MADTLS: Fine-grained Middlebox-aware End-to-end Security for Industrial Communication

Eric Wagner
eric.wagner@fkie.fraunhofer.de
Fraunhofer FKIE
RWTH Aachen University

David Heye
david.heye@rwth-aachen.de
RWTH Aachen University
Fraunhofer FKIE

Martin Serror
martin.serrror@fkie.fraunhofer.de
Fraunhofer FKIE

Ike Kunze
kunze@comsys.rwth-aachen.de
RWTH Aachen University

Klaus Wehrle
wehrle@comsys.rwth-aachen.de
RWTH Aachen University

Martin Henze
henze@spice.rwth-aachen.de
RWTH Aachen University
Fraunhofer FKIE

ABSTRACT

Industrial control systems increasingly rely on middlebox functionality such as intrusion detection or in-network processing. However, traditional end-to-end security protocols interfere with the necessary access to in-flight data. While recent work on middlebox-aware end-to-end security protocols for the traditional Internet promises to address the dilemma between end-to-end security guarantees and middleboxes, the current state-of-the-art lacks critical features for industrial communication. Most importantly, industrial settings require fine-grained access control for middleboxes to truly operate in a least-privilege mode. Likewise, advanced applications even require that middleboxes can inject specific messages (e.g., emergency shutdowns). Meanwhile, industrial scenarios often expose tight latency and bandwidth constraints not found in the traditional Internet. As the current state-of-the-art misses critical features, we propose Middlebox-aware DTLS (MADTLS), a middlebox-aware end-to-end security protocol specifically tailored to the needs of industrial networks. MADTLS provides bit-level read and write access control of middleboxes to communicated data with minimal bandwidth and processing overhead, even on constrained hardware.

CCS CONCEPTS

• **Networks** → **Middle boxes / network appliances**; • **Security and privacy** → **Security protocols**; **Hash functions and message authentication codes**.

KEYWORDS

industrial IoT, end-to-end security, middlebox

ACM Reference Format:
Eric Wagner, David Heye, Martin Serror, Ike Kunze, Klaus Wehrle, and Martin Henze. 2024. MADTLS: Fine-grained Middlebox-aware End-to-end Security for Industrial Communication . In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '24), July 1–5, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3634737.3637640

ASIA CCS '24, July 1–5, 2024, Singapore, Singapore

ACM ISBN 979-8-4007-0482-6/24/07
https://doi.org/10.1145/3634737.3637640

1 INTRODUCTION

With the rise of the Industrial Internet of Things (IIoT), Industrial Control Systems (ICSs) heavily rely on machine-to-machine communication between constrained devices to realize control of delicate physical processes [42]. Here, the timely and frequent exchange of short and predictable sensor and command messages is essential to ensure precise control over, *e.g.*, robot arms [52]. However, this reliable data exchange is heavily threatened by a surge of attacks on the underlying industrial networks [5]. Due to a widespread lack of security measures in these networks, attackers can eavesdrop on traffic or even manipulate it, potentially causing severe (physical) harm [27]. To thwart such attacks, the de-facto standard is end-to-end security, primarily in the form of TLS.

While TLS experiences wide-scale adoption in the traditional Internet, the prospects of deployments in industrial networks look rather grim. One major roadblock in deploying end-to-end security is the widespread use of middleboxes that rely on deep packet inspection: Traditional middlebox functionality, such as intrusion detection, requires access to sensor and actuator data to monitor industrial processes [61], and the increasing interest in in-network computing likewise requires read and write access to in-flight data [43]. Hence, middleboxes prevent the deployment of traditional end-to-end security, raising a serious security dilemma.

To depict this issue, we consider an illustrative example of how interconnected IIoT communication may look in the future in Figure 1. A controller operates a robot arm. An intrusion detection system (IDS) and a traffic logging server monitor their communication. The industrial IDS analyzes traffic to detect anomalies and flags suspicious packets. The traffic logging server only captures flow metadata but stores suspicious packets entirely for potential subsequent forensic investigations.

The naïve approach for middlebox access control while still providing security is to create separate secure point-to-point connections, one from robot arm to IDS, one from IDS to logging server, and a final one from logging server to controller. This approach, commonly referred to as SplitTLS [46], gives each middlebox full access to manipulate messages. Consequently, a compromised IDS could move the robot arm unexpectedly and thus damage expensive equipment or cause physical harm. Since middleboxes are typically placed at critical vantage points with access to large amounts of traffic, they then become especially attractive targets for attackers. Therefore, it is of utmost importance to deploy middleboxes in

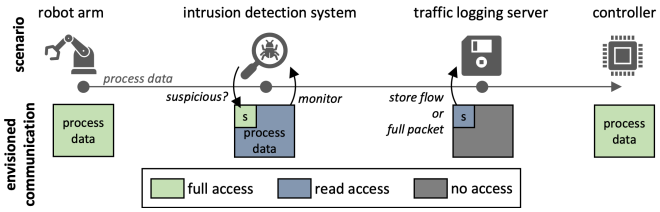


Figure 1: Middleboxes should operate in a least-privilege mode. For example, an industrial IDS has read-only access to packets, and write access to a dedicated flag to mark suspicious traffic. A logging server can only read this flag.

a *least-privilege mode*. Considering the IDS from our example, it should only have read access, except for writing to a single flag to mark suspicious packets. In contrast, SplitTLS provides full insight and control over the entire communication channel to middleboxes.

