

P4ICS: P4 in-network security for Industrial Control Systems networks[☆]

Lorenzo Rinieri[✉], Andrea Melis^{✉*}, Roberto Girau[✉], Giovanni Pau[✉], Marco Prandini[✉], Franco Callegati[✉]

Department of Computer Science and Engineering (DISI), University of Bologna, Italy

ARTICLE INFO

Keywords:

ICS security
In-network
P4
Modbus
MQTT
DNP3
EtherNet/IP

ABSTRACT

Industrial Control Systems (ICS) are increasingly interconnected with enterprise IT and cloud services, yet their communications remain largely unprotected due to the limited adoption of Transport Layer Security (TLS) and other cryptographic standards. Legacy devices often lack the resources to support TLS, and operators face performance constraints and complex certificate management. To address this gap, we present P4ICS, a framework that provides confidentiality, integrity, and replay protection for industrial protocols by shifting security functions from endpoints into P4-programmable switches. P4ICS transparently parses and protects Modbus, DNP3, EtherNet/IP, and MQTT traffic, establishing switch-to-switch encrypted tunnels that secure untrusted network segments while preserving interoperability with legacy equipment. Our evaluation on an ad hoc physical testbed shows that P4ICS introduces only a modest overhead compared to plaintext communication, while consistently outperforming TLS, reducing delays by about 12% for Modbus and DNP3, 43% for EtherNet/IP, and 47% for MQTT.

By leveraging in-network computing, P4ICS delivers a practical and deployable security layer for Industry 4.0 communications, narrowing the gap between available secure protocol profiles and their limited use in operational ICS.

1. Introduction

Industrial Control Systems (ICS) underpin critical manufacturing and utility processes by connecting field devices (sensors/actuators), Programmable Logic Controllers (PLCs), and supervisory systems (HMI/SCADA) over purpose-built communication stacks. Despite their centrality to modern Industry 4.0, many ICS deployments still rely on legacy protocols designed for isolated, trusted networks, leaving communications exposed when connectivity expands across plant segments, enterprise IT, and cloud services [1]. To secure these channels, the networking community has traditionally relied on end-to-end cryptographic standards. Transport Layer Security (TLS) [2], with its datagram variant DTLS [3], is the de facto standard for end-to-end channel protection on IP networks, providing confidentiality, integrity, and peer authentication. While TLS-secured variants exist for several industrial protocols (e.g., Modbus/TCP Security [4], CIP Security [5], IEC 62351 profiles [6]), empirical studies show that TLS remains sparsely deployed in operational ICS due to performance overheads, device limitations, and the operational burden of key and certificate management [1,7–9]. Legacy PLCs/RTUs often lack the computational headroom or firmware support for modern cryptographic stacks, and

operators are understandably reluctant to introduce changes that might affect deterministic timing.

This motivates alternative approaches that move beyond endpoints. In this context, in-network computing has emerged as a complementary trend: by relocating selected application and security functions into the data plane, it becomes possible to enforce protection directly within the network fabric, reducing latency, improving efficiency, and simplifying endpoint requirements. Recent standardization work has consolidated its motivations and concrete scenarios across telemetry, transport, and security services [10]. In parallel, the now well-established technologies for programmable data planes (e.g., P4 [11]) enable fine-grained parsing and transformation of packets at line rate, creating an opportunity to enforce confidentiality and integrity within the network fabric rather than at resource-constrained endpoints [12].

This paper explores an alternative, pragmatic design point: P4ICS, an in-network security framework that offloads confidentiality and integrity from endpoints to P4-programmable switches. Instead of requiring every ICS device to implement TLS, P4ICS establishes switch-to-switch protection domains that (i) parse common industrial protocols in the data plane, (ii) apply AES-based encryption and HMAC-based

[☆] This article is part of a Special issue entitled: 'Reconfigurable Networks' published in Computer Networks.

* Corresponding author.

E-mail address: a.melis@unibo.it (A. Melis).

integrity to payloads as they traverse untrusted segments, and (iii) preserve plaintext only on short, trusted links to first-hop switches. In doing so, P4ICS aims to close the gap between the availability of secure protocol profiles and their limited adoption in practice, maintaining interoperability with legacy devices while delivering TLS-like protection where it matters most.

Contributions. This work:

- Designs and implements a P4-based pipeline that parses Modbus, DNP3, Ethernet/IP, and MQTT, and applies in-network encryption and integrity verification at line rate using P4 externs. The source code of P4ICS is publicly available.¹
- Presents a deployment and keying model consistent with the Purdue architecture, with selective tunnel activation and replay protection bound to transport-layer sequencing.
- Provides an extensive evaluation on a physical testbed (Hybrid Industrial Digital Twin), comparing P4ICS against plaintext and TLS modes across RTT, resource usage, and power, and against state-of-the-art P4 AES designs.
- Demonstrates a monitoring case study over the Tennessee Eastman Chemical Process that reconciles confidentiality with process visibility needed for anomaly detection.

Paper organization. The remainder of the paper is structured as follows. Section 2 reviews the industrial protocols considered and surveys related work. Section 3 states the problem and deployment realities that limit TLS adoption in ICS. Section 4 details our design goals and challenges. Section 5 defines the system and threat model. Section 6 describes the P4ICS architecture and data-plane implementation. Section 7 evaluates performance and resource overheads, and compares with prior P4 cryptography. Section 8 presents the Modbus monitoring case study. Section 9 discusses key management, security properties, and limitations. Section 10 concludes.

2. Background & related work

2.1. Background on ICS protocols

In this section, we describe the industrial protocols considered in our study. For each protocol, we summarize its origin, network-level characteristics, and security-enhanced versions where available. We then conclude with a synthesis of their adoption trends, highlighting the persistence of insecure deployments.

2.1.1. Modbus

Modbus, developed by Modicon in 1979, is one of the earliest and most widely adopted industrial protocols [13]. It follows a client-server model (historically called master-slave) where the client issues requests and the server responds. The original implementation operated over serial lines (RS-232/RS-485), while Modbus/TCP encapsulates Modbus messages into TCP/IP packets, typically using port 502. The protocol is lightweight: each message contains a function code, address, and data field. It provides no inherent session management, error correction beyond simple checksums, or security features. Due to its simplicity, Modbus is widely used to connect PLCs, RTUs, and sensors to SCADA systems. In 2008, the Modbus Organization introduced the Modbus/TCP Security specification [4], which incorporates TLS to provide confidentiality, integrity, and authentication. However, the secure version is rarely used in practice, mainly because of performance concerns and the large installed base of legacy devices.

2.1.2. DNP3

The Distributed Network Protocol version 3 (DNP3), introduced in the early 1990s and later standardized as IEEE 1815 [14], was designed for utilities and SCADA networks. Unlike Modbus, DNP3 adopts a layered structure with physical, data link, pseudo-transport, and application layers. It typically uses TCP port 20000 when operating over IP, but it can also run over serial lines. Key features include unsolicited reporting (where devices can send data without being polled), time synchronization, and support for file transfer. The data link layer provides reliable delivery with Cyclical Redundancy Check (CRC) calculation, while the pseudo-transport layer segments large messages. In 2020, the IEEE published DNP3-Secure Authentication [15], introducing challenge-response authentication, integrity codes, and AES-256 encryption. Over time, several versions of Secure Authentication have been proposed, with different degrees of adoption. For instance, DNP3 SAV5 (and the earlier v2) suffered from a number of design flaws [16,17] and were never widely deployed. In particular, the multi-user support introduced in SAV5 was later deprecated [18] as it did not achieve its intended goals. The most recent version, SAV6, aims to reduce the complexity of SAV5, but it has not yet been fully standardized and currently lacks complete implementations.

2.1.3. Ethernet/IP

Ethernet/IP, maintained by ODVA, adapts the Common Industrial Protocol (CIP) to standard Ethernet and TCP/IP [19]. It is widely deployed in manufacturing systems because it leverages existing Ethernet infrastructures while supporting real-time communication. Ethernet/IP distinguishes between explicit messages (typically carried over TCP port 44818) for configuration and diagnostics, and implicit messages (carried over UDP port 2222) for time-critical process data. CIP Security, introduced in 2015 [5], adds TLS for explicit messages and DTLS for implicit messages. It defines authentication via X.509 certificates or pre-shared keys.

2.1.4. MQTT

MQTT, first released by IBM in 1999 [20], is a lightweight publish-subscribe protocol optimized for low-bandwidth and high-latency networks. It runs on TCP, using port 1883 by default, with communications mediated by a broker that manages topics and subscriptions. Its simplicity and minimal overhead have made it popular in IoT and IIoT scenarios. MQTT supports three Quality of Service (QoS) levels, ranging from “at most once” delivery to “exactly once”. The secure variant, MQTT over TLS, operates on port 8883 and is formalized in the OASIS MQTT 5.0 specification [21]. Recent studies highlight that MQTT is among the most widely adopted protocols in the Industrial Internet-of-Things (IIoT), but also one of the most vulnerable: default authentication is weak (username/password in plaintext), TLS adoption is limited due to performance concerns, and brokers often lack scalability and fine-grained access control [22]. These characteristics make MQTT a key enabler for Industry 4.0 while simultaneously representing a major security challenge.

2.1.5. Overall adoption

Recent surveys and large-scale traffic analyses highlight a clear trend: in research datasets and testbeds, Modbus dominates (approximately 39.5%), followed by DNP3 (12.9%), Ethernet/IP (10.5%), and smaller shares of IEC 104 [23] (4.8%), IEC 61850 [24] (4%), Profinet [25] (3.2%), and BACnet [26] (about 2%–3%) [1]. Internet-wide measurements confirm this picture: Modbus, MQTT, and BACnet are among the most frequently exposed protocols, while DNP3 and Profinet appear less often [27]. Table 1 summarizes these findings, comparing adoption levels, transport characteristics, and the availability of secure extensions. Crucially, in almost all cases, the secure variants of these protocols exist but are rarely adopted in practice, as detailed in the next section. The persistence of insecure legacy protocols, combined with their widespread exposure, highlights a clear

¹ <https://github.com/UniboSecurityResearch/P4ICS>.

Table 1

Comparison of industrial communication protocols in terms of transport support, well-known ports, security properties, and adoption. Adoption values are taken from large-scale traffic studies [1,27]. ✓ indicates that a feature is supported, while × denotes its absence.

Protocol	Serial	TCP/IP	Ports	Native security	Adoption (%)	Secure extension
Modbus [13]	✓	✓	502	×	39.5%	TLS [4]
DNP3 [14]	✓	✓	20 000	CRC	12.9%	DNP3-SA [15]
Ethernet/IP [19]	×	✓	44 818/2222	×	10.5%	TLS/DTLS [5]
IEC 104 [23]	×	✓	2404	×	4.8%	IEC 62351 [6]
IEC 61850 [24]	×	✓	102	×	4.0%	IEC 62351 [6]
Profinet [25]	×	✓	34 962–34 964	×	3.2%	Security Classes 1–3 [28]
BACnet [26]	✓	✓	47 808	×	2%–3%	BACnet/SC [29]
MQTT [20]	×	✓	1883	×	–	TLS [21]

misalignment between available standards and real-world implementations. This gap provides the foundation for our problem statement, which stresses that TLS and other cryptographic protections are still largely absent from industrial communications.

2.2. Related work on P4-based security for ICS

Programmable data planes have been increasingly explored to implement cryptographic security mechanisms directly in the network. Hauser et al. proposed P4-MACsec, demonstrating how IEEE 802.1AE link-layer encryption can be realized within a P4-based SDN environment, combining frame protection with dynamic topology monitoring and key management [30]. The same authors extended this approach to the network layer with P4-IPsec, which enabled site-to-site and host-to-site VPN tunnels by implementing IPsec endpoints directly on P4 switches [31]. More recently, they integrated these efforts in P4sec, a framework that automates the deployment of 802.1X access control, MACsec link protection, and IPsec tunneling, under the orchestration of a central controller [32]. Beyond standardized protocols, researchers have also explored the feasibility of executing heavy cryptographic algorithms in the data plane itself. Chen [33] demonstrated an AES implementation on Tofino ASICs by exploiting scrambled lookup tables across multiple pipeline stages, while Oliveira et al. proposed dh-aes-p4, combining in-band Diffie–Hellman key exchange with AES encryption directly between P4-programmable nodes [34]. These works highlight the growing maturity of P4 for in-network cryptography.

