards the fulfillment of their objectives i.e., earning higher rewards (Rwd).

RL Agent For FDI Attack Estimation Following the intelligent attacker modeling in Sect. 4.1, it can be considered that the attacker has information about the system characteristics (Eq. 2) and it can manipulate the sensor and actuator data communicated between the plant and controller side. An *Attacker RL Agent* Λ^a intelligently injects false data into the system by observing the sensor data, actuator data, and the χ^2-test result on the system residue. These false data injections should be bounded by the sensor and actuator saturation limits (refer

Eqs. 18 and 19). The *actions* of Λ^a are $act^a = [a^y, a^u]$ and the *observations* are $obs^a = [y, u, g]$ (refer Fig. 13). Here y, u, a^y, a^u denote sensor and actuator data and false data injected in sensor and actuator respectively (refer Eq. 8). Here, g is the χ^2-test result on the system residue r as mentioned earlier. We also provide the last set of actions chosen by the agent Λ^a as its observation, i.e.,$obs^a[k] = [y[k], u[k], g[k], act^a[k-1]]$ at k-th simulation instance. At every simulation instance, the attacker agent aims to choose proper $act^a[k]$ in order to make the system state unsafe without being detected by observing the above data. This *measurement of stealth and success* of the chosen attack effort is captured in the *reward* function $Rwd[k]^a(obs^a[k], act^a[k], obs^a[k+1])$. Like usual RL policies, the agent is *rewarded* against its choice of $act^a[k]$ at every k-th simulation instance following this reward function. The reward function for Λ^a is built following J_a (Eq. 16) i.e.,

$$\begin{aligned} Rwd^a[k] = & -w_1 \times TPR[k] + w_2 \times FAR[k] \\ & + (\mid x^a[k+1] \mid - \mid X_S \mid)^T W_3 (\mid x^a[k+1] \mid - \mid X_S \mid) \end{aligned} \tag{23}$$

The notations carry the same meaning as in Eq. 16 and the index k denotes their value at k-th simulation instance. As described earlier, the two parts of $Rwd^a[k](obs^a[k], act^a[k], obs^a[k+1])$ have opposing objectives. The part $-w_1 \times TPR[k] + w_2 \times FAR[k]$ accounts for stealthiness with minimized FAR and $(\mid x^a[k+1] \mid - \mid X_S \mid)^T W_3 (\mid x^a[k+1] \mid - \mid X_S \mid)$ accounts for the success of a chosen FDI attack action $act^a[k]$. Λ^a moves towards gaining a higher $Rwd^a[k]$ at every simulation instance by choosing an optimal action $act^a[k]$. Therefore, a learned optimal attack estimation agent would ensure that the overall return (the cumulative reward discounted over time) is maximized, which translates to the fact that the attacker agent will estimate the false data in a way such that the system under attack goes unsafe as quickly as possible without being detected. Note that this agent also gives an idea of the actual system states which we can use for secure state estimation.

RL Agent For Adaptive Detection The *Variable Threshold-based Detector Agent* Λ^d also acts on the same system environment under FDI attack as a competitor to the Attacker RL Agent. It chooses an optimal attack detection threshold $Th[k]$ and a suitable χ^2-window $l[k]$ at k-th iteration by observing the χ^2 statistics g of the system residue, current non-centrality λ of this χ^2 distribution and the previous action $act^d[k-1]$ chosen by itself. The intuition behind its formulation is already discussed above and the authors in [38] provide rigorous mathematical proof. Depending on the observations from the attacked environment, the Detector agent chooses a χ^2-window length and threshold. Therefore, we consider the action vector $act^d_k = [Th[k], l[k]]$ and observation vector $obs^d[k] = [g[k], \lambda[k], act^d[k-1]]$ (refer Fig. 13). The reward function $Rwd^d[k](obs^d[k], act^d[k], obs^d[k+1])$ is designed following J_t from Eq. 15, i.e.,

$$Rwd^d[k](obs^d[k], act^d[k], obs^d[k+1]) = \begin{cases} TPR[k] & \text{when } \lambda[k] > \delta \\ -FAR[k] & \text{when } \lambda[k] \leq \delta \end{cases} \tag{24}$$

Here also, the variables carry a similar meaning as in Eq. 15. With a goal to increase the discounted reward over time in an episode, Λ^d chooses $act^d[k]$ at every k-th simulation instance. The structure of the reward function thus ensures that the Neural Network will be trained such that it can always choose its actions to maximize TPR and minimize FAR.