To address these limitations, *middlebox-awareness* has been proposed to adapt end-to-end security protocols to the reality of middleboxes in corporate networks and the Internet [17]. A first branch of research relies on searchable encryption [9, 36, 47, 55] or zero-knowledge proofs [23, 63] to perform a limited set of computations (*e.g.*, string matching) on encrypted data. While these approaches may work for basic rule-based intrusion detection, they are too restrictive for most middleboxes in the industrial context. Other proposals move middleboxes into Trusted Execution Environments (TEEs) [19, 26, 49, 56]. While such approaches work great in theory, the secure implementation of the concept of TEEs is difficult in practice, as recent attacks have shown [8, 44]. Finally, a branch of research extends TLS to allow for the authorization of on-path middleboxes to read or write to a predetermined set of communicated data [4, 21, 37, 38, 45, 46]. Still, operating on the granularity of complete messages or only supporting two access modes, these proposals do not provide the fine-grained access control necessary to operate middleboxes in a least-privilege mode.

While middlebox-aware end-to-end security thus offers an attractive solution for the security dilemma, current proposals do not address the unique challenges of industrial communication. First, tight latency and bandwidth demands, paired with resource-constrained embedded hardware, require fast and efficient protocols [42]. Secondly, predictable and well-structured messages enable, but also require, fine-grained control down to the *bit level* (instead of complete messages) over the access rights of each involved middlebox to constrain their privileges to a minimum. Thereby, we can minimize the damage inflicted by potentially compromised middleboxes. Thirdly, middleboxes may need to inject entirely new messages (*e.g.*, emergency shutdowns) into established communication channels, where the least-privilege principle must also be upheld.

This paper tackles these state-of-the-art limitations by proposing *Middlebox-Aware DTLS* (MADTLS). In short, MADTLS provides fine-grained access control to industrial communication via specialized cryptographic protocols. We enable the transparent segmentation of messages to assign read and write access on a per-segment granularity. Therefore, each segment is encrypted and authenticated individually. Here, MADTLS leverages a specifically tailored message authentication scheme to aggregate authentication data, thus conserving valuable bandwidth. Moreover, MADTLS provides an efficient extension to the DTLS 1.2 handshake protocol to exchange the additional information required by the communicating endpoints and middleboxes. While industrial networks may rely on reliable (*e.g.*, TCP) or lossy channels (*e.g.*, UDP, with a recent focus on wireless communication), we specifically target the more challenging domain where packets may be lost arbitrarily. Still, the techniques designed in this paper are easily transferable to traditional TLS (over TCP) and other end-to-end protocols.

Contributions. To unlock the potential of middlebox-aware end-to-end security for resource-constrained industrial networks, we make the following contributions in this paper:

- We provide a taxonomy of middlebox use cases in industrial communication and derive corresponding requirements for middlebox-aware end-to-end security (Sections 2 & 3).
- To meet these requirements, we design MADTLS, which relies on specifically tailored cryptographic schemes to enable the distribution of fine-grained (down to the bit level) read and write access rights to middleboxes (Section 4).
- To show the practical applicability and feasibility of our approach, we prototypically implement MADTLS as DTLS 1.2 extension and show its competitive performance in realistic scenarios on representative hardware.

2 MIDDLEBOXES IN INDUSTRIAL NETWORKS

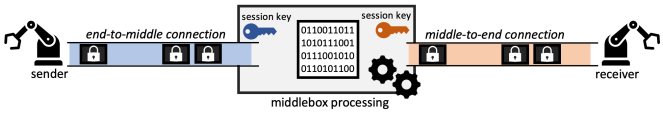
Middleboxes play an integral role in industrial networks for performance enhancements and intrusion detection [40], but they also severely hinder the adoption of traditional end-to-end security. In the following, we first give a brief background on industrial networks and why their unique properties hinder the use of traditional security protocols (Section 2.1). Afterwards, we provide an overview of middlebox functionalities in industrial networks, specifically focusing on the respective required access to communicated data (Section 2.2). Finally, we derive why the current state-of-the-art in middlebox-aware end-to-end security protocols cannot cope with the inherent requirements of industrial networks (Section 2.3).

2.1 Security Challenges in Industrial Networks

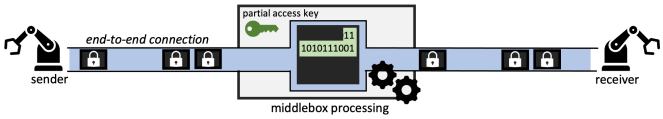
Industrial networks are dominated by machine-to-machine communication that keeps physical processes running safely. Depending on the underlying process, this communication can be subject to harsh requirements w.r.t. latency and bandwidth [22]. In particular, a plethora of sensors and actuators need to frequently exchange messages, hence per-message bandwidth overhead is especially expensive. Meanwhile, devices must often be kept small and cheap, such that their processing resources are heavily constrained.