Additionally, recent research has expanded the scope of P4 in-network security to advanced detection and inspection tasks. Kecskeméti et al. [35,36] introduced P4RROT, an automated P4 code generator designed to simplify the deployment of hybrid security applications, validated across three distinct use cases: port knocking for authentication, a firewall component for the Network Time Protocol, and passive OS fingerprinting. Similarly, AlSabeh et al. demonstrated the feasibility of performing Deep Packet Inspection on DNS traffic directly in the data plane [37], as well as implementing Machine Learning-based detection for Domain Generation Algorithms on programmable switches [38]. In contrast to these approaches, which primarily focus on in-network security through traffic analysis, P4ICS enforces security by performing in-line encryption and integrity protection for legacy ICS protocols directly in the data plane.

Parallel to these developments, the use of P4 for securing industrial networks has received increasing attention. In the industrial domain, recent research has applied P4 to enhance the protection of ICS communications through in-network cryptography. Hu et al. proposed a framework for industrial network protocol security enhancement using programmable switches, introducing payload parsing and cryptographic mechanisms tailored to protocols such as DNP3 and Modbus [39]. Pittalà et al. investigated how data plane programmability can be leveraged at the edge of industrial networks, combining service orchestration with P4-based cryptographic enforcement for protocol traffic [12]. These contributions demonstrate the feasibility of adopting P4 to provide confidentiality and integrity for ICS protocols, paving the

way for secure channel establishment without relying exclusively on end hosts.

Lastly, Raj et al. proposed a dynamic data-driven security framework tailored for ICS, specifically demonstrating the detection and mitigation of Modbus DoS attacks using P4 programmable switches [40]. P4ICS complements these detection and mitigation strategies by providing a transparent encryption layer that proactively secures the communication channel itself against eavesdropping and replay attacks, ensuring that legacy protocols are protected by default.

Table 2 compares P4ICS with the P4-based security solutions discussed above. Existing proposals span two main directions. On the one hand, frameworks such as P4-MACsec [30], P4-IPsec [31], and P4sec [32] provide general-purpose protection at Layers 2 and 3, securing the communication channel without operating on industrial protocol semantics. On the other hand, the approaches by Hu et al. [39] and Pittalà et al. [12] move closer to ICS-specific processing by targeting industrial traffic directly in the data plane. Within this space, P4ICS occupies a distinct design point: it combines protocol-aware protection for industrial communications with confidentiality, integrity, and replay protection, while also retaining explicit support for monitoring functions through the selective extraction of relevant features before encryption.

3. Problem statement

Traditionally, industrial networks were designed as isolated environments, but digital transformation and Industry 4.0 initiatives have led to increased interconnection with enterprise IT systems and even cloud-based services [41]. As a consequence, protocols that were originally intended to operate only in closed OT networks are now frequently exposed to broader and potentially hostile environments. This paradigm shift makes the adoption of strong security mechanisms an urgent necessity, since communication channels that once relied on physical isolation for security are now accessible to external attackers [42].

Although secure variants of industrial communication protocols exist, their adoption in practice remains marginal. During the last decade, standards bodies and industrial consortia have updated legacy protocols such as Modbus, DNP3, IEC 104, and EtherNet/IP to support TLS, while newer protocols such as MQTT were designed with TLS as the default security option. In theory, this should provide confidentiality, integrity, and authentication for Industry 4.0 communications. In practice, however, multiple large-scale studies demonstrate that these secure variants are almost absent in operational deployments.

Dahlmans et al. [8] conducted an Internet-wide scan of more than 3.7 billion IPv4 addresses and found that only 6.5% of 967,551 identified IIoT deployments used TLS. Even more critically, adoption is uneven: while modern publish–subscribe protocols like MQTT and AMQP reached 7.2% TLS adoption, retrofitted legacy protocols such as Modbus or DNP3 exhibited only 0.4% support. This means that the overwhelming majority of Internet-facing industrial services continue to use unprotected plaintext communication. Furthermore, of the

Table 2

Qualitative comparison of P4ICS with existing P4-based security solutions. The column “Payload visibility” indicates whether the solution explicitly permits access to ICS payload semantics for monitoring or payload-aware anomaly detection.

Paper	Prot. level	ICS- aware	Legacy- transparent	Confidentiality	Integrity	Replay protection	Payload visibility
P4-MACsec [30]	L2	×	✓	✓	✓	✓	×
P4-IPsec [31]	L3	×	×	✓	✓	✓	×
P4sec [32]	L2/L3	×	×	✓	✓	✓	×
Hu et al. [39]	L4	✓	✓	✓	✓	×	×
Pittalà et al. [12]	L4	✓	✓	×	✓	×	×
P4ICS	L4	✓	✓	✓	✓	✓	✓

limited set of TLS-enabled deployments, 42% suffer from misconfigurations that nullify or weaken security, including reliance on deprecated TLS versions, weak cipher suites, expired or self-signed certificates, and even the reuse of compromised cryptographic material across thousands of hosts. Alarming, up to 30% of certificates were found publicly available in software repositories or Docker images, making impersonation trivial.

In a subsequent study [9], the authors provide a detailed analysis of the recurring obstacles that hinder the adoption of TLS in industrial environments. A first limitation is the strong dependence on legacy devices: industrial hardware often remains in operation for 10 to 30 years, and many Programmable Logic Controllers or Remote Terminal Units (RTUs) simply do not have the computational resources or firmware capabilities to support TLS. Even when devices are technically able to use cryptographic functions, the additional performance overhead becomes critical. TLS handshakes and the continuous process of encryption and decryption introduce latency and increase CPU load, factors that are often unacceptable in real-time or safety-critical loops where a deterministic response is essential. This problem is even more evident in constrained embedded IIoT devices, where limited memory and the absence of cryptographic accelerators make the execution of modern TLS libraries inefficient or impractical. In particular, the use of public-key cryptography during TLS handshakes incurs significant bandwidth and processing overheads [43].

Beyond hardware constraints, the complexity of key and certificate management represents another major obstacle. Industrial operators are required to provision, distribute, and periodically renew certificates for thousands of devices. In practice, this often leads to expired certificates, the persistence of default credentials, or, in some cases, the complete deactivation of authentication to avoid operational disruptions. Compatibility issues further exacerbate the situation: to maintain interoperability with legacy clients, some vendors and operators deliberately configure systems to allow outdated TLS versions or weak cipher suites, effectively reducing the security guarantees of the connection. Finally, there is a broader organizational challenge: the high cost of downtime makes operators reluctant to apply updates or enable stricter authentication, leading to operational inertia. Security measures are therefore postponed or disabled in order to avoid any potential interference with production processes, even when this leaves communications exposed.

As a result, despite the availability of secure protocol variants, empirical evidence shows that industrial networks overwhelmingly rely on insecure plaintext communications [1,8,9]. When TLS is used, deployments are frequently misconfigured, with weak cryptography, expired or reused certificates, and missing access controls. This gap between available protocol security and the actual near absence of secure deployments is the key motivation for our work. We aim to design an alternative security framework that can be integrated into industrial networks with minimal overhead, compatibility issues, and management complexity, while still ensuring confidentiality and integrity of ICS communications.

4. Design goals & challenges

The central design goal of our framework is to remove the burden of confidentiality and integrity mechanisms from end devices. Many

industrial assets in the field are either legacy (and cannot be updated to support TLS) or resource-constrained (lacking the CPU, memory, or cryptographic accelerators needed to run modern TLS libraries). Our idea is therefore to shift security functions into the network, leveraging programmable data planes so that packets are transparently processed and protected as they traverse the infrastructure. This approach ensures that communications remain secure even if the end hosts themselves are unaware of encryption, or entirely incapable of providing it.

By exploiting P4, encryption and integrity checks can be performed directly in the forwarding pipeline. This allows industrial traffic to be selectively identified and protected, without requiring changes at the application endpoints. In this way, secure channels are created within the network, while the original devices continue to communicate using the legacy protocol stack. The approach maintains interoperability, reduces operational complexity for plant operators, and makes it possible to extend cryptographic protection also to long-lived equipment that cannot be upgraded.

An important consideration is that this solution deliberately sacrifices end-to-end protection, as offered by TLS, in favor of switch-to-switch protection. In practice, the secure tunnel is established between programmable P4 switches, meaning that traffic is encrypted while traversing the untrusted network segments but remains in clear on the local link between device and switch. Our proposal is valid under a threat model that considers it highly unlikely for an attacker to obtain physical access to the local cabling between end devices and switches, or to the switches themselves. This design choice is formally grounded in the ISA/IEC 62443 [44] standard, specifically in the Zones and Conduits model. In this framework, assets with similar security requirements – such as legacy field devices and PLCs – are grouped into a Security Zone, which is defined as a physically trusted environment protected by access controls. Consequently, the local cabling between the endpoint and the switch is treated as an intra-zone connection, secured by the physical perimeter itself. By contrast, the network segments interconnecting these zones constitute Conduits, which traverse potentially untrusted environments and require logical protection.

P4ICS is intended to secure the Conduits between zones once traffic reaches a trusted first-hop programmable network element. When the endpoint-to-switch segment is physically protected within the same Security Zone, P4ICS provides confidentiality and integrity over the most exposed part of the network path without requiring any modification to legacy devices. As a result, the trust boundary is shifted from the endpoints to the edge of the Security Zone (the first-hop network device), ensuring a pragmatic balance between deployability and security while adhering to industrial defense-in-depth principles.

Conversely, when sensors or legacy devices are deployed in physically exposed positions and the device-to-switch segment cannot be assumed trusted, that segment falls outside the protection boundary of P4ICS. In these cases, P4ICS should be regarded as complementary to first-hop protection mechanisms, such as protected analog cabling, gateway relocation, link- or network-layer protection, or host-based cryptography on capable IIoT nodes.

To exemplify the concept, we refer to a Modbus communication scenario, as illustrated in Fig. 1. In the traditional Modbus/TLS setting (Fig. 1(a)), confidentiality is provided end-to-end by means of a TLS handshake between client and server, which requires that both

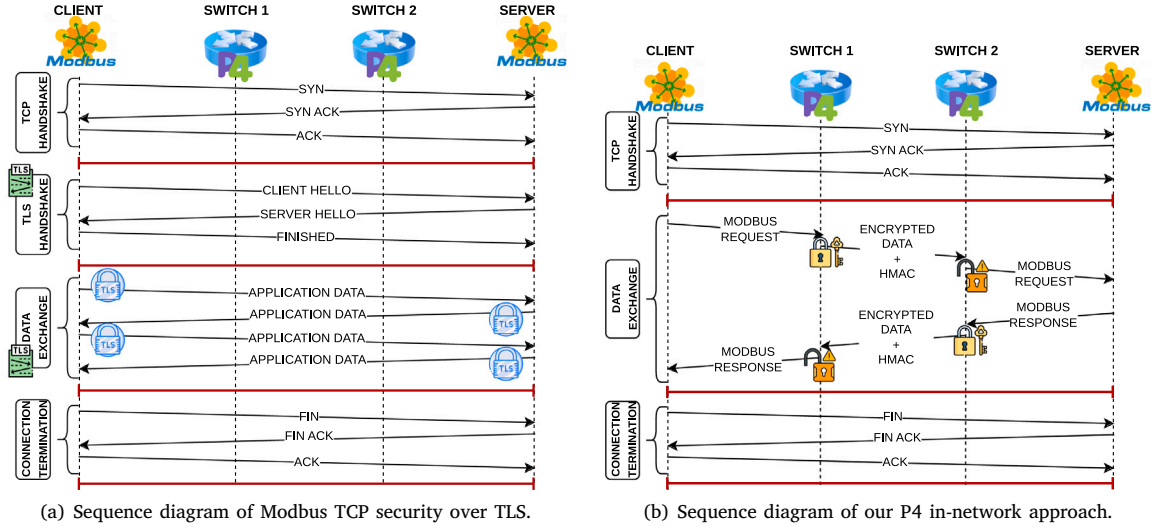


Fig. 1. Comparison of the sequence flow of Modbus security and our proposed approach.

end hosts support the secure variant of the protocol. Conversely, our in-network approach (Fig. 1(b)) assumes that client and server are legacy devices, which communicate in plaintext. P4-enabled switches deployed at the network boundary transparently identify Modbus traffic, establish an encrypted tunnel, and carry the Modbus payload securely across the untrusted network segment. Decryption is then performed before delivering the traffic to the final host, which remains oblivious to the security mechanism. This provides transparent in-network confidentiality and integrity protection for the most exposed parts of the communication path, while overcoming the limitations of legacy equipment.