Learning Technique Here we use Deep Deterministic Policy Gradient (DDPG) algorithm [54]. Each DDPG agent consists of an *actor* neural network that deterministically chooses an action (act) by observing the states (obs) of the environment. Another Deep Q-Network(DQN) acts as a critic. In each simulation instance, the actor chooses an action ($act[k]$) by exploring the action space randomly. The transitions from $obs[k]$ to $obs[k+1]$ due to the action $act[k]$ taken are stored in the experience replay buffer along with the corresponding reward $Rwd[k]$ achieved during this transition. Note that this action was chosen based on the maximum possible return. The critic network calculates corresponding Q values in every iteration picking a random batch from the replay buffer and updates itself by the mean square loss between the calculated Q values from consecutive iterations. The actor-network policies are updated using the policy gradient over the *expected Q value return*. Figure 14 depicts the learning flow of a DDPG agent. The training algorithm finally learns the highest expected return from its experiences and then keeps updating the RL policy to output the optimal action that earns the expected maximum return. This, in turn ensures the objective functions we chose to define

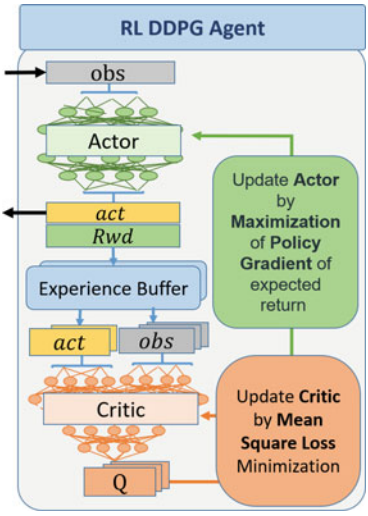

Fig. 14 A DDPG RL agent with actor and critic networks [38]

$Rwd^d[k]$ and $Rwd^a[k]$ in Eq. 15 and Eq. 16 respectively are maximized. Given a secure CPS model, we first train such DDPG agents as discussed with system-specific simulation data so that they can reprise their designated roles in the environment. The *learning process* is collaborative and competitive. The standard DDPG algorithm as in Algorithm 1 in [54] is modified according to the requirements in our case. Interested readers are encouraged to read [54] for a detailed discussion on how DDPG policy optimization works. The work in [55] is also another interesting read to know more about how Agent Environment Cycle (AEC) Games model turn-based games like our MARL setup where the Attacker and Detector compete with each other in every iteration by taking optimal action in a stochastic system environment. Without going into those implementation-specific challenges, we stick to the CPS design aspect without sidelining the main topic of discussion in this section. In the next section, we move on to describe the most plausible next action that should follow an intelligent attack detection in a secure CPS design.

5 Attack Mitigation

To complete the circle of the discussion on the security-aware design of CPS, in this section we briefly talk about system recovery steps to be taken once attack attempts are detected. The idea proposed by most researchers is to switch the system to a *secure mode* once an attack is detected. We explain this idea using Fig. 15. By secure mode, we mean an operational mode where every communication is secured via some cryptographic methods, like MAC, RSA, AES, etc. For schedulability, un-important messages can be dropped in this mode. We assume that the cryptographic methods provide utmost security and that no stealthy attacks are possible in this mode. Therefore, the attack model is to exploit the normal mode of operations