Communication has traditionally been facilitated via (reliable) cabled connections using different application layer and subordinate protocols. Today, ICS networks see a notable shift toward wireless and, thus, lossy communication, partly enabled by new protocols such as Sigfox or WirelessHART. Therefore, any security solution cannot be tailored to a single protocol or communication medium but must be adaptable to reliable as well as lossy communication.

Concerning security, encrypted and authenticated end-to-end communication is still rarely seen within industrial networks [16], despite being commonplace in the traditional Internet. The main



(a) SplitTLS solutions allow full access to middleboxes.



(b) A least-privilege access for middleboxes would be preferable.

Figure 2: While SplitTLS allows full access to middleboxes, achieving least-privilege access for middleboxes should be the goal for middlebox-aware security protocols.

reason for this lack of security is that, in the past, industrial networks were designed without security as their inherent physical separation from attackers seemingly sufficed. As this assumption crumbles due to increased connectivity demands from industry (*e.g.*, remote monitoring) and more advanced attacks, industrial networks become increasingly exposed to cyberattacks. This vulnerability is demonstrated impressively through a rising numbers of attacks with, at times, detrimental consequences [5].

Addressing this lack of security and also accounting for the high demands in industrial networks, we observe an increasing interest in middlebox deployments [43]. Security-wise, these middleboxes can, *e.g.*, realize deep packet inspection for intrusion detection. Regarding performance, the full range of functionality extends to a diverse set of tasks including caching [24, 39], data aggregation [53], fault detection [54], and in-network processing [15, 51, 52].

While these advancements alleviate specific limitations and *some* security flaws within the IIoT, they also hinder properly deploying end-to-end security protocols, as we demonstrate in Figure 2. Current industrial deployments of end-to-end security are mainly realized with TLS or DTLS and can, at most, provide a so-called SplitTLS [46] solution, where middleboxes can freely access and modify any traffic that passes through them (*cf.* Figure 2a). However, middleboxes are usually deployed for a particular task, so an ideal security protocol would restrict read and write access to a minimum (*cf.* Figure 2b). Only then can the IIoT operate in a least-privilege mode and minimize the damage of compromised middleboxes.

2.2 Diversity of Industrial Middlebox Use Cases

To understand the requirements for middlebox-aware end-to-end security in industrial networks, we need to understand the range of potential tasks. Therefore, we categorize corresponding middlebox applications according to their required access rights.

Read Access. Examples for read access include a middlebox that caches, *e.g.*, sensor readings to unburden the constrained end devices [24, 39]. Similarly, industrial IDSs monitoring the physical process to detect suspicious activities only require read access to otherwise encrypted data for deep packet inspection [61].

Limited Read Access. However, even for tasks such as intrusion detection or caching, middleboxes often only require partial insight into each message. For example, most Snort [3] rulesets for Modbus (a widespread industrial communication protocol) do not look beyond the function code field (indicating the type of a message, *e.g.*, request the state of an individual bit). Even more sophisticated industrial IDSs only require partial read access to messages in some cases [13, 14, 41]. Moreover, other use cases only require partial read access by design: For fault detection, middleboxes only need access to selected sensor readings from specific machinery to detect upcoming failures [54]. Similar access rights to only specific sensor readings are necessary for complex event detection, *e.g.*, the outbreak of a fire [31, 43, 58].

Write Access. Middlebox tasks that require *full* write access to messages are rare, *e.g.*, when the middlebox translates between different application layer protocols [1, 57]. Instead, most middleboxes that alter messages only require limited write access.

Limited Write Access. A typical example of limited write access in the industrial context and beyond is data compression, where a middlebox can, *e.g.*, base-delta encode timestamps for many different data sources [48]. Similarly, in-network aggregation or other map-reduce functionality can be executed by middleboxes that only have write access to the corresponding data fields they are reducing [53]. More industry-specific applications for restricted write access include, *e.g.*, the transformation of coordinates between reference frames [34] or the insertion of precise timestamps into payload data [33]. Furthermore, as seen in the example in Section 1, various middleboxes may take advantage of flagging individual packets, *e.g.*, to mark them as suspicious.

Drop Messages. In the industrial context, it is often not desirable to directly drop messages due to irrevocable impact on the physical process controlled by the system. Thus, industrial networks mostly only employ IDSs that flag suspicious traffic or alert the operator through other channels. Still, for some use cases, a middlebox requires the ability to drop messages even in industrial networks, *e.g.*, to downsample sensor readings [35, 58], carefully reduce traffic [25], or if a serious cyberattack is identified with high likelihood (*i.e.*, the benefit of likely preventing a high-impact attack outweighs the risk of blocking genuine traffic).

Inject Messages. Finally, middleboxes may also require the ability to inject messages, most importantly when a middlebox directly responds to a request or event to reduce latency or traffic. Complex event detection could, *e.g.*, identify critical conditions such as a fire [31], that warrant the issuing of emergency stop messages [15]. Other reasons to allow middleboxes to inject messages include responding caching servers [24, 39] or the enabling of low-latency and low-jitter control commands [15, 51, 52].