Designing such a framework raises several challenges. First, parsing and processing industrial protocols in the data plane requires dealing with variable-length fields and application-specific headers. For example, Modbus/TCP messages include the Modbus Application Protocol (MBAP) header, from which payload length must be derived before encryption can be applied. Second, cryptographic operations in the data plane must respect strict performance requirements. Industrial environments impose real-time constraints, meaning that packet processing time added by encryption and decryption must remain in the order of microseconds to avoid disrupting control loops. Third, key management becomes a network-level responsibility: symmetric keys must be provisioned or exchanged between P4 switches at the control plane, and refreshed periodically, without involving the constrained end devices. Finally, the solution must balance security with operational transparency: operators should be able to deploy secure tunnels without modifying PLC firmware, client applications, or field devices.

In summary, our design goals are to provide confidentiality and integrity for industrial communications without modifying end hosts, to ensure performance compatibility with real-time constraints, and to enable flexible, network-level key management. The key challenge is to implement these goals within programmable data planes, proving that in-network security can deliver practical protection for Industry 4.0 environments where traditional TLS-based solutions remain largely unused.

5. System & threat model

We describe our system and threat model using the Purdue Enterprise Reference Architecture [45], which is widely adopted to conceptualize Industrial Control Systems security. As shown in Fig. 2, the architecture organizes the industrial network into six hierarchical levels. Levels 4 and 5 represent the Information Technology (IT)

domain, which is responsible for business applications and enterprise management. Levels 0 to 3 constitute the Operational Technology (OT) domain, where physical processes are monitored and controlled.

At level 0, sensors and actuators interact directly with the physical process. These field devices are connected to level 1, which is composed of Programmable Logic Controllers executing the control logic. PLCs read sensor values and update actuators accordingly, implementing the fundamental closed control loops of the plant. Level 2 includes Human-Machine Interfaces (HMIs) and SCADA servers that collect plant data, display operational dashboards, and enable supervisory control. At level 3, engineering workstations and control servers are used by operators to configure PLCs, deploy control logic, and coordinate plant operations. Above level 3, a demilitarized zone (DMZ) manages the connection between the IT and the OT networks while keeping them isolated from each other [46].

In our system model, programmable P4 switches are strategically deployed at key boundaries of the Purdue architecture (as illustrated in Fig. 2). These devices implement in-network security protection: they parse industrial traffic, identify critical flows, and apply encryption and integrity mechanisms before forwarding the packets. The P4 controller, located at level 3, orchestrates these operations by installing appropriate flow rules and managing symmetric keys installed on the P4 programmable switches.

Regarding the threat model, we assume an adversary with the capabilities of a Dolev-Yao intruder [47]. This means that the attacker can eavesdrop on all communications, inject new messages, replay old ones, or drop and delay packets at will. However, we adopt the perfect cryptography assumption: cryptographic primitives cannot be broken without access to the relevant keys. In line with recent ICS security studies [48–51], we constrain the attacker's reach to levels 2 and 3 of the Purdue architecture. In practice, this models an adversary who has compromised a workstation, SCADA server, network device, or who has gained access to the network interconnecting IT and OT. We do not consider an attacker with direct physical access to field devices and PLCs (levels 0 and 1), as such an adversary could trivially bypass any in-network protection. This limitation is grounded in the ISA/IEC 62443 [44] concept of Zones, which mandates that critical assets are housed within physically secured perimeters (e.g., locked control cabinets), effectively preventing local tampering. Furthermore, given that Level 0 interactions between sensors and actuators often rely on hardwired analog I/O rather than packet-switched protocols, we assume the adversary cannot intercept or inject traffic at this layer without invasive physical wire-tapping. We further clarify that these

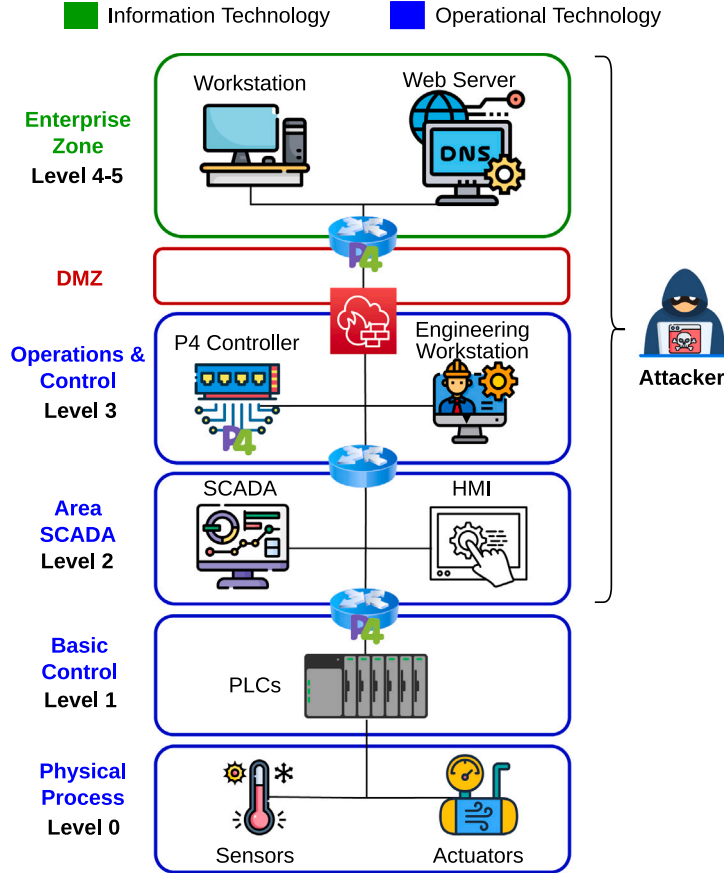


Fig. 2. The threat model of this work is based on the ICS Purdue Reference Architecture. We assume the attacker has Dolev–Yao capabilities above level 1, but cannot tamper with the channel between the P4 controller and the P4 programmable switches.

assumptions do not imply universal physical protection for all Level 0 devices. Rather, whenever P4ICS is applied, we assume that the first-hop P4 programmable switch belongs to the same trusted cell/area as the protected endpoint. If the device-to-switch segment is physically exposed, that last-hop segment is not covered by the guarantees claimed for P4ICS.

Finally, we assume that the communication channel between the P4 controller and the programmable switches is secure [52], and cannot be intercepted or manipulated by the adversary.

6. P4ICS

The proposed framework, named P4ICS, is designed to shift the responsibility for confidentiality and integrity mechanisms away from legacy and resource-constrained end devices and into the network itself. By exploiting programmable data planes, P4ICS provides transparent in-network protection for industrial traffic without requiring any modification to end applications, PLC firmware, or field devices.

P4ICS builds on the P4 programming language, which allows the definition of custom packet processing pipelines in programmable switches. Within this environment, P4ICS supports the four industrial communication protocols that are most commonly used in practice and were analyzed in Section 2, namely Modbus, DNP3, Ethernet/IP, and MQTT. To implement cryptographic functions, the framework leverages the extern² mechanism of P4 as provided by the bmv2³ software switch, which enables the invocation of optimized C++ libraries directly from

the data plane. In our design, this capability is exploited to integrate symmetric AES encryption and decryption [53] routines for payload protection.

In addition to confidentiality, P4ICS ensures integrity and authenticity by computing SHA-256-based HMAC [54] tags over the transmitted payloads. These tags are attached to outgoing packets and verified upon reception, ensuring that any unauthorized tampering is detected within the network. By combining AES-based encryption with SHA256 HMAC verification, the framework ensures that industrial communications are simultaneously protected against eavesdropping, replay, and modification attacks, while maintaining compatibility with legacy devices that lack native TLS support.

6.1. Parsing pipeline

To enable in-network encryption and integrity verification, the parsing process is divided into two stages: a protocol-agnostic stage common to all packets, followed by a protocol-specific stage that extracts the application payload to the granularity required by the cryptographic routines.

A key constraint of the bmv2 target is that any single header or metadata field can be at most 2048 bits (256 bytes). Consequently, application payloads larger than 256 bytes cannot be represented in a single header instance and must instead be parsed iteratively into multiple fixed-size chunks.

In the common stage, the parser extracts the Ethernet, IPv4, and TCP (or UDP) headers. At this point, we define the *application payload length*, denoted as L_{app} , as the number of bytes following the transport header:

$$L_{app} = \text{IPv4.TotalLength} - (4 \cdot \text{IPv4.IHL}) - (4 \cdot \text{TCP.DataOffset}) \quad (1)$$

² <https://p4.org/p4-spec/docs/p4-16-working-draft.html#sec-external-units>.

³ <https://github.com/p4lang/behavioral-model>.

for TCP, and

$$L_{app} = \text{UDP.Length} - 8 \quad (2)$$

for UDP. If $L_{app} \leq 0$, the packet contains no application data and it is transmitted to the ingress pipeline for IPv4 forwarding purposes. Otherwise, the parser inspects the transport port to identify which industrial protocol is being carried.

In the protocol-specific stage, the application payload is extracted in compliance with the 256-byte constraint of the bmv2 target. The process begins from the application payload length L_{app} , computed in the common stage from IPv4 and transport-layer headers. Protocol-specific headers are then parsed to refine this value and determine the exact number of application bytes to be processed. Whenever the resulting payload length does not exceed 256 bytes, the entire data unit can be parsed in a single header instance containing a `varbit` field. Conversely, if the payload length is greater than 256 bytes, the parser engages an iterative extraction strategy based on header stacks. In this design, the payload is segmented into fixed-size slices of 256 bytes, each extracted by invoking the `packet.extract` primitive into a dedicated header instance. After each extraction, a counter tracking the remaining length is decremented, and the parser transitions either to a new extraction state (if at least 256 bytes remain) or to a final state that extracts the last incomplete slice. The latter is implemented as a `varbit` field whose width is determined at runtime to exactly match the remaining number of bytes. This mechanism guarantees that even large application data units are reconstructed deterministically in the data plane.

The four protocols considered in this work each require a dedicated extraction logic. For Modbus/TCP (port 502), the Modbus Application Protocol header carries a *Length* field, which specifies the number of bytes following the field itself, including the Unit Identifier and the Protocol Data Unit (PDU). The Modbus PDU length is therefore derived as $\text{MBAP.Length} - 1$, since the Unit Identifier is not part of the PDU. Since this value never exceeds 253 bytes according to the Modbus specification [13], the entire PDU fits within the 256-byte constraint of bmv2, allowing it to be parsed in a single header instance.