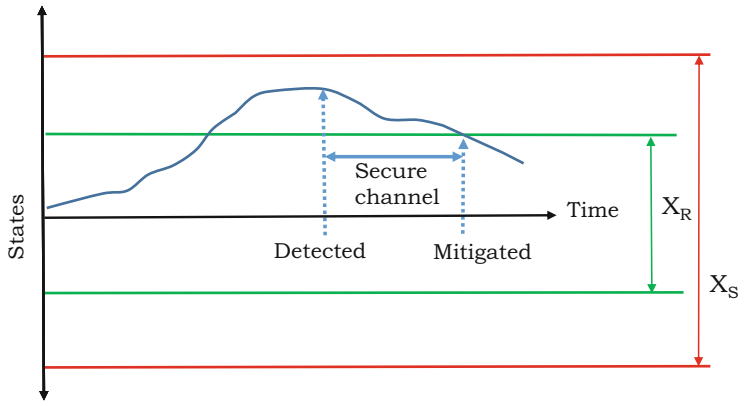

Fig. 15 Attack mitigation through secure channel

and stealthily drive the system to some unsafe region. In Fig. 15, X_R denotes the operating region (in a single dimension) where the system is expected to reside when no attack is taking place. Also, X_S denotes the safety region of the system. An attacker would try to steer the system toward an unsafe zone (see Eq. 9). Once such an attack is detected, the system will be switched to a secure channel. And, during this secure mode, the controller will mitigate the damage done by the attacker (Fig. 15) by steering the system back to its preferable operating region.

While in [34] it is suggested to simply use the available controller gain (K as mentioned in Eq. 2) for attack mitigation, the authors in [38] have suggested using additional control input along with the inputs from the usual feedback controller, following the theory of [56]. As it is shown in Fig. 15, due to an attack, the system may go beyond the preferred operating region X_R. It is desirable to bring back the system from $X_S \setminus X_R$ as early as possible. The motivation behind this is that the duration spent in secure mode must be as minimum as possible. Also, the faster the system is back to the desired operating region, the better will be its average performance. As we have already discussed in Sect. 3, the cryptographic methods incur quite a significant computational and communication load. Thus, it is infeasible to secure every communication. Releasing a secure channel at the earliest will help other control loops to use it. It can be seen in [38] that the use of additional control inputs can actually speed up the recovery process. We briefly demonstrate the idea here.

The authors in [56] have proposed an SMT-based method to pre-calculate a sequence of control inputs that take the system from $X_S \setminus X_R$ to X_R provided during this time, system state should always be retained within X_S. This means safety is guaranteed during recovery. Since $X_S \setminus X_R \in \mathcal{R}^n$, it is not possible to compute control sequence for all possible points in $X_S \setminus X_R$. As a solution to this problem, the authors in [56] proposed a region-wise control synthesis method. They divide $X_S \setminus X_R$ into such sub-regions that the control sequence computed to take the system trajectory from the center of each sub-region to X_R will also work for every other point in that sub-region, as elaborated in Fig. 16. Initially, the length of the control sequence is set as $t = 1$ and the method tries to compute safe control sequences considering the entire $X_S \setminus X_R$ as the source region. If failed, the source region is reduced to half and this process repeats until a safe control sequence can be found. If

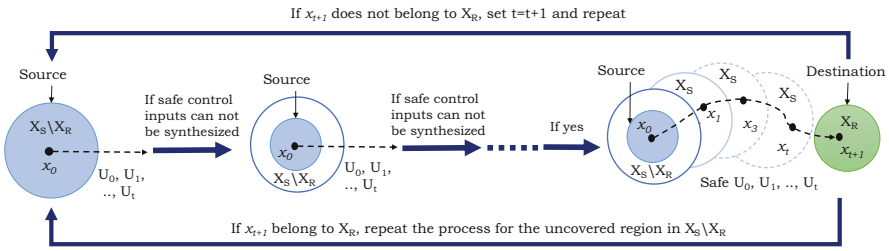

Fig. 16 Fast control action synthesis

the final safe control sequence of length t can not take the system to X_R, the length t is incremented by 1 and the process is repeated until a safe and successful control sequence is found for every sub-region in $X_S \setminus X_R$. This incremental process is also adapted in [35] such that it also ensures that the length of the final control sequence is minimum. This means this is the minimum required sequence of control inputs to bring back the system from $X_S \setminus X_R$ to X_R while safety remains intact.

The above line of work however relies on attacks getting detected in the normal mode of operations. The security model assumes that using the secure mode is costly for the entire system and hence it should be relinquished to other potential subsystems under attack as soon as possible. In that case, how do we guarantee safety from stealthy attacks in a non-probabilistic way? For that, the use of secure and normal modes of operations must be interleaved by design so that every control task switches among such modes periodically. The normal model duration should be chosen in such a way that a stealthy attack cannot drive the system to an unsafe region as verified formally in the model itself [34]. This duration should be followed by a secure mode of suitable duration which helps the system crawl back to its desired performance region [34, 38]. The sequence keeps repeating with such sporadic integrity checks in between [33] coupled with recovery control mechanisms [56]. In the next section, we present a hardware-in-loop experimental set-up on a real-time platform to demonstrate the security-aware design of an automotive CPS considering the variable threshold-based security scheme discussed in Sect. 3.2.