Overall, we see that middleboxes in the IIoT cover a diverse set of functionalities that require different levels of access to a communication channel. Most importantly, these functionalities are often specific to the IIoT and must be considered when designing middlebox-aware end-to-end security.

2.3 Prior Work on Middlebox-aware Security

Research on such *middlebox-aware security* protocols goes back over a decade for the traditional Internet [17], but we still see few deployments today. Corresponding concerns can, among other things, be traced back to the Internet community's end-to-end principle [10, 11], which is typically interpreted to imply that the

Publication	Year	Venue	Mechanism	read-access		write-access		inject	drop	diff.
				full	limited	full	limited			
BlindBox [55]	2015	SIGCOMM	searchable encryption	◐	◐	○	○	○	●	○
Embark [36]	2016	USENIX NSDI	searchable encryption	◐	◐	○	○	○	●	◐
BlindIDS [9]	2017	ACM AsiaCCS	searchable encryption	◐	◐	○	○	○	●	○
PrivDPI [47]	2019	ACM CCS	searchable encryption	◐	◐	○	○	○	●	○
ZKMB [23]	2022	USENIX Security	zero knowledge proofs	◐	◐	○	○	○	◐	●
Zombie [63]	2023	IEEE S&P	zero knowledge proofs	◐	◐	○	○	○	◐	●
SGX-Box [26]	2017	APNet	trusted execution environment	●	○	●	○	○	◐	○
Safebricks [49]	2018	USENIX NSDI	trusted execution environment	●	◐	●	◐	○	◐	●
Shieldbox [56]	2018	ACM SOSR	trusted execution environment	●	○	●	○	○	◐	○
Lightbox [19]	2019	ACM CCS	trusted execution environment	●	○	●	○	○	◐	○
mcTLS [46]	2015	SIGCOMM	protocol extension	●	◐	●	◐	○	○	●
mbTLS [45]	2017	ACM CoNEXT	protocol extension	○	○	●	○	○	○	●
maTLS [37]	2019	NDSS	protocol extension	●	◐	●	◐	○	○	●
ME-TLS [38]	2019	IEEE IoTJ	protocol extension	●	◐	●	◐	○	○	●
Stealth Key Exchange [21]	2023	ACM CCS	protocol extension	●	◐	●	◐	○	○	◐
mdTLS [4]	2023	ICISC	protocol extension	●	◐	●	◐	○	○	●
MADTLS	2024	ACM AsiaCCS	protocol extension	●	●	●	●	●	◐	●

● : yes ◐ : partial ○ : no

Table 1: The current state-of-the-art on middlebox-aware security protocols cannot address all requirements of industrial networks. Searchable encryption or zero-knowledge proofs only provide limited functionality (*e.g.*, string matching). Vulnerabilities within TEEs expose all approaches relying on them. Finally, extensions to the TLS protocol do not provide the fine-grained access control required to truly operate middleboxes in a least-privilege mode. Moreover, no proposal offers features to give middleboxes the ability to inject (a restricted set of) messages into the communication stream.

functionality in the network should be kept minimal while end-hosts implement most, if not all, functionality. Thus, especially in security contexts, the addition of middleboxes terminating end-to-end connections is often regarded as a slippery slope toward security and privacy loss on the Internet. In contrast to the general Internet, limited domains [12] allow for more liberal solutions that can be tailored to the needs of the domain. Industrial networks are one prominent example of such a limited domain, as they are typically under a single administrative control. Additionally, all devices follow a common goal: ensuring the successful operation of the industrial process. Industrial networks and other limited domains thus represent a contrasting deployment scenario compared to the Internet, calling to revisit secure middlebox-aware communication specifically from the perspective of industrial communication. Since current proposals for middlebox-aware end-to-end security protocols are designed for the general Internet, we now investigate to which extent existing proposals are suited for industrial deployments. Table 1 summarizes the results of our analysis.

An initial set of proposals attempts to realize middlebox functionality directly on encrypted traffic [9, 36, 47, 55]. Here, BlindBox [55] introduces the original idea but only enables the functionality to evaluate regular expressions on encrypted data. Subsequent efforts extend this functionality [36], improve performance by reusing computations [47], or focus on specific use cases such as intrusion detection [9]. However, searchable encryption enables only a limited set of computations on network traffic, does not support altering or injecting traffic, and brings unacceptable performance penalties to resource-constrained industrial devices. A similar picture is drawn by the recent first proposal to employ zero-knowledge proofs provided by clients that attest the abidance to certain rules, *e.g.*, to prevent routing traffic to a blacklisted IP address [23]. Again, this approach has significant performance drawbacks, restricted functionality, and no support for altering or injecting traffic.