For DNP3 over TCP (port 20000), the encapsulated unit is a *frame* consisting of a 10-byte header and a data section. The header specifies the total frame size, carries link-layer control information, and identifies the source and destination device addresses. The data section, commonly referred to as the payload, contains sensor and actuator values and is protected by CRCs: every 16 data bytes is followed by a 2-byte CRC. The maximum number of data bytes in the payload is 250, not including CRCs, which results in a maximum frame size of 292 bytes when all overheads are counted. From the parser's perspective, this corresponds to approximately 282 bytes of payload plus the 10-byte header. Since this can exceed the 256-byte constraint, the payload is segmented into two slices: the first fixed 256-byte entry in the stack and a final `varbit` slice carrying the residual bytes.

EtherNet/IP (TCP port 44818 or UDP port 2222) presents a more demanding case, as CIP payloads can reach the Ethernet MTU. After parsing the encapsulation and CIP headers, the residual length is computed as $L_{CIP} = L_{app} - L_{hdr}$, where L_{hdr} denotes the combined header size. Because L_{CIP} may approach 1500 bytes, the payload is segmented into 256-byte slices stored in a header stack of depth 5, covering up to 1280 bytes, with the final slice implemented as a `varbit`.

Finally, MQTT (port 1883) can theoretically support message payloads up to 268 MB, as defined by the four-byte *Remaining Length* field in the fixed header. However, in practical IIoT deployments, cloud providers deliberately impose stricter limits to ensure compatibility with resource-constrained devices. For instance, Amazon AWS restricts the maximum payload size to 128 KB [55]. In this work, we adopt this bound as representative of realistic IIoT scenarios. Accordingly, the payload is parsed iteratively in 256-byte slices stored in a dedicated header stack of depth 512, which covers exactly 128 KB.

In summary, the parsing pipeline employs a uniform two-stage strategy. The common stage computes L_{app} and identifies the protocol, while the protocol-specific stage adapts extraction to each protocol under the 256-byte constraint. Modbus fits within a single header instance, DNP3 requires a small header stack of depth 2, and EtherNet/IP and MQTT are parsed through larger stacks of depth 6 and 500, respectively. This design guarantees that the complete application data is made available to the data plane, enabling subsequent AES encryption and SHA256-HMAC integrity verification in the ingress pipeline.

6.2. Ingress pipeline

Once the application payload has been extracted by the parser, the ingress pipeline applies the cryptographic transformations required to guarantee confidentiality and integrity. These operations are implemented directly in the data plane through externs compiled in C++ and invoked inside match-action tables, ensuring that encryption and decryption are performed at line rate without involving the end devices.

The extern interface exposes three primitives:

- `Encrypt(payload, key)` executes AES encryption on the payload. Keys are provided as up to eight 32-bit words, from which the extern infers whether AES-128, AES-192, or AES-256 must be applied, and accordingly sets the number of rounds to $N_r = N_k + 6$.
- `Decrypt(payload, key)` applies the inverse AES transformation using the expanded key schedule, reconstructing the original plaintext when the correct key is supplied.
- `Sha256(payload, key, TCP.seq)` computes a SHA256-based HMAC over the ciphertext concatenated to the TCP sequence number of the packet.

The `Encrypt` and `Decrypt` externs operate on 16-byte blocks. If the payload length is not a multiple of 16, padding is applied so that encryption always proceeds deterministically. Furthermore, because the parser delivers data in 256-byte slices, the extern guarantees alignment at this boundary: whenever the encrypted payload is shorter than 256 bytes, null bytes are appended to reach the fixed slice size. This strategy produces a constant-width ciphertext representation, which simplifies pipeline processing and ensures predictable latency. During decryption, both the AES padding and the slice padding are removed by relying on the *original application length* parsed from the industrial protocol header: this field is never altered by the pipeline, thereby allowing the exact plaintext to be reconstructed.

Additionally, the `Sha256` extern expects as input the ciphertext slice concatenated with the TCP sequence number of the packet. In this way, authenticity is guaranteed and replay attempts are detected: reusing a valid ciphertext under a different sequence number leads to verification failure, which causes the packet to be dropped. By including the TCP sequence number in the HMAC computation, cryptographic integrity is directly tied to the expected transport-layer ordering, providing robustness against both tampering and replay: if a ciphertext is reused or delivered out of order, the verification fails and the packet is discarded.

While the externs provide the cryptographic primitives, their invocation is governed by the actions defined in the ingress pipeline. Each action specifies how the packet is transformed and is triggered by match-action table rules installed by the control plane. The logic of the `encrypt` action – which replaces the plaintext payload with its ciphertext, generates an HMAC tag with Sha256, attaches the tag to the packet as a dedicated header, and updates the length fields of the IPv4 and transport headers – is detailed in Algorithm 1. The complementary `decrypt` action recomputes the HMAC and drops the packet if verification fails; otherwise, it decrypts the ciphertext slice by slice, restores the plaintext payload, removes the HMAC tag from the packet, and updates the corresponding length fields, as summarized in Algorithm 2.

Algorithm 1: Encrypt action in the ingress pipeline.

Input : The payload p of the input packet.
Output: Ciphertext c and HMAC tag h .

```

1  $c \leftarrow \text{Encrypt}(p, k_{aes});$ 
2  $t \leftarrow \text{Sha256}(c, k_{hmac}, TCP.seq);$ 
3 Update IPv4 and transport length fields;
4  $p.setInvalid();$ 
5  $c.setValid();$ 
6  $t.setValid();$ 

```

Algorithm 2: Decrypt action in the ingress pipeline.

Input : Ciphertext c and HMAC tag h .
Output: The decrypted payload p of the input packet if verification succeeds.

```

1  $t' \leftarrow \text{Sha256}(c, k_{hmac}, TCP.seq);$ 
2 if  $t \neq t'$  then
3   mark_to_drop();
4 else
5    $p \leftarrow \text{Decrypt}(c, k_{aes});$ 
6   Update IPv4 and transport length fields;
7    $p.setValid();$ 
8    $c.setInvalid();$ 
9    $t.setInvalid();$ 

```

7. Evaluation

This section evaluates the overall efficacy of P4ICS to determine its applicability in real-world Industry 4.0 scenarios for securing industrial communications under real-time constraints. In particular, we focus on empirically answering the following research questions:

- Q1** How does the deployment of P4ICS affect the round-trip time (RTT) of Modbus, DNP3, EtherNet/IP, and MQTT when using AES keys of 128, 192, and 256 bits, and how do these results compare with both plaintext communication and conventional TLS-secured communication?
- Q2** What is the total resource consumption introduced by deploying P4ICS (in terms of CPU, memory, and power), and how does this overhead impact the runtime performance of industrial communications under realistic operational conditions?
- Q3** How does P4ICS perform compared to state-of-the-art in-network cryptography approaches based on P4 programmable switches?

We begin by outlining the experimental setup used to conduct the tests needed to answer these questions.

7.1. Experimental setup

To evaluate P4ICS, we deployed a physical testbed that mirrors the Purdue Enterprise Reference Architecture, as depicted in Fig. 3. The testbed preserves the canonical separation between the Information Technology and Operational Technology domains, while incorporating devices and systems that are representative of modern Industry 4.0 environments. It provides a realistic setting to assess both the applicability and the performance of P4ICS under practical deployment conditions.

The IT domain is realized through a high-performance workstation located in the Enterprise Zone. The workstation, a Dell PC equipped with an Intel i7 processor and 64 GB of RAM, runs client applications for the industrial protocols under evaluation, namely Modbus/TCP, DNP3, EtherNet/IP, and MQTT, and initiates communication sessions towards the OT network. In particular, the Modbus, EtherNet/IP, and MQTT clients are implemented in Python using the

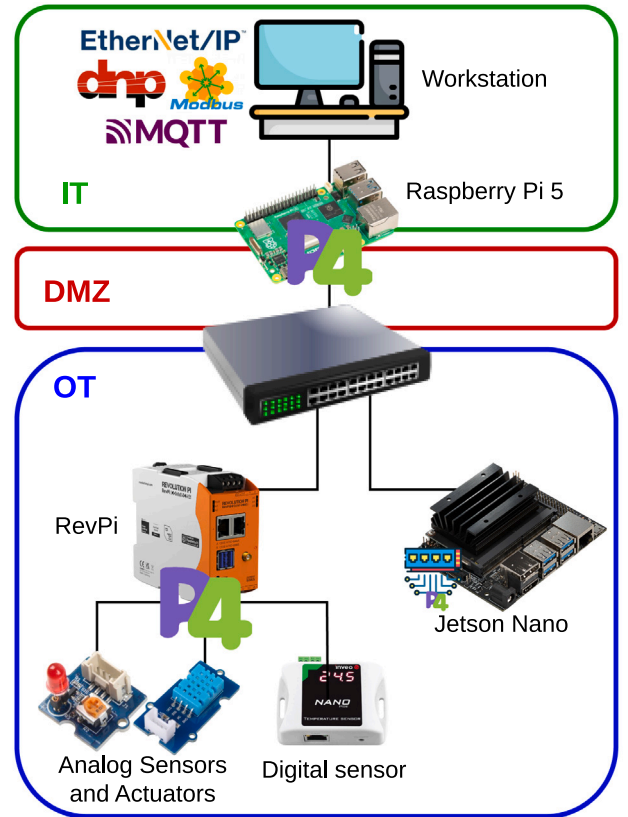


Fig. 3. Experimental setup employed to conduct the evaluation of the P4ICS pipeline, which is deployed on the Raspberry Pi at the IT/OT boundary and on the RevPi at Level 1 of the Purdue model.

pymodbus,⁴ cpppo,⁵ and paho-mqtt⁶ libraries, respectively, while the DNP3 client is implemented in Rust using the commercial DNP3 library by Step Function I/O.⁷

The demilitarized zone is implemented through a Raspberry Pi 5 Model B (4 CPU cores, 8 GB RAM) in combination with an Aruba switch. The Raspberry Pi hosts the Suricata intrusion detection system and a bmv2 software switch, acting as the programmable enforcement point for industrial traffic entering from the IT domain. In this setting, the Raspberry Pi plays a dual role: it inspects traffic at the IT/OT boundary and enforces the P4ICS cryptographic transformations, transparently encrypting and decrypting industrial flows according to the rules pushed by the controller. The Aruba switch mediates the connection towards the OT domain, which in our testbed is organized into three levels, with Level 2 (SCADA) omitted due to the limited scale of the deployment. At Level 0, we deployed analog sensors and actuators,⁸ together with a digital IoT temperature sensor.⁹ These devices generate real field data that feed into the Revolution Pi (RevPi)¹⁰ installed at Level 1. The RevPi acts as a soft-PLC, supporting various industrial communication protocols and hosting a bmv2 software switch to interface with the programmable data plane. Both the Raspberry Pi and the RevPi are used to deploy the P4ICS pipeline described in the previous section, enabling in-network parsing, encryption, and integrity verification of

⁴ <https://www.pymodbus.org/>.

⁵ <https://github.com/pjkundert/cpppo>.

⁶ <https://pypi.org/project/paho-mqtt/>.

⁷ <https://stepfunc.io/products/libraries/dnp3/>.

⁸ <https://www.dexterindustries.com/grovepi/>.

⁹ <https://sklep.inveo.com.pl/en/monitoring/49-nano-temp.html>.

¹⁰ <https://revolutionpi.com/en/revolution-pi-series>.