6 A Platform Level Example

In this section, we first discuss a security-aware control implementation on an automotive-grade ECU setup. The setup contains an Infineon Tricore AURIX TC397 ECU where software controllers are mapped. A Hardware-In-Loop (HIL) simulator (ETAS LabCar) is used to emulate the automotive plants. These plants are periodically manipulated by control tasks co-scheduled in a single core of the Infineon ECU. The plant and ECUs are connected via CAN bus, interfaced using the integrated CAN shield in the ECU. The closed-loop setup is depicted in Fig. 17. The ECU is running two control tasks for two automotive plants i.e., Trajectory Tracking Control (TTC) and Electronic Stability Program (ESP). The details of the TTC loop is described in Sect. 3. It controls the longitudinal deviation (D) of the vehicle from a desired trajectory and maintains a target Velocity (V) by changing the acceleration(acc) of the vehicle. The ESP regulates the yaw rate (γ) and side-slip (β) of the vehicle by controlling the steering angle (θ). The system transition, controller, and observer gain matrices of the ESP are taken from [38]. Both of these tasks are implemented as runnables and invoked periodically depending on the sampling period of the discretized plant model considered while designing the LQR controllers, i.e., 100 ms for TTC and 40 ms for ESP. In every sampling period, these statically scheduled tasks are invoked, and control inputs are calculated using the

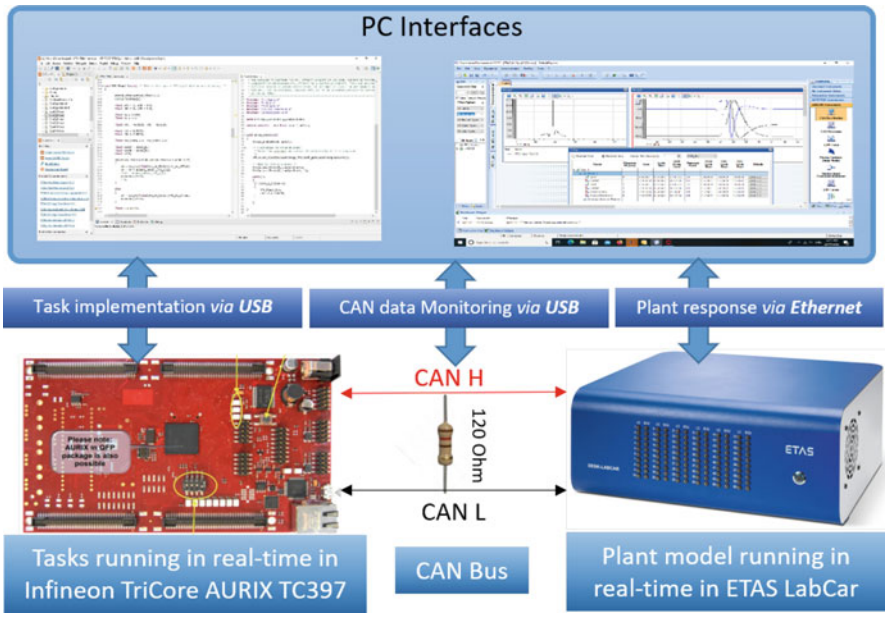

Fig. 17 Real-time automotive test bed

estimated plant states and transmitted via CAN. The control inputs actuate the plant, which emulates itself in real-time in the HIL. The plant states are estimated using estimation tasks that read the received plant output data from the sensor readings transmitted from the plant through CAN. These functions are called before the control tasks in order to supply the estimated plant state to the control tasks. The estimation tasks also run with similar periodicity as their corresponding control calculations. Note that the last calculated control input is also used in order to estimate the current state based on the sensor data. As a detection task, we consider the variable threshold-based detector. The detection task synthesizes thresholds using the pivot-based method as explained in Sect. 3.2. It uses the norm-based generalized detection scheme as mentioned in Eq. 4.