A fundamentally different idea is to encapsulate middlebox functionality into TEEs to shield them against malicious or compromised hosts [19, 26, 49, 56]. These approaches, such as SGX-Box [26] share TLS session keys with the TEE. Within the TEE, packets are entirely decrypted and can even be altered by the middlebox. Lightbox [19] further protects traffic metadata and enables stateful middleboxes. Shieldbox [56] facilitates the implementation of middleboxes in Intel SGX via the CLICK framework [30]. Meanwhile, SafeBricks [49] restricts middleboxes to only partially access packets and improves the performances of chained middleboxes. However, the access restriction of SafeBricks does not cryptographically protect against malicious middleboxes and relies on a correct realization of Rust's type system and the security of the TEE itself. Still, the limited memory of TEEs impedes the application of these proposals even if TEEs were available in industrial settings. Furthermore, all protocols are designed for TLS, such that dropping individual messages implies that all future sequence numbers must be adapted. More importantly, all these approaches assume a secure implementation of the TEE primitive, which a plethora of recent attacks (*e.g.*, Plundervolt [44] or Æpic leak [8]) have shown to be difficult.

Finally, recent work investigates the direct integration of middleboxes into TLS sessions via protocol extensions [4, 21, 37, 38, 45, 46]. In mcTLS [46], each message is assigned a preconfigured context consisting of a unique set of middleboxes with read and/or write access. The TLS record protocol header is then extended by a context identifier and two authentication tags, one for readers and one for writers. Readers verify the reader tags, writers verify reader and writer tags, and endpoints verify all tags. Thus, readers can verify that no third party modified the packet, writers know that

any modification stem from writers, and endpoints additionally know whether the packet has been modified at all. Still, mcTLS only provides coarse access control on a per-message level. As an alternative to mcTLS, maTLS [37] proposes to use different TLS sessions for middleboxes and append a modification log to each packet to track changes. Despite performance improvements in mdTLS [4], this approach introduces significant processing delay and bandwidth overhead. In turn, ME-TLS [38] improves the handshake efficiency over that of mcTLS by providing authorized middleboxes with the necessary information to recover session keys from passively observed handshake messages. Furthermore, mbTLS [45] enables the dynamic integration of middleboxes into a special TLS session where a new key is used for each middlebox to enforce path integrity, but without restricting data access. Stealth key exchanges [21] take yet another route by exchanging a secondary encryption key in standard-conform TLS 1.3 communication that can be used to keep traffic encrypted and/or authenticated when sharing the primary session key with a middlebox.

Overall, even these closely related protocol proposals only provide access control on a per-message basis and often introduce significant overhead. In general, the current state-of-the-art on middlebox-aware end-to-end security protocols cannot provide the fine-grained access control to packets necessary to operate middleboxes in a least-privilege mode. Furthermore, no current proposal covers the case of middleboxes injecting traffic, as may be necessary within industrial applications to, *e.g.*, issue emergency stop commands [15] or reduce the latency of control tasks [15, 51, 52]. Consequently, we conclude that current middlebox-aware end-to-end security protocols are not suited for industrial networks.

3 THREAT MODEL & REQUIREMENTS

Despite extensive research on middlebox-aware end-to-end security, industrial networks can still not be efficiently equipped with such functionality. Meanwhile, these networks offer a prime target for such approaches as established middlebox deployments often prevent other security measures. In the following, we first establish the threat model against which middlebox-aware end-to-end security (in the industrial networks) must protect (Section 3.1). Afterwards, we distill concrete requirements that must be fulfilled to enable the least-privilege operation of middleboxes (Section 3.2).

3.1 Threat Model

We strive to prevent attacks aiming at unauthorized read or write access to communication channels in industrial networks. To achieve this goal, we consider an attacker according to the Dolev-Yao threat model [18], *i.e.*, an attacker which has complete control over the entire network (but neither of the two endpoints of a communication channel). Accordingly, the attacker can arbitrarily read, alter, reroute, inject, and drop packets. Additionally, an attacker may arbitrarily compromise one or multiple middleboxes which access the communication channel. Within our threat model, the attack must be constrained from extending their control over a communication session beyond the minimum access requirements any compromised middleboxes have over that session. Furthermore, an attack must be prevented from altering middleboxes' order or skipping some middleboxes entirely, as this may lead to incorrect data being processed or ineffective intrusion detection. Denial of Service (DoS) attacks and side-channels attacks are out of scope in this paper as these kinds of attacks affect all security protocols.

3.2 Requirements

Middlebox-aware security protocols should provide the same security guarantees as end-to-end protocols towards outsiders, *i.e.*, entities not part of the communication. Concretely, any communication session should authenticate the communication endpoints, provide data secrecy, and ensure data integrity. Beyond these inherited requirements, middlebox-aware end-to-end security protocols (for industrial networks) must fulfill additional requirements regarding the integration of middleboxes into communication sessions.

Explicit Middlebox Authentication. The authentication of endpoints in end-to-end security must be extended to all middleboxes with read or write access to any message. Thus, both endpoints must explicitly acknowledge and verify all middleboxes involved in the communication and the privileges they got assigned.

Least Privilege Read and Write Access. Middleboxes are often located at critical vantage points to process as much traffic as possible. This makes middleboxes a particularly attractive target for attacks, while they often fulfill dedicated tasks which require read and/or write access to specific parts of messages. Consequently, middleboxes should operate in a least-privilege mode where they are restricted to exactly those access rights that are inevitable to fulfill their task. The selection of applications from Section 2.2 shows that this access might have to be restricted to bit-wise read and/or write access to specific fields in a message.