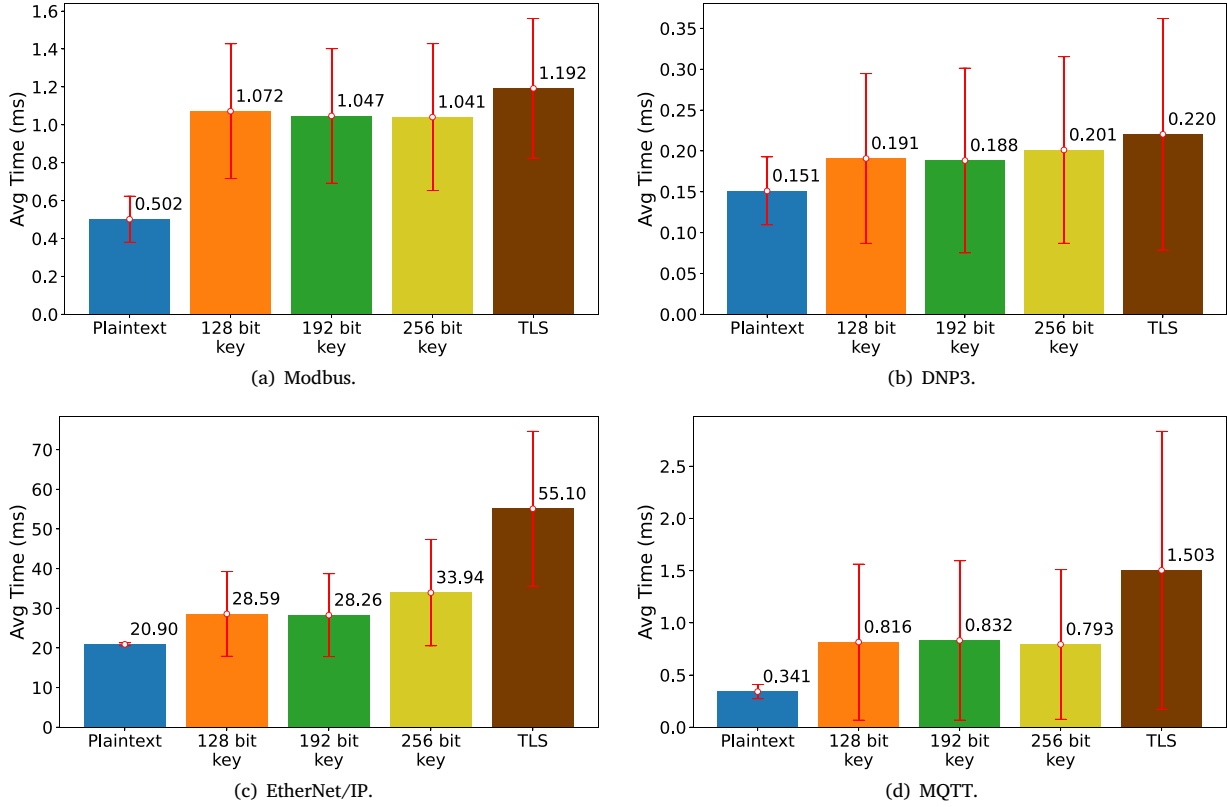


Fig. 4. Comparison of round-trip time (RTT) across industrial protocols in plaintext, TLS-secured, and P4ICS modes with AES-128, AES-192, and AES-256 keys. The reported values are the mean of 100 000 samples, and the error bars report the standard deviation of the measure.

industrial traffic directly at the IT/OT boundary and at Level 1 of the Purdue model.

Finally, the OT domain includes Level 3, where an NVIDIA Jetson Nano is deployed to host the P4 controller. This controller, introduced in our previous work [56,57], manages the programmable switches via P4Runtime, orchestrates key distribution, and activates secure tunnels according to the policies defined at the control plane.

7.2. RTT comparison (Q1)

To address Q1, we measured the RTT of 100,000 read requests from the IT workstation to the digital temperature sensor for each of the four industrial protocols under evaluation. Fig. 4 reports the average values obtained in plaintext, TLS-secured, and P4ICS modes with AES-128, AES-192, and AES-256 keys.

For Modbus, P4ICS introduces an overhead of approximately 107%–113% compared to plaintext communication, but it remains 10%–13% faster than TLS. For DNP3, the additional latency introduced by P4ICS is between 24% and 33% relative to plaintext, while still achieving a 10%–15% reduction compared to TLS. For EtherNet/IP, P4ICS increases RTT by 35%–62% over plaintext, yet it reduces latency by almost half with respect to TLS (38%–48% faster). For MQTT, P4ICS results in a 133%–144% overhead relative to plaintext, but still consistently outperforms TLS with a 46%–48% improvement.

Overall, P4ICS systematically outperforms TLS across all protocols and key sizes, while introducing an expected increase in latency compared to plaintext communication due to the added cryptographic operations. These findings confirm that P4ICS can offer a practical trade-off between security and performance in real-time industrial settings.

It is important to emphasize that these results should not be interpreted as a performance comparison across different protocols, but only

within each protocol. The client implementations are heterogeneous, relying on distinct libraries and languages, which strongly influence their baseline performance. For instance, the DNP3 client, implemented using a commercial industrial-grade library in Rust, exhibits significantly lower RTTs than the other protocols, reflecting differences in software stacks rather than protocol characteristics.

7.3. Overall resource consumption (Q2)

To answer Q2, we measured the total resource consumption introduced by deploying P4ICS on the two programmable switches of the testbed: the Revolution Pi and the Raspberry Pi 5. The measurements were carried out during the same experimental campaign of 100,000 read requests used for RTT evaluation, and cover three dimensions: CPU utilization, memory footprint, and power consumption. For each protocol and AES key size, Table 3 reports both the mean and maximum values observed.

The data show that CPU utilization remains modest across all protocols and key sizes. On average, P4ICS requires between 25% and 35% of CPU resources, with occasional peaks slightly above 50%. Such figures are well within the capabilities of embedded industrial devices, especially considering that both platforms are constrained, low-power hardware. The memory footprint is essentially negligible: mean values are consistently in the range of 55–127 MB depending on the protocol, representing less than 2% of the 8 GB RAM available on both the RevPi and the Raspberry Pi 5. Power consumption is practically constant across all experiments, with average values around 2.7–2.8 W, showing that encryption and integrity verification do not introduce significant energy overhead.

Overall, these results indicate that the deployment of P4ICS imposes only a limited resource overhead, even when executed on software switches running on resource-constrained platforms such as Raspberry Pi devices.

Table 3

Resource usage of P4ICS across protocols, AES key sizes, and programmable switches. Values report the mean and maximum CPU, memory, and power consumption measured on the Revolution Pi and Raspberry Pi 5, both running bmv2 as software switches. Results are averaged over 100,000 requests per protocol.

Protocol	Key size	CPU (%)				Memory (MB)				Power (W)			
		RevPi		Raspberry Pi 5		RevPi		Raspberry Pi 5		RevPi		Raspberry Pi 5	
		Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max
DNP3	128	25.10	36.68	27.10	36.68	115.47	116.37	116.00	116.90	2.73	2.77	2.81	2.81
	192	32.85	51.68	31.68	54.70	116.63	116.90	110.65	116.72	2.77	2.77	2.81	2.81
	256	33.01	38.33	33.13	53.13	115.98	116.40	112.50	117.12	2.77	2.77	2.81	2.81
EtherNet/IP	128	22.17	26.68	24.91	30.00	126.19	126.93	127.38	127.60	2.77	2.77	2.65	2.81
	192	22.80	26.68	24.91	30.00	125.78	126.34	127.44	127.68	2.77	2.77	2.68	2.81
	256	21.80	28.33	24.77	28.33	126.91	127.40	127.42	127.78	2.77	2.77	2.64	2.81
Modbus	128	29.25	38.33	35.09	53.33	57.28	60.35	116.41	117.58	2.76	2.77	2.81	2.81
	192	28.40	35.00	38.53	51.68	54.06	54.18	116.28	117.59	2.76	2.77	2.81	2.81
	256	30.13	38.33	34.03	58.33	57.48	57.89	115.63	117.60	2.77	2.77	2.81	2.81
MQTT	128	36.05	41.68	34.25	53.33	117.49	118.00	115.54	117.22	2.77	2.77	2.81	2.81
	192	35.09	40.00	34.42	56.25	117.86	118.37	116.07	117.34	2.77	2.77	2.81	2.81
	256	35.52	41.68	33.96	55.00	116.49	117.35	116.09	117.08	2.77	2.77	2.81	2.81

Table 4

Performance comparison of P4ICS with respect to [33] on a Revolution Pi running bmv2, in terms of packet processing time, packet dequeuing time, CPU utilization, memory usage, and power consumption.

Method	PPT (ms)		PDT (ms)		CPU (%)		Memory (MB)		Power (W)	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max
Chen et al. [33]	2188.56	438652.86	0.00065	0.627	25.09	35.00	202.74	216.45	2.76	2.76
P4ICS	0.90	69.25	0.0025	3.96	6.05	13.35	48.95	48.95	2.76	2.76

7.4. Comparison with the state-of-the-art (Q3)

The state-of-the-art solution for in-switch AES encryption is the Scrambled Lookup Tables (LUTs) approach proposed by Chen et al. [33]. This method implements AES encryption entirely in the P4 data plane by precomputing and storing parts of the AES transformation in lookup tables. The authors demonstrated a working implementation on both a hardware Tofino switch and the bmv2 software switch. On Tofino, their P4 program achieved up to 10.92 Gbit/s throughput for AES-128 encryption, while using less than 15% of switch memory. The design is limited to a 128-bit key and operates on a single 128-bit (16-byte) data block per packet. Due to Tofino's pipeline stage limitations, the switch must recirculate each packet through the pipeline multiple times: the prototype completes AES-128 (10 rounds) in five passes for a single 16-byte payload (six and seven passes are needed for AES-192 and AES-256, respectively). This LUT-based approach leverages abundant hardware memory to replace expensive arithmetic operations with table lookups, making in-network AES feasible on ASICs.

To compare our solution with Chen et al. approach, we deployed their LUT-based AES pipeline on a Revolution Pi running bmv2, and evaluated both solutions under identical conditions. In order to integrate the comparison within our industrial testbed, we extended their implementation to include a Modbus/TCP parser, thereby enabling the LUT-based design to process industrial traffic in the same manner as our extern-based solution. We transmitted 1000 Modbus packets using Scapy,¹¹ each carrying a single 16-byte payload (the official AES-128 test vector from NIST FIPS-197 [53]) encrypted with a 128-bit key. We measured the packet processing time (PPT) inside the P4 pipeline, any packet dequeuing time (PDT) before processing, as well as CPU utilization, memory usage, and power consumption on the device. The results are summarized in Table 4, reporting both mean and maximum values for each metric.

The results show that P4ICS dramatically outperforms the scrambled-LUT design by Chen et al. [33] when executed on bmv2 software switches. The average packet processing time of the LUT-based implementation is 2188.56 ms, compared to only 0.90 ms for P4ICS,

corresponding to a speedup of more than three orders of magnitude ($\sim 2400\times$). Worst-case latency is also significantly higher in the LUT-based solution (438653 ms vs. 69 ms for P4ICS). In terms of resource usage, Chen's approach consumes about four times more CPU (25% vs. 6%) and memory (203 MB vs. 49 MB), while power consumption remains practically identical at 2.76 W.

Both approaches correctly produced the expected NIST test vector output, confirming functional correctness. However, the LUT-based method is rigid: it supports only 128-bit keys and single 16-byte payloads per packet. By contrast, P4ICS supports AES-128/192/256 and arbitrary payload sizes segmented into 256-byte slices.

In summary, while the scrambled-LUT approach is effective on Tofino ASICs, P4ICS is vastly more efficient and flexible on software and embedded targets, reducing CPU load by 75%, memory usage by 76%, and latency by several orders of magnitude. These properties make P4ICS more suitable for realistic industrial deployments with variable payloads and constrained hardware.