We consider an attack model where an ECU connected to the same CAN bus is compromised. Therefore, it has the capability to inject false data into the CAN transmissions. The attack model is feasible because a compromised ECU can send a real sender to bus-off mode for some interval and mimic the actual sender [12]. We emulated this insider attack scenario by running the attacker routine from a different core of the same ECU. The false data injected were synthesized for TTC using the SMT-based FDI attack synthesis method used in [34, 35]. As mentioned in our attack model in Sect. 2.1, the successful attack criteria while synthesizing the attack vectors is given by Eq. 9. The plot in Fig. 18 shows the outputs of the TTC plant under attack. This is a screenshot taken from the ETAS LabCar environment that shows a 1.3 seconds long stealthy (i.e., 1-norm of the residue under this attack always

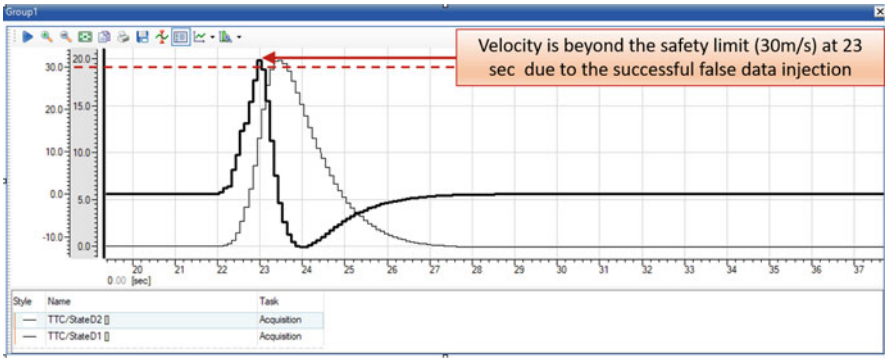

Fig. 18 Successful FDI emulation on TTC

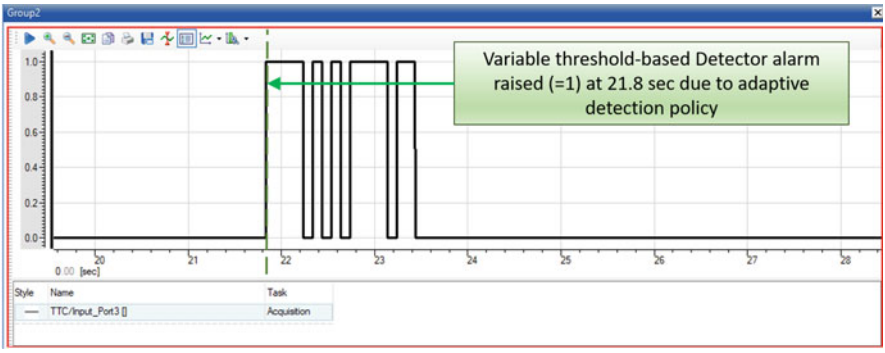

Fig. 19 Adaptive detection of attack before success for TTC

remains below a static threshold of 2.5) attack vector injection (refer Fig. 4, which shows a simulation of the same). The x-axis of Figs. 18 and 19 represents time. The two different scales in the y-axis of Fig. 18 represent two states of TTC i.e., deviation from the trajectory (D) in meters (the first scale from the right) and velocity (V) in meters/sec (the second scale from the right). Y axis of Fig. 19 denotes the detector output. 1 signifies attack detection and 0 signifies no attack scenario. The variable threshold-based detection task is implemented as mentioned earlier in this section. As we can see, both states are starting from 0 units and the 13-length FDI attack vector drives the velocity of the vehicle (the bold one) beyond the safety limit i.e., 30 m/s at 23-rd sec bypassing the static threshold-based detector (in Fig. 18). Whereas, the variable threshold-based detection task selects certain thresholds in real-time which are able to detect this attack attempt (at $\sim$ 21.8 s in Fig. 19) before the attack becomes successful or the system becomes unsafe. Validation of MARL-based detectors and mitigation strategies have been reported in [38] where the full system is simulated in Matlab. In the future, we plan to have a HIL-based validation of the same in this automotive test-bed.