Least Privilege Traffic Injection. Some middlebox tasks need to inject control commands for low-latency control or emergency shutdowns (Section 2.2). However, such abilities must not be accompanied with full control over the communication channels, *i.e.*, a middlebox should not be able to inject arbitrary traffic. Instead, privileges to inject traffic must again be restricted to the minimum required for correct functionality, *e.g.*, to only inject messages with specific Modbus function codes.

Path Integrity. Generally, the order in which middleboxes process a message is important. A middlebox performing complex computation may, *e.g.*, need to be placed behind a filtering middlebox to be able to keep up at line-rate. Therefore, an attacker should not be able to change the processing order of middleboxes, or worse, skip certain middleboxes (*e.g.*, an IDS) entirely. To prevent such attacks, a middlebox-aware security protocol should enforce path integrity, *i.e.*, a message's correct verification depends on it passing all intended middleboxes in the right order.

Accounting for Resource Constraints. Middlebox-aware end-to-end security for industrial networks has to account for resource-constrained devices and networks. More specifically, adequate latencies must be ensured even with limited processing power. Additionally, any per-message overhead for short messages should be minimized to conserve bandwidth.

Our requirements show similarities with the traditional Internet (*e.g.*, path integrity [45]), but also a range of challenges unique to the IIoT (*e.g.*, least-privilege traffic injection). As the current state-of-the-art cannot fulfill all these requirements, we propose a middlebox-aware security protocol tailored to the IIoT.

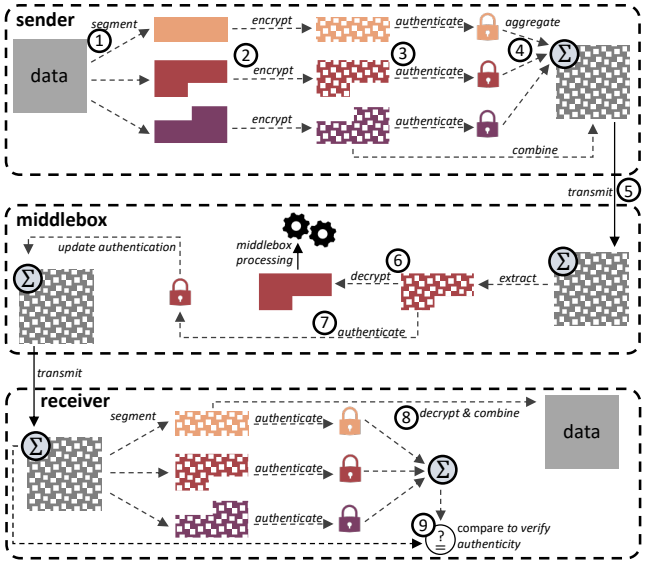


Figure 3: The core idea behind MADTLS is to segment messages according to predetermined templates, which allows to encrypt and authenticate each segment individually. To save bandwidth, segment authentication is only verified by the receiver and selected middleboxes. Therefore, each middlebox updates the authentication tag, such that later verification of this tag ensures path integrity and data integrity at all times.

4 HIGH-LEVEL DESIGN OF MADTLS

To enable middlebox-aware end-to-end security in industrial networks, we propose Middlebox-Aware DTLS (short: MADTLS). We choose to integrate MADTLS as extension to DTLS as it represents the bigger challenge compared to TLS, since any message can be lost during its transmission. However, the integration into, *e.g.*, TLS 1.3 should be possible without significant changes. MADTLS is designed to fulfill the requirements for industrial networks outlined in Section 3.2. Still, MADTLS can also be advantageous in other scenarios such as data center networks.

The main idea underlying MADTLS is to partition messages into segments, which are then individually encrypted and authenticated such that middleboxes can only read or write to a specific subset of those segments. We illustrate the entire process of securing a message before transmission, over the partial access by a middlebox, until the final reception of the message at its destination in Figure 3. First, ① the sender partitions a message into non-overlapping segments such that a specific set of access rights for each middlebox can be assigned to each segment. The resulting segments are then ② separately encrypted (*e.g.*, with AES in counter mode), and ③ an authentication tag is computed for each segment. Notably, the authentication scheme for individual segments is designed such that ④ all authentication tags for individual segments can be aggregated to save valuable bandwidth. The encrypted segments and the aggregated authentication tag are then ⑤ transmitted to their destination and intercepted by a middlebox.

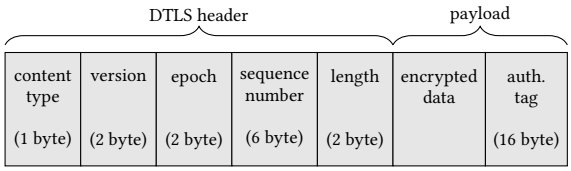


Figure 4: The DTLS 1.2 header format realizes a concise message format that can easily be extended to support new functionality through the addition of content types.