8. Modbus monitoring case study

The limited adoption of TLS in industrial networks is primarily motivated by operational requirements rather than technical unavailability. The first obstacle is the necessity of maintaining full visibility over process data. Predictive maintenance frameworks and anomaly detection systems rely on continuous access to raw sensor values and control variables to infer plant conditions and detect incipient failures [1,39]. If traffic were encrypted end-to-end, these values would be hidden, forcing monitoring to either terminate encryption at intermediate nodes – introducing additional attack surfaces – or to move all analytics to constrained endpoints, which is rarely feasible in ICS deployments. These factors explain why confidentiality is frequently sacrificed in favor of visibility and real-time performance. However, this choice leaves process communications vulnerable to eavesdropping and manipulation. Our approach directly addresses this trade-off by exploiting P4 programmability and SDN orchestration to selectively extract relevant process variables before encryption, while preserving encrypted end-to-end delivery towards workstations. To demonstrate its feasibility, we conducted a case study based on the Tennessee Eastman Process (TEP), a widely recognized standard benchmark for

¹¹ <https://scapy.net/>.

assessing process monitoring and fault diagnosis methods. It consists of five main modules, each a two-stage reactor, a condenser, a vapor-liquid separator, a stripper, and a reboiler, as well as 11 pneumatic valves, two pumps, and a compressor. The process is described by 53 continuous variables, including 41 measured outputs (e.g., flows, pressures, compositions, and temperatures) and 12 manipulated variables corresponding to control inputs.

In our experimental setup, the TEP simulator [58] is connected in hardware-in-the-loop to the testbed presented in Section 7.1. The RevPi acts as a programmable logic controller and Modbus/TCP master. The RevPi collects process variables and transmits them via Modbus to the IT workstation. It also applies in-network payload encryption and authentication in the data plane, as described in Section 6, employing 256-bit keys. At the same ingress stage, before encryption, the RevPi collects the process variables and, via `packet_in` message, sends them to the P4 controller, which is deployed on the Jetson Nano. This P4 controller is responsible for control-plane operations, such as policy management and rule installation, and additionally acts as the anomaly detection engine by running machine learning models on the TEP features.

For anomaly detection, we trained five Long Short-Term Memory (LSTM) models on the TEP dataset introduced in [59], with history lengths $N = \{1, 5, 10, 25, 50\}$. Each model consists of two stacked LSTM layers with 256 hidden units each, followed by a dense output layer. Training was performed with a batch size of 512 for up to 100 epochs with early stopping based on validation loss. The reconstruction error distribution was modeled during training, and anomaly thresholds were selected using a chi-square (χ^2) criterion on the residuals [60].

Since each LSTM model with history length N requires N consecutive samples of the 53 process variables to perform detection, the monitoring traffic must be organized accordingly. Each monitoring message sent from the RevPi to the P4 controller therefore aggregates N samples (212 bytes per sample) together with a fixed 58-byte Ethernet/IP/TCP header. The resulting control channel utilization (CCU) can be expressed as

$$CCU = \frac{\text{PacketSize}(N)}{\Delta t(N)} \text{ [B/s]},$$

where $\text{PacketSize}(N) = 58 + 212 \cdot N$ and $\Delta t(N) = 1.8 \cdot N$ s is the inter-packet interval imposed by the TEP simulator.

Detection latency reflects the time needed to collect the N samples and process them along the path. We define it as

$$L(N) = 1.8 N + \text{PPT} \cdot N + \frac{\text{RTT}}{2} + t_{\text{LSTM}} \cdot N,$$

where PPT = 0.3 ms denotes per-packet processing time, RTT = 60 ms the round-trip delay, and t_{LSTM} the per-sample inference time. In Table 5 we report latencies excluding inference time in order to isolate network and processing overhead.

To evaluate the trade-off between communication overhead and detection performance, we computed CCU, latency, and four standard metrics – accuracy, precision, recall, and F1-score – across the five history lengths. Results are summarized in Table 5.

The results show that as the history length N increases, control-channel utilization (CCU) decreases slightly, from about 150 B/s at $N = 1$ to 118 B/s at $N = 50$. This effect is due to the amortization of the fixed protocol header across a larger number of samples. The gain in communication efficiency, however, is offset by a substantial increase in detection latency, which grows almost linearly with N , ranging from approximately 1.8 s at $N = 1$ to about 90 s at $N = 50$. The dominant factor in this increase is the time required to accumulate N samples, while networking and processing delays contribute only marginally.

Detection performance metrics remain stable in terms of overall accuracy, which stays around 0.95 regardless of N . Recall improves significantly with larger histories, reaching very high values at $N = 10$ and above, confirming that longer temporal windows help the model to

Table 5

Anomaly detection results for the Modbus monitoring case study on the Tennessee Eastman Process (TEP). Each entry corresponds to an LSTM model with history length N , where monitoring messages aggregate N process samples before encryption. Reported metrics include control channel utilization, detection latency, and classification performance (accuracy, precision, recall, F1-score).

History	CCU (B/s)	Latency (s)	Accuracy	Precision	Recall	F1-score
1	150.0	1.83	0.953	0.467	0.922	0.620
5	124.2	9.03	0.951	0.454	0.868	0.596
10	121.0	18.03	0.951	0.458	0.988	0.626
25	119.1	45.04	0.951	0.460	0.960	0.622
50	118.4	90.05	0.952	0.466	0.988	0.633

capture a broader range of anomalies and reduce false negatives. Precision, on the other hand, remains consistently low (around 0.45–0.47), indicating that a considerable fraction of alarms are false positives. As a result, the F1-score shows only moderate improvements when recall increases.

These findings highlight a fundamental trade-off. Short histories yield low latency and faster reaction times, which may be preferable in scenarios where timely detection is critical. Longer histories reduce communication overhead and improve recall, but at the cost of substantially higher detection delays. The consistently low precision is nevertheless in line with what can be expected from LSTM-based reconstruction models when the anomaly threshold is determined in a fully unsupervised manner, i.e., without tuning on any portion of the attack dataset. As highlighted in [61], this choice favors generality and avoids the need for labeled anomalies, but typically leads to higher false positive rates compared to supervised or semi-supervised threshold calibration.

9. Discussion & limitations

9.1. Key management and tunnel activation

The establishment of secure tunnels between P4-enabled switches is governed by the control plane through P4Runtime [62], which populates the match-action tables that decide how industrial traffic is processed. Two complementary activation strategies are possible:

1. Static activation: an encrypted tunnel is instantiated at startup and persistently covers all traffic of a given industrial protocol. This minimizes control-plane interactions and guarantees continuous protection, although at the cost of encrypting also traffic that may not require confidentiality.
2. On-demand activation: the controller installs flow rules dynamically, triggering the creation of a secure tunnel only when traffic between specific endpoints is detected. This approach avoids unnecessary cryptographic overhead while still protecting sensitive flows as soon as they become active.

In both cases, the controller determines which flows are directed to the encrypt and decrypt actions and which instead follow the default forwarding path. The selection is entirely transparent to the end devices: legacy industrial nodes continue to exchange plaintext according to their native protocol specifications, while the switches automatically encrypt packets before they leave the trusted domain and decrypt them upon re-entry. In this way, confidentiality, integrity, and replay protection are enforced selectively in-network without requiring any modification of industrial endpoints.

The control plane is not only responsible for activating secure tunnels, but also for the distribution of the symmetric keys that protect them. Keys may be pre-installed at configuration time for static deployments, or dynamically negotiated through a Diffie-Hellman (DH) key exchange protocol [63]. In the latter case, the control plane coordinates

the handshake, derives the session key, and installs it in dedicated registers of the P4-enabled switches via the P4Runtime API. The security of these control-plane operations relies on the integrity of the communication channel between the P4 controller and the switches. To mitigate the risk of key interception or unauthorized rule injection, P4ICS leverages the security features of the P4Runtime protocol, which mandates the use of gRPC over Mutual TLS (mTLS) [62]. This ensures that both the controller and the network devices are mutually authenticated and that the symmetric keys exchanged for tunnel establishment remain encrypted in transit. The registers in which keys are installed are subsequently accessed by the ingress pipeline during encryption and decryption operations. Keys are bound to the flow-table entries that activate the `encrypt` and `decrypt` actions, ensuring that identical cryptographic material is consistently applied across the entire tunnel. To limit the exposure window in case of compromise, the controller can also enforce periodic rekeying, transparently refreshing the registers with new symmetric keys without interrupting ongoing communications.

Finally, to quantify the scalability impact of the on-demand tunnel activation in scenarios involving multiple controllers, we conducted a targeted test using Modbus as a reference protocol. Specifically, we measured the time required to negotiate and install keys between two switches managed by two distinct controllers. Fig. 5 reports the average latency computed over 10 000 connection attempts for each of the following three scenarios:

1. Modbus Connection: the baseline time necessary to complete a standard TCP three-way handshake.
2. Modbus TLS Connection: the time required to establish a secure session using the Modbus/TCP Security protocol, comprising both the TCP and TLS handshakes.
3. P4ICS Connection with DH: the total time to establish a P4ICS secure tunnel dynamically. This encompasses (1) the Diffie-Hellman (DH) key exchange between two distinct controllers, (2) the installation of the derived symmetric keys into the P4 switch tables via P4Runtime, (3) the installation of the tunnel activation rules for the two selected endpoints, and (4) the final TCP handshake time between endpoints.

As shown in Fig. 5, the true overhead introduced by the secure tunnel activation is represented by the difference between the P4ICS Connection with DH and the baseline Modbus Connection. This delta averages to 4.549 ms and corresponds exclusively to control-plane operations (key exchange and rule installation). Crucially, this overhead is independent of the specific industrial protocol being secured and remains within the same order of magnitude of milliseconds as a standard TLS handshake. Consequently, this value can be considered the intrinsic computational cost for securing any flow between two endpoints, effectively representing the system's baseline orchestration overhead. Net of variations strictly attributable to network propagation delays and geographical distances, this result confirms that the proposed P4ICS framework scales efficiently even in multi-controller environments, as the impact on overall network performance is comparable to standard TLS overhead.

9.2. Security analysis

As discussed in Section 5, our threat model assumes a Dolev-Yao adversary [47] with full control over the network at Purdue Levels 2 and 3: the attacker can intercept, replay, drop, and inject arbitrary packets, but cannot break standard cryptographic primitives under the perfect cryptography assumption. The channel between the P4ICS controller and the programmable switches is assumed secure. Within this model, we analyze the guarantees provided by P4ICS.

First, confidentiality is enforced by in-network AES encryption, which ensures that industrial payloads traversing untrusted network

segments cannot be understood by an eavesdropper. Even if the attacker captures complete flows, the payload remains indistinguishable from random noise without the symmetric key.

Second, integrity and authenticity are guaranteed through SHA-256-based HMAC tags, which bind each ciphertext to its TCP sequence number. Any modification of the payload or reordering of packets results in verification failure at the receiving switch, causing the packet to be dropped. Forged traffic is likewise discarded since the adversary lacks the secret keys.

Third, replay protection is achieved by incorporating transport-layer sequence numbers into the HMAC computation. This design ensures freshness of messages: if an attacker records a valid packet and later attempts to replay it, the network will recognize the duplicate or out-of-order sequence and discard the packet. In practice, this thwarts replay attacks, wherein an adversary could otherwise resend legitimate commands (e.g., opening a valve or tripping a breaker) to cause unauthorized actions at improper times.

By enforcing these measures directly in the data plane, P4ICS significantly reduces the adversary's room for maneuver. An attacker operating at Levels 2/3 of the Purdue model can no longer silently read or subtly manipulate industrial commands as in legacy plaintext deployments. Tampering attempts are detected through HMAC verification, replayed messages are discarded, and encrypted payloads remain unintelligible. As a result, the adversary is largely confined to disruptive actions such as Denial-of-Service (DoS), while stealthy man-in-the-middle attacks are effectively mitigated.

However, it is important to delineate the scope of P4ICS's protection. The security guarantees are provided switch-to-switch, not truly end-to-end at the application layer. In other words, P4ICS creates secure tunnels between network devices (P4-programmable switches) in the industrial network. Traffic is decrypted once it leaves the final P4-enabled switch, so a compromise of endpoints or local subnets may still expose plaintext communications. This limitation reflects the design choice of preserving interoperability with legacy devices that cannot support TLS or other secure protocol variants. Compared to full end-to-end encryption, which would provide stronger guarantees but require upgrading all devices [8], P4ICS represents a pragmatic compromise: it secures the most exposed network segments against attackers, while remaining deployable in legacy ICS environments.