7 Conclusion

This chapter discusses methods for security-aware automotive CPS design leveraging adaptive lightweight attack detection and mitigation schemes. The presented design methodology reduces the compute and communication overhead incurred by standard cryptographic methods suggested by AUTOSAR. We discuss two heuristic-based algorithms for variable threshold selection and a multi-agent reinforcement learning (MARL)-based adaptive threshold selection method in order to increase false data injection attack detectability and decrease false alarm rate in a system. The heuristic-based methods choose thresholds based on solver based vulnerability analysis. Thus, this design technique provides a guarantee that the synthesized variable threshold-based detectors will detect a false data injection attack attempt. The more scalable approach employing the adaptive detector infers a stochastically optimal threshold in order to catch a competing FDI attacker agent, which is designed to falsify the sensed and actuated data. We also discuss a formal method-based attack mitigation scheme which is activated via a secure channel once the attacks are detected. Overall they promote an end-to-end security-aware CPS design idea.

The objectives of this security-aware co-design framework targeting automotive systems had been, (i) lightweight, real-time detection of FDI attacks (ii) while maintaining the least possible false alarm rate; and (iii) guaranteeing the mitigation of the attack-effect as early as possible so that (iv) the compute and communication overhead incurred by the cryptographic schemes are reduced. We discuss the evaluation of the variable threshold-based detection technique in a real-time automotive test bed in order to demonstrate its applicability. Essential future extension of such work is to test the performance of the proposed RL-based adaptive detection and formal mitigation units in this automotive test bed.

Acknowledgments We acknowledge generous grants received from *IHUB NTIHAC Foundation—IIT Kanpur* and *Meity (Grant No. AAA.22/8/2021-CSRD-Meity)* for partially supporting this work.

References

1. Möller, D.P., Haas, R.E.: Guide to Automotive Connectivity and Cybersecurity. Springer, Berlin (2019)
2. Reif, K.: Automotive Mechatronics. Springer, Berlin (2014)
3. HPL SC: Introduction to the controller area network (CAN). Application Report SLOA101, pp. 1–17 (2002)
4. Makowitz, R., Temple, C.: Flexray-a communication network for automotive control systems. In: 2006 IEEE International Workshop on Factory Communication Systems, pp. 207–212. IEEE, Piscataway (2006)
5. Ruff, M.: Evolution of local interconnect network (LIN) solutions. In: 2003 IEEE 58th Vehicular Technology Conference, vol. 5, pp. 3382–3389. IEEE, Piscataway (2003)