In the example in Figure 3, the middlebox has read access to the segment in the middle (red). After ⑥ decrypting this segment, it can process the contained data to realize its middlebox functionality. Meanwhile, the middlebox also ⑦ updates the aggregated authentication tag such that the final receiver can retrace whether this middlebox received the correct data. Thus, the receiver can verify that no attacker manipulated data before a middlebox processes it, just to revert these changes afterwards.

The final receiver ⑧ decrypts all segments and ⑨ computes an authentication tag over the received data to compare it to the transmitted tag. If both tags match, the integrity of the transmitted data is proven: The receiver *and all middleboxes* received data that has only been modified by middleboxes that have been explicitly allowed to make these changes. After discussing MADTLS on a high level, we now dive into the details of the record layer protocol.

5 THE MADTLS RECORD PROTOCOL

To introduce the idea of middlebox-awareness to industrial networks, we extend the DTLS 1.2 protocol. We start with a summary of the DTLS 1.2 record layer layouts in Section 5.1. Then, in Section 5.2, we discuss how we extend this layout to support limited read and write access for middleboxes. In Section 5.3 and 5.4, we then discuss how encryption and integrity protection are handled. Finally, we discuss extensions to realize integrity verification by middleboxes (Section 5.5) and limited data injection (Section 5.6).

5.1 Background: DTLS Record Layer

We first describe the DTLS record layer in Figure 4. The `content type` is used to identify the type of messages (*e.g.*, DTLS handshake messages are represented by the type `0x16`). New content types can be added to extend the functionality of DTLS. The `version` field indicates the protocol version (*e.g.*, 1.2). Then, DTLS uses two fields that combine to a nonce to prevent replay attacks. The `epoch` tracks cipher suit changes and a zero-initialized `sequence number` increments with each message. The `length` field then indicates the length of the DTLS record layer payload, which is appended afterwards. To this end, the payload is encrypted and an authenticated tag is appended according to the chosen cipher suite. In summary, DTLS realizes a concise message format that can easily be extended to new functionality through new content types.

5.2 MADTLS' Record Layer Header Structure

To realize MADTLS, we extend the DTLS 1.2 header by three new content types (`0x1D`, `0x1E`, and `0x1F`). One of those new content types (`0x1E`) represents the MADTLS record layer. The remaining

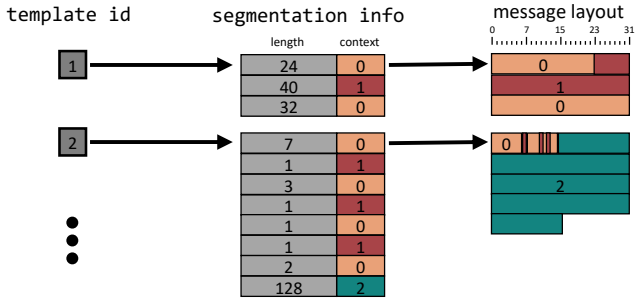


Figure 5: The `segmentation info` describes the layout of a packet and can either be explicitly transmitted, or, for repeating patterns, the mapping between a 1-byte `template id` and it can be communicated during the handshake.

header is extended by a, usually 1-byte long, `segmentation info` field that defines the encrypted payload's segmentation.

This `segmentation info` field starts with two 1-bit flags: The `m`-flag, which is set to enable middlebox authentication (discussed in Section 5.5), and the `l`-flag, which is set if the layout of the data is explicitly indicated. If the `l`-flag is not set, the remaining 6 bits form the `template id` that maps to one of up to 64 pre-exchanged segmentation infos as shown in Figure 5 and explained below.

Conceptually, a plaintext MADTLS message is divided into n segments, addressed as $S[0]$ to $S[n-1]$. Each segment $S[\cdot]$ is assigned a context that defines which middleboxes have read or write access to that segment. The segmentation of a message is then indicated indirectly through the `template id` or by explicitly including the `segmentation info` in the packet header. The `segmentation info` encodes the layout where alternating fields indicate the bitlength of segments and their assigned context. Currently, variable-length fields result in entire segmentation info specifications, but a compact formatting for such cases could be imagined in the future.

As in DTLS, the header precedes the payload that contains the encrypted message and the integrity-protecting authentication tag that is verified by the final receiver of a message. Crucially, MADTLS' authentication tag is updated by middleboxes to ensure data consistency and path integrity. In the following, we discuss the encryption and authentication scheme employed by MADTLS in more detail.

5.3 Segment Encryption

We start by discussing the encryption scheme used by MADTLS, before diving into the more important (for industrial networks) and challenging aspect of integrity protection. Note that encryption is not mandatory in MADTLS, which may be useful for some (industrial) applications that only care about integrity-protection. When using encryption, the use of a single key does obviously not allow for limited read access. Therefore, each segment $S[\cdot]$ is assigned a context c, which is associated with a unique encryption key k_c^{enc}. Only those middleboxes that should read a specific context are then provided with the corresponding key. These keys are distributed during the handshake, as we will discuss in Section 6.