Nevertheless, the proposed architecture is flexible enough to accommodate stronger security models for modern industrial assets. Specifically, for IIoT devices running Linux-based operating systems capable of executing Extended Berkeley Packet Filter (eBPF) programs with sufficient computational headroom [64], we propose extending the secure boundary directly to the network edge. In this hybrid deployment, the P4 switch is configured via control-plane policies to bypass the decryption stage for these capable endpoints selectively. Instead, the encrypted payload and HMAC tag are forwarded directly to the host, where an eBPF program hooked at the eXpress Data Path (XDP) or Traffic Control (TC) layer performs transparent decryption and verification in the kernel space [65] before passing the data to the userspace application. Crucially, this approach preserves complete interoperability with legacy equipment. The architecture supports a mixed environment where modern devices utilize end-to-end encryption (terminating at the host kernel), while legacy hosts continue to rely on the switch-to-switch tunnel. Consequently, traffic is exposed in plaintext only on the specific local links connecting legacy assets that strictly lack the resources for cryptographic operations, while communication involving modern IIoT nodes remains protected across the entire path.

9.3. Portability limitation

While P4ICS demonstrates the feasibility of in-network security for industrial protocols, some limitations emerge in terms of portability. The most significant challenge lies in the use of P4 externs for AES and HMAC operations. These externs are supported by the BMv2 software

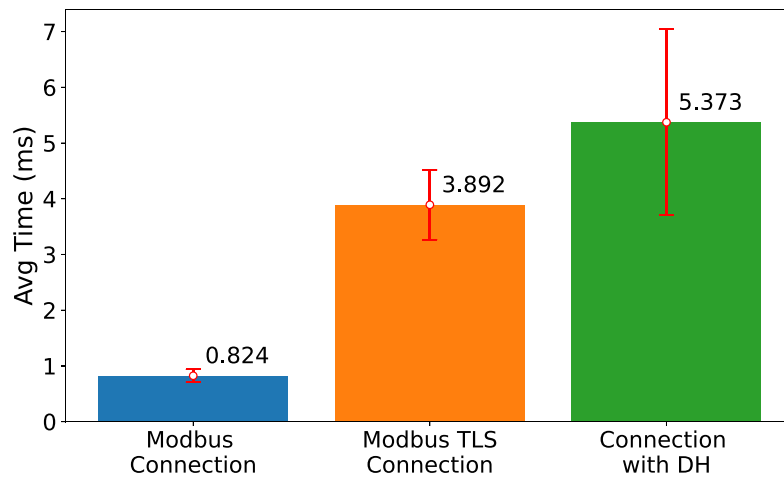


Fig. 5. Comparison of connection establishment times. The overhead of P4ICS key exchange is the difference between the P4ICS setup time and the baseline Modbus TCP connection time.

switch, but not by hardware targets such as Intel/Barefoot Tofino, whose pipeline does not allow arbitrary cryptographic primitives [66]. As a result, the current design cannot be directly ported to production ASICs. However, alternative approaches utilizing Field-Programmable Gate Arrays (FPGAs) externs [67] offer a concrete path to overcome these hardware constraints. Unlike fixed-function ASICs, FPGA targets support the integration of custom externs defined in Hardware Description Languages (HDL). Frameworks such as P4FPGA [68] and the P4→NetFPGA workflow [69] allow developers to map P4 programs onto FPGA hardware while offloading computationally intensive tasks to specialized hardware modules. In the context of P4ICS, the C++ cryptographic externs used in our prototype could be replaced by optimized HDL implementations. The feasibility of this approach is supported by recent work demonstrating cryptographic hashing in P4 data planes [70] and the effective offloading of complex security protocols on FPGA-capable SmartNICs [71]. These solutions indicate that P4ICS can be transitioned to hardware targets by leveraging the flexibility of FPGAs HDL externs to enforce encryption and integrity directly on the data plane itself.

A second limitation concerns the handling of large or variable-length payloads, particularly in MQTT. Unlike Modbus or DNP3, which have small and bounded payloads, MQTT supports messages up to 256 MB [20]. Even with practical limits such as the 128 KB cap imposed by AWS IoT Core [55], processing these payloads in a P4 pipeline remains challenging. Current P4 hardware is optimized for fixed-length headers and short packets, and struggles with deeply nested or very large structures [39]. In our BMv2 prototype, large MQTT payloads were managed by slicing them into 256-byte chunks, but this approach already revealed performance and memory bottlenecks. While a software switch can technically handle packets of hundreds of kilobytes, this is not sustainable in constrained environments. On hardware ASICs the problem is even more severe: parsing application data on the order of 128 KB would require fragmentation, reassembly, and stateful processing beyond the capabilities of the data plane. As a result, P4ICS can handle MQTT traffic within bounded message sizes in controlled settings, but its portability to hardware targets remains limited, especially for multi-megabyte or streaming payloads.

10. Conclusion

This paper introduced P4ICS, an in-network security framework that shifts confidentiality and integrity enforcement from legacy and resource-constrained ICS endpoints into programmable data planes. By exploiting P4, our design enables transparent encryption and authentication of Modbus, DNP3, EtherNet/IP, and MQTT traffic, without

requiring modifications to end devices or industrial applications. In doing so, P4ICS provides transparent in-network confidentiality and integrity protection across untrusted network segments, while preserving interoperability with the heterogeneous and long-lived equipment typical of industrial environments.

Our evaluation on a realistic testbed demonstrated that P4ICS can be deployed on resource-constrained embedded platforms such as Raspberry Pi and Revolution Pi, achieving practical round-trip latencies and modest CPU/memory overheads. Specifically, P4ICS consistently outperforms traditional TLS, reducing communication delays by approximately 12% for Modbus and DNP3, and up to 48% for MQTT and EtherNet/IP. Furthermore, our comparative study against state-of-the-art scrambled-LUT AES designs confirmed that the extern-based approach of P4ICS is orders of magnitude more efficient on software targets, reducing packet processing time by a factor of $\sim 2400\times$ and memory usage by 76%. Finally, the Modbus case study highlighted how P4ICS can reconcile confidentiality with process visibility, enabling selective pre-encryption monitoring that supports anomaly detection frameworks without sacrificing secure delivery.

Some limitations remain, particularly regarding the portability of P4ICS to hardware targets that do not support custom externs and the handling of large or variable-length payloads such as those of MQTT. Future work will therefore focus on extending the design towards hardware accelerators and FPGA-supported externs, as well as optimizing the processing of large payloads to ensure scalability and deployability in production environments.

Overall, P4ICS demonstrates that in-network cryptography can offer a deployable and efficient alternative to end-to-end TLS in industrial environments, narrowing the gap between secure protocol specifications and their limited adoption in practice. By leveraging P4 programmability, our framework provides a foundation for secure, interoperable, and performance-oriented industrial networking in Industry 4.0.

CRediT authorship contribution statement

Lorenzo Rinieri: Writing – review & editing, Writing – original draft, Validation, Software, Formal analysis, Data curation. **Andrea Melis:** Writing – review & editing, Writing – original draft, Validation, Investigation, Conceptualization. **Roberto Girau:** Writing – review & editing, Supervision, Project administration. **Giovanni Pau:** Writing – review & editing, Project administration, Funding acquisition. **Marco Prandini:** Writing – review & editing, Supervision, Project administration. **Franco Callegati:** Writing – review & editing, Supervision, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. This study was also partially funded within the MOST – Sustainable Mobility National Research Center and received funding from the European Union Next-GenerationEU (PNRR – Missione 4 Componente 2, Investimento 1.4 – D.D. 1033 17/06/2022, CN00000023). This manuscript reflects only the authors' views and opinions; neither the European Union nor the European Commission can be considered responsible for them.

Data availability

No data was used for the research described in the article.