6. Sumorek, A., Buczaj, M.: New elements in vehicle communication "media oriented systems transport" protocol. Teka Komisji Motoryzacji i Energetyki Rolnictwa. **12**(1), 275–279 (2012)
7. Bo, H., Hui, D., Dafang, W., Guifan, Z.: Basic concepts on AUTOSAR development. In: 2010 International Conference on Intelligent Computation Technology and Automation, vol. 1, pp. 871–873. IEEE, Piscataway (2010)
8. Deng, P., Cremona, F., Zhu, Q., Di Natale, M., Zeng, H.: A model-based synthesis flow for automotive CPS. In: Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems, pp. 198–207 (2015)
9. Chakraborty, S., Al Faruque, M.A., Chang, W., Goswami, D., Wolf, M., Zhu, Q.: Automotive cyber–physical systems: a tutorial introduction. IEEE Des. Test **33**(4), 92–108 (2016)
10. AUTOSAR: Specification of secure onboard communication. AUTOSAR CP Release **R20-11**(969), 1–28 (2017)
11. Munir, A., Koushanfar, F.: Design and analysis of secure and dependable automotive CPS: a steer-by-wire case study. IEEE Trans. Depend. Sec. Comput. **17**(4), 813–827 (2018)
12. Cho, K.T., Shin, K.G.: Error handling of in-vehicle networks makes them vulnerable. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1044–1055 (2016)
13. Lesi, V., Jovanov, I., Pajic, M.: Integrating security in resource-constrained cyber-physical systems. ACM Trans. Cyber-Phys. Syst. **4**(3), 1–27 (2020)
14. Mo, Y., Sinopoli, B.: False data injection attacks in control systems. In: Preprints of the 1st Workshop on Secure Control Systems, pp. 1–6 (2010)
15. Teixeira, A., et al.: Secure control systems: a quantitative risk management approach. IEEE Control Syst. Mag. **35**(1), 24–45 (2015)
16. Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., et al.: Comprehensive experimental analyses of automotive attack surfaces. In: USENIX Security Symposium, San Francisco, vol. 4, pp. 447–462 (2011)
17. Åström, K.J., Wittenmark, B.: Computer-Controlled Systems. Prentice-Hall, Hoboken (1997)
18. Becker, M., Mohamed, S., Albers, K., Chakrabarti, P., Chakraborty, S., Dasgupta, P., et al.: Timing analysis of safety-critical automotive software: the AUTOSAFE tool flow. In: 2015 Asia-Pacific Software Engineering Conference, pp. 385–392. IEEE, Piscataway (2015)
19. Boulanger, J.L.: Industrial Use of Formal Methods: Formal Verification. Wiley, Hoboken (2013)
20. AUTOSAR: Specification of crypto service manager. AUTOSAR FO Release **R22-11**(402), 1–337 (2020)
21. Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., et al.: Experimental security analysis of a modern automobile. In: 2010 IEEE Symposium on Security and Privacy, pp. 447–462. IEEE, Piscataway (2010)
22. Miller, C., Valasek, C.: A survey of remote automotive attack surfaces. Black Hat USA. **2014**, 94 (2014)
23. Miller, C., Valasek, C.: Remote exploitation of an unaltered passenger vehicle. Black Hat USA **2015**, 91 (2015)
24. Mazloom, S., Rezaeirad, M., Hunter, A., McCoy, D.: A security analysis of an in-vehicle infotainment and app platform. In: 10th {USENIX} Workshop on Offensive Technologies (2016)
25. Serag, K., Bhatia, R., Kumar, V., Celik, Z.B., Xu, D.: Exposing new vulnerabilities of error handling mechanism in CAN. In: 30th USENIX Security Symposium, pp. 4241–4258 (2021)
26. Alrabady, A.I., Mahmud, S.M.: Analysis of attacks against the security of keyless-entry systems for vehicles and suggestions for improved designs. IEEE Trans. Veh. Technol. **54**(1), 41–50 (2005)
27. Francillon, A., Danev, B., Capkun, S.: Relay attacks on passive keyless entry and start systems in modern cars. In: Proceedings of the Network and Distributed System Security Symposium. Eidgenössische Technische Hochschule Zürich, Department of Computer Science (2011)
28. Rouf, I., Miller, R.D., Mustafa, H.A., Taylor, T., Oh, S., Xu, W., et al.: Security and privacy vulnerabilities of in-car wireless networks: a tire pressure monitoring system case study. In: USENIX Security Symposium, vol. 10 (2010)