References

- [1] M. Conti, D. Donadel, F. Turrin, A survey on industrial control system testbeds and datasets for security research, *IEEE Commun. Surv. Tutor.* 23 (4) (2021) 2248–2294.
- [2] E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.3, RFC 8446, 2018, <http://dx.doi.org/10.17487/RFC8446>, URL <https://www.rfc-editor.org/info/rfc8446>.
- [3] E. Rescorla, N. Modadugu, Datagram Transport Layer Security Version 1.2, RFC 6347, 2012, <http://dx.doi.org/10.17487/RFC6347>, URL <https://www.rfc-editor.org/info/rfc6347>.
- [4] Modbus Organization, Modbus/TCP security protocol specification, 2021, URL <https://www.modbus.org/file/secure/modbussecurityprotocol.pdf>.
- [5] ODVA, CIP security specification, 2015, URL <https://www.odva.org/technology-standards/distinct-cip-services/cip-security/>.
- [6] IEC, IEC 62351: Power systems management and associated information exchange – data and communications security, 2007, URL <https://webstore.iec.ch/en/publication/63742>.
- [7] M. Dahlmans, J. Lohmöller, I.B. Fink, J. Pennekamp, K. Wehrle, M. Henze, Easing the conscience with OPC UA: An internet-wide study on insecure deployments, in: *Proceedings of the ACM Internet Measurement Conference*, ACM, 2020, pp. 101–110.
- [8] M. Dahlmans, J. Lohmöller, J. Pennekamp, J. Bodenhausen, K. Wehrle, M. Henze, Missed opportunities: measuring the untapped TLS support in the industrial internet of things, in: *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, 2022, pp. 252–266.
- [9] M. Dahlmans, K. Wehrle, Protocol security in the industrial internet of things, in: *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, IEEE, 2024, pp. 1–4.
- [10] I. Kunze, K. Wehrle, D. Trossen, M.-J. Montpetit, X. de Foy, D. Griffin, M. Rio, Use Cases for In-Network Computing, RFC 9817, 2025, <http://dx.doi.org/10.17487/RFC9817>, URL <https://www.rfc-editor.org/info/rfc9817>.
- [11] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al., P4: Programming protocol-independent packet processors, *ACM SIGCOMM Comput. Commun. Rev.* 44 (3) (2014) 87–95.
- [12] G.F. Pittalà, L. Rinieri, A. Al Sadi, G. Davoli, A. Melis, M. Prandini, W. Cerroni, Leveraging data plane programmability to enhance service orchestration at the edge: A focus on industrial security, *Computer Networks* 246 (2024) 110397, <http://dx.doi.org/10.1016/j.comnet.2024.110397>.
- [13] Modicon, Modbus protocol specification, 1979, URL https://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf.
- [14] IEEE, IEEE standard for electric power systems communications – distributed network protocol (DNP3), 2010, URL <https://standards.ieee.org/ieee/1815/4894/>.
- [15] G. Harris, Overview of DNP3 security version 6, 2020, URL https://scadahacker.com/library/Documents/ICS_Protocols/DNP3%20Secure%20Authentication%20v5%202011-11-08.pdf.
- [16] J.A. Crain, S. Bratus, Bolt-on security extensions for industrial control system protocols: A case study of dnp3 sav5, *IEEE Secur. Priv.* 13 (3) (2015) 74–79.
- [17] C. Rosborough, G. Gordon, B. Waldron, All about eve: comparing DNP3 secure authentication with standard security technologies for SCADA communications, in: *13th Australasian Information Security Conference*, Vol. 161, 2019.
- [18] R. Landheer-Cieslak, Authentication of individual users in DNP3 secure authentication- TB2019-001, and more, 2019, URL <https://rlc.vlinder.ca/blog/2019/10/26/tb2019-001-multi-user-sa>. (Online Accessed 18 September 2025).
- [19] ODVA, Ethernet/IP specification: CIP on ethernet technology, 2000, URL https://www.odva.org/wp-content/uploads/2024/04/PUB00138R8_Ethernet.pdf.
- [20] IBM, MQTT version 3.1 specification, 1999, URL <https://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html>.
- [21] OASIS, OASIS standard: MQTT version 5.0, 2019, URL <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>.
- [22] B. Mishra, A. Kertesz, The use of MQTT in M2M and IoT systems: A survey, *IEEE Access* 8 (2020) 201071–201086.
- [23] IEC, IEC 60870-5-104: Telecontrol equipment and systems – transmission protocols – network access for IEC 60870-5-101 using standard transport profiles, 2000, URL <https://webstore.iec.ch/en/publication/3746>.
- [24] IEC, IEC 61850: Communication networks and systems in substations, 2003, URL <https://webstore.iec.ch/en/publication/6028>.
- [25] P.P. International, PROFINET standard, 2003, URL <https://www.profinet.com/profinet-technology>.
- [26] ASHRAE, ANSI/ASHRAE standard 135-1995: BACnet — A data communication protocol for building automation and control networks, 1995, URL <https://www.ashrae.org/technical-resources/standards-and-guidelines/standards-addenda/standard-135-2016-bacnet-a-data-communication-protocol-for-building-automation-and-control-networks>.
- [27] G. Barbieri, M. Conti, N.O. Tippenhauer, F. Turrin, Assessing the use of insecure ics protocols via ixp network traffic analysis, in: *2021 International Conference on Computer Communications and Networks (Icccn)*, IEEE, 2021, pp. 1–9.
- [28] P.P. International, PROFINET security classes, 2019, URL <https://www.profinet.com/profinet-explained/security>.
- [29] ASHRAE, BACnet secure connect (BACnet/SC), 2019, URL https://www.ashrae.org/File%20Library/Technical%20Resources/Bookstore/BACnet-SC-Whitepaper-v15_Final_20190521.pdf.
- [30] F. Hauser, M. Schmidt, M. Häberle, M. Menth, P4-MACsec: Dynamic topology monitoring and data layer protection with MACsec in P4-based SDN, *IEEE Access* 8 (2020) 58845–58858.
- [31] F. Hauser, M. Häberle, M. Schmidt, M. Menth, P4-ipsec: Site-to-site and host-to-site vpn with ipsec in p4-based sdn, *IEEE Access* 8 (2020) 139567–139586.
- [32] F. Hauser, M. Häberle, M. Menth, P4sec: Automated deployment of 802.1 X, Ipsec, and MACsec network protection in P4-based SDN, *IEEE Access* 11 (2023) 56300–56309.
- [33] X. Chen, Implementing AES encryption on programmable switches via scrambled lookup tables, in: *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, 2020, pp. 8–14.
- [34] I. Oliveira, E. Neto, R. Immich, R. Fontes, A. Neto, F. Rodriguez, C.E. Rothenberg, Dh-aes-p4: on-premise encryption and in-band key-exchange in p4 fully programmable data planes, in: *2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, IEEE, 2021, pp. 148–153.
- [35] K. Keckskeméti, C. Györgyi, P. Vörös, S. Laki, In-network security applications with p4rrot, in: *Proceedings of the Twenty-Fourth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, 2023, pp. 346–351.
- [36] C. Györgyi, S. Laki, S. Schmid, P4rrot: Generating p4 code for the application layer, *ACM SIGCOMM Comput. Commun. Rev.* 53 (1) (2023) 30–37.
- [37] A. AlSabeh, E. Kfoury, J. Crichigno, E. Bou-Harb, P4ddpi: Securing p4-programmable data plane networks via dns deep packet inspection, in: *NDSS Symposium 2022*, 2022.
- [38] A. AlSabeh, K. Friday, E. Kfoury, J. Crichigno, E. Bou-Harb, On dga detection and classification using p4 programmable switches, *Comput. Secur.* 145 (2024) 104007.
- [39] Z. Hu, et al., Industrial network protocol security enhancement using programmable switches, in: *IEEE SmartGridComm*, 2023.
- [40] R.S. Raj, D. Jin, Dynamic data driven security framework for industrial control networks using programmable switches, in: *Dynamic Data Driven Applications Systems: 5th International Conference, DDDAS/Infosymbiotics for Reliable AI 2024*, New Brunswick, NJ, USA, November 6–8, 2024, *Proceedings*, Springer Nature, 2025, p. 153.
- [41] D. Berardi, F. Callegati, A. Giovine, A. Melis, M. Prandini, L. Rinieri, When operation technology meets information technology: challenges and opportunities, *Future Internet* 15 (3) (2023) 95.
- [42] K.E. Hemsley, D.R. E. Fisher, History of Industrial Control System Cyber Incidents, Technical Report, Idaho National Laboratory (INL), Idaho Falls, ID (United States, 2018, <http://dx.doi.org/10.2172/1505628>, URL <https://www.osti.gov/biblio/1505628>.
- [43] J. Bodenhausen, S. Mangel, T. Vogt, M. Henze, Bidirectional TLS handshake caching for constrained industrial IoT scenarios, 2025, arXiv preprint arXiv: 2508.03321.
- [44] ISA/IEC, Security for Industrial Automation and Control Systems – Part 3-2: Security Risk Assessment for System Design, Standard IEC 62443-3-2:2020, International Electrotechnical Commission, 2020, URL <https://webstore.iec.ch/en/publication/30727>.

- [45] T.J. Williams, The Purdue enterprise reference architecture, *Comput. Ind.* 24 (2–3) (1994) 141–158.
- [46] International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), ISO/IEC 62443-3-3: Industrial Communication Networks - Network and System Security - Part 3-3: System Security Requirements and Security Levels, ISO/IEC, 2013, Available from ISO, <https://www.iso.org/standard/65696.html>.
- [47] D. Dolev, A. Yao, On the security of public key protocols, *IEEE Trans. Inform. Theory* 29 (2) (1983) 198–208.
- [48] S.G. Abbas, M.O. Ozmen, A. Alsaheel, A. Khan, Z.B. Celik, D. Xu, {*SAIN*}: Improving {*ICS*} attack detection sensitivity via {*State-Aware*} invariants, in: 33rd USENIX Security Symposium (USENIX Security 24), 2024, pp. 6597–6613.
- [49] M. Ike, K. Phan, K. Sadoski, R. Valme, W. Lee, Scaphy: Detecting modern ics attacks by correlating behaviors in scada and physical, in: 2023 IEEE Symposium on Security and Privacy, SP, IEEE, 2023, pp. 20–37.
- [50] H.R. Ghaeini, D. Antonioli, F. Brasser, A.-R. Sadeghi, N.O. Tippenhauer, State-aware anomaly detection for industrial control systems, in: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, 2018, pp. 1620–1628.
- [51] A. Erba, R. Taormina, S. Galelli, M. Pogliani, M. Carminati, S. Zanero, N.O. Tippenhauer, Constrained concealment attacks against reconstruction-based anomaly detectors in industrial control systems, in: Proceedings of the 36th Annual Computer Security Applications Conference, 2020, pp. 480–495.
- [52] Y.-C. Wang, P.-Y. Su, Collaborative defense against hybrid network attacks by SDN controllers and P4 switches, *IEEE Trans. Netw. Sci. Eng.* 11 (2) (2023) 1480–1495.
- [53] N.I. of Standards, T. (NIST), M.J. Dworkin, E. Barker, J. Nechvatal, J. Foti, L.E. Bassham, E. Roback, J.D. Jr., Advanced encryption standard (AES), 2001, <http://dx.doi.org/10.6028/NIST.FIPS.197>, URL https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=901427.
- [54] T. Hansen, D.E.E. 3rd, US Secure Hash Algorithms (SHA and HMAC-SHA), RFC 4634, 2006, <http://dx.doi.org/10.17487/RFC4634>, URL <https://www.rfc-editor.org/info/rfc4634>.
- [55] Amazon Web Services, AWS IoT core: Developer guide, 2025, URL <https://docs.aws.amazon.com/iot/latest/developerguide/mqtt.html>. (Online Accessed 18 September 2025).
- [56] A. Al Sadi, M. Savi, A. Melis, M. Prandini, F. Callegati, Unleashing dynamic pipeline reconfiguration of P4 switches for efficient network monitoring, *IEEE Trans. Netw. Serv. Manag.* 21 (3) (2024) 3482–3497, <http://dx.doi.org/10.1109/TNSM.2024.3377538>.
- [57] L. Rinieri, A. Iacobelli, A. Al Sadi, A. Melis, F. Callegati, M. Prandini, In-network encryption for secure industrial control systems communications, in: 2024 IEEE 10th International Conference on Network Softwarization, NetSoft, IEEE, 2024, pp. 190–194, <http://dx.doi.org/10.1109/NetSoft60951.2024.10588908>.
- [58] D. Formby, M. Rad, R. Beyah, Lowering the barriers to industrial control system security with {*GRFICS*}, in: 2018 USENIX Workshop on Advances in Security Education (ASE 18), 2018.
- [59] C. Fung, E. Zeng, L. Bauer, Attributions for ML-based ICS anomaly detection: From theory to practice, in: Proc. 31st Netw. Distrib. Syst. Secur. Symp, 2024.
- [60] C. Gao, Y. Yao, Determining thresholds in three-way decisions with chi-square statistic, in: International Joint Conference on Rough Sets, Springer, 2016, pp. 272–281.
- [61] C. Fung, S. Srinarasi, K. Lucas, H.B. Phee, L. Bauer, Perspectives from a comprehensive evaluation of reconstruction-based anomaly detection in industrial control systems, in: European Symposium on Research in Computer Security, Springer, 2022, pp. 493–513.
- [62] T.P.L. Consortium, P4runtime specification, 2024, URL <https://p4.org/wp-content/uploads/sites/53/2024/10/P4Runtime-Spec-v1.4.1.pdf>.
- [63] D.K. Gillmor, Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS), RFC 7919, 2016, <http://dx.doi.org/10.17487/RFC7919>, URL <https://www.rfc-editor.org/info/rfc7919>.
- [64] M.A. Vieira, M.S. Castanho, R.D. Pacífico, E.R. Santos, E.P.C. Júnior, L.F. Vieira, Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications, *ACM Comput. Surv.* 53 (1) (2020) 1–36.
- [65] S. Miano, F. Risso, M.V. Bernal, M. Bertrone, Y. Lu, A framework for eBPF-based network functions in an era of microservices, *IEEE Trans. Netw. Serv. Manag.* 18 (1) (2021) 133–151.
- [66] Y. Gao, Z. Wang, A review of P4 programmable data planes for network security, *Mob. Inf. Syst.* 2021 (2021) 1–16, 1257046.
- [67] A.-A. Agape, M.C. Danceanu, R.R. Hansen, S. Schmid, P4fuzz: Compiler fuzzer for dependable programmable dataplanes, in: Proceedings of the 22nd International Conference on Distributed Computing and Networking, 2021, pp. 16–25.
- [68] H. Wang, R. Soulé, H.T. Dang, K.S. Lee, V. Shrivastav, N. Foster, H. Weatherpoon, P4fpga: A rapid prototyping framework for p4, in: Proceedings of the Symposium on SDN Research, 2017, pp. 122–135.
- [69] S. Ibanez, G. Brebner, N. McKeown, N. Zilberman, The p4-> netfpga workflow for line-rate packet processing, in: Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2019, pp. 1–9.
- [70] D. Scholz, A. Oeldemann, F. Geyer, S. Gallenmüller, H. Stubbe, T. Wild, A. Herkersdorf, G. Carle, Cryptographic hashing in p4 data planes, in: 2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS, IEEE, 2019, pp. 1–6.
- [71] A. Kumar, S. Patnaik, N. Sable, M.R. Raghupatruni, R. AN, D. Chitti, S. Padubidri, Programmable architecture for offloading PDPC on FPGA-capable SmartNIC, in: 2024 IEEE Future Networks World Forum, FNWF, IEEE, 2024, pp. 226–231.