29. Dworkin M.: Recommendation for block cipher modes of operation: The CMAC mode for authentication. Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD (2016). https://doi.org/10.6028/NIST.SP.800-38B
30. Zalman, R., Mayer, A.: A secure but still safe and low cost automotive communication technique. In: Proceedings of the 51st Annual Design Automation Conference, pp. 1–5 (2014)
31. Willsky, A.S., Deyst, J.J., Crawford, B.S.: Two self-test methods applied to an inertial system problem. J. Spacecraft Rockets **12**(7), 434–437 (1975)
32. Giraldo, J., Urbina, D., Cardenas, A., Valente, J., Faisal, M., Ruths, J., et al.: A survey of physics-based attack detection in cyber-physical systems. ACM Comput. Surv. **51**(4), 1–36 (2018)
33. Jovanov I, et al.: Sporadic data integrity for secure state estimation. In: 2017 IEEE 56th Annual Conference on Decision and Control (CDC). IEEE, Piscataway (2017)
34. Adhikary, S., Koley, I., Ghosh, S.K., Ghosh, S., Dey, S., Mukhopadhyay, D.: Skip to secure: securing cyber-physical control loops with intentionally skipped executions. In: Proceedings of the 2020 Joint Workshop on CPS&IoT Security and Privacy, pp. 81–86 (2020)
35. Koley, I., Ghosh, S.K., Dey, S., Mukhopadhyay, D., KN, A.K., Singh, S.K., et al.: Formal synthesis of monitoring and detection systems for secure cps implementations. In: 2020 Design, Automation & Test in Europe Conference & Exhibition, pp. 314–317. IEEE, Piscataway (2020)
36. Ghafouri, A., Abbas, W., Laszka, A., Vorobeychik, Y., Koutsoukos, X.: Optimal thresholds for anomaly-based intrusion detection in dynamical environments. In: International Conference on Decision and Game Theory for Security, pp. 415–434. Springer, Berlin (2016)
37. Murguia, C., Ruths, J.: Characterization of a cusum model-based sensor attack detector. In: 2016 IEEE 55th Conference on Decision and Control, pp. 1303–1309. IEEE, Piscataway (2016)
38. Koley, I., Adhikary, S., Dey, S.: Catch me if you learn: real-time attack detection and mitigation in learning enabled CPS. In: 2021 IEEE Real-Time Systems Symposium, pp. 136–148. IEEE, Piscataway (2021)
39. Zhou, Y., Vamvoudakis, K.G., Haddad, W.M., Jiang, Z.P.: A secure control learning framework for cyber-physical systems under sensor attacks. In: 2019 American Control Conference (ACC), pp. 4280–4285. IEEE, Piscataway (2019)
40. Zhang, L., Chen, X., Kong, F., Cardenas, A.A.: Real-time attack-recovery for cyber-physical systems using linear approximations. In: 2020 IEEE Real-Time Systems Symposium, pp. 205–217. IEEE, Piscataway (2020)
41. Kong, F., Xu, M., Weimer, J., Sokolsky, O., Lee, I.: Cyber-physical system checkpointing and recovery. In: 2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems, pp. 22–31. IEEE, Piscataway (2018)
42. Mohan, S., Bak, S., Betti, E., Yun, H., Sha, L., Caccamo, M.: S3A: secure system simplex architecture for enhanced security and robustness of cyber-physical systems. In: Proceedings of the 2nd ACM International Conference on High Confidence Networked Systems, pp. 65–74 (2013)
43. Zhao, C., Gill, J.S., Pisu, P., Comert, G.: Detection of false data injection attack in connected and automated vehicles via cloud-based sandboxing. IEEE Trans. Intell. Transp. Syst. **23**, 9078–9088 (2021)
44. Mundhenk, P., Paverd, A., Mrowca, A., Steinhorst, S., Lukasiewycz, M., Fahmy, S.A., et al.: Security in automotive networks: lightweight authentication and authorization. ACM Trans. Des. Autom. Electron. Syst. **22**(2), 1–27 (2017)
45. Zheng, B., Deng, P., Anguluri, R., Zhu, Q., Pasqualetti, F.: Cross-layer codesign for secure cyber-physical systems. IEEE Trans. Comput.-Aided Des. Integrated Circuits Syst. **35**(5), 699–711 (2016)
46. Ewing, G.: Reverse-engineering a crc algorithm. https://www.cosc.canterbury.ac.nz/greg.ewing/essays/CRC-Reverse-Engineering.html. Accessed 06 Feb 2021
47. Tunga, R., Murguia, C., Ruths, J.: Tuning windowed chi-squared detectors for sensor attacks. In: 2018 Annual American Control Conference, pp. 1752–1757. IEEE, Piscataway (2018)
48. Page, E.S.: Continuous inspection schemes. Biometrika **41**(1–2), 100–115 (1954)

49. Moura, L.D., Bjørner, N.: Z3: an efficient SMT solver. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 337–340. Springer, Berlin (2008)
50. Ferdowsi, A., Challita, U., Saad, W., Mandayam, N.B.: Robust deep reinforcement learning for security and safety in autonomous vehicle systems. In: 2018 21st International Conference on Intelligent Transportation Systems, pp. 307–312. IEEE, Piscataway (2018)
51. Wang, Y., Huang, C., Zhu, Q.: Energy-efficient control adaptation with safety guarantees for learning-enabled cyber-physical systems (2020). arXiv:200806162
52. Artin, E.: The Gamma Function. Courier Dover Publications, New York (2015)
53. Jameson, G.: The incomplete gamma functions. Math. Gazette **100**(548), 298–306 (2016)
54. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:150902971 (2015)
55. Terry, J.K., Grammel, N., Black, B., Hari, A., Horsch, C., Santos, L.: Agent environment cycle games (2020) arXiv:200913051
56. Fan, C., Mathur, U., Mitra, S., Viswanathan, M.: Controller synthesis made real: reach-avoid specifications and linear dynamics. In: International Conference on Computer Aided Verification, pp. 347–366. Springer, Berlin (2018)