To avoid unnecessary overhead when segments are shorter than a multiple of the block sizes of the cipher (*e.g.*, 16 bytes for AES), MADTLS can take advantage of cipher streams as realized, *e.g.*, by AES in counter mode. This approach ensures that messages do not expand through encryption of individual segments. Each segment is then separately encrypted with the key corresponding to its context, *i.e.*, $C[i] = enc_{k_c^{enc}}(S[i])$, where $C[i]$ is the encrypted plaintext segment $S[i]$. Middleboxes and the final receiver derive either from the *template id* or from the *segmentation info* field which segments they have access to, where those are located, and which keys they must use to decrypt which segment.

5.4 Compact Authentication Scheme

While MADTLS' encryption scheme is straightforward, the similarly trivial approach towards integrity protection is impossible. MADTLS's authentication scheme must differentiate individually between read and write access to all segments. Consequently, we cannot transfer approaches such as mcTLS's three authentication tags [46] to a setting with significantly more fine-granular access control, as its authentication scheme introduces three 16-byte tags for each context and does not protect path integrity.

Therefore, we design a custom authentication scheme for MADTLS that ensures compactness, path integrity, and high performance. To achieve these goals, MADTLS takes advantage of authentication tag aggregation [29] to combine multiple tags into a single tag without loss of security for a verifier that is able to verify each of the aggregated tags individually. While tag aggregation has been explored previously to achieve path integrity [20], MADTLS's authentication scheme only needs to transmit a single tag even if messages are modified or divided into multiple contexts. This design naturally favors the verification of data and path integrity by the final receiver who has access to all contexts to verify all tags. In case the receiver notices that a packet has been manipulated on its path, it can alert the concerned middleboxes or an operator. For now, we focus on this case and describe the design of MADTLS's single authentication tag in the following. We later revisit this limitation in Section 5.5 and see how MADTLS supports the efficient creation of partial authentication tags that middleboxes with limited data access can still verify.

The gist of our authentication scheme is that each node that is authorized to read or write data manipulates the authentication tag in a deterministic way such that the final tag is correct iff no unauthorized manipulation has taken place during message transmission. The manipulation of the tag by each entity with access rights, even those only allowed to read, ensures that no entity receives manipulated data that is subsequently changed back without this being noticeable by the final receiver. In the following, we describe MADTLS' authentication scheme in detail, starting with the key setup. Afterwards, we learn how the sender computes an initial authentication tag and how this tag is subsequently updated by middleboxes according to their access rights.

We have e communication entities and c contexts in a MADTLS session. Each segment $S[\cdot]$ is mapped to a context that uniquely describes a set of access rights for each middlebox. Entities 0 and $e-1$ are the sender and receiver, respectively. All other entities (1 to $e-2$) are middleboxes with read and/or write access to a subset of all segments. The corresponding symmetric keys for access to the i-th context for the j-th entity are denoted as $k_{i,j}$ as shown in Figure 6. For each index, there exist up to two keys, one for read

		communication entities					
		0	1	2	3	…	$e-2$
contexts	0	$k^{\text{read}}_{0,0}$	φ	$k^{\text{read}}_{0,2}$		…	
	1	$k^{\text{read}}_{1,0}$	$k^{\text{read}}_{1,1}$	φ	$k^{\text{read}}_{1,3}$	…	$k^{\text{read}}_{0,e-2}$
	⋮	⋮	⋮	⋮	⋮	⋱	⋮
	$c-1$	$k^{\text{read}}_{c-1,0}$			$k^{\text{read}}_{c-1,3}$	…	

Figure 6: MADTLS assigns read keys to communication entities according to their access rights for each context. The same key assignment is conducted for write keys.

access ($k^{\text{read}}_{i,j}$) and one for write access ($k^{\text{write}}_{i,j}$). As shown in the example of Figure 6, Entity 3 has read access to Context 1 such that $k^{\text{read}}_{1,3}$ exists. As Entity 2 has no read access to this Context, $k^{\text{read}}_{1,2}$ does not exist. The sender has read and write access to all contexts of a message. Thus, all keys $k_{\_,0}$ always exist. Meanwhile, no key $k_{\_,e-1}$ exists as the receiver does not need to authenticate the message to another entity. To update authentication tags for the next entity, the k-th communication entity has access to all keys $k_{\_,k}$ that exist. To update (and verify) tags, each entity additionally knows the *previous* existing key for all contexts it has access to. We denote this previous key as $\varphi(k_{\_,k})$. In the example from Figure 6, $\varphi(k^{\text{read}}_{0,2}) = k^{\text{read}}_{0,0}$ and $\varphi(k^{\text{read}}_{1,3}) = k^{\text{read}}_{1,1}$.

In the following, we explain how the sender uses the keys $k_{\_,0}$ to compute the initial authentication tag. Each entity subsequently updates the aggregated authentication tags by removing the old partial segment tag (*i.e.*, the authentication tag computed only over a segment with the corresponding read or write key) and adding a new tag for each accessed context. As each update requires access to a unique set of keys that only that specific middlebox knows, it cannot be impersonated by another entity. An in-depth discussion of the soundness and security of MADTLS' authentication scheme can be found in the Appendix.

The initial tag is computed as follows, where $\delta(\cdot)$ maps the segment index i to the corresponding context:

$$t = \bigoplus_{0 \le i < n} \left(\sigma_{k^{\text{read}}_{\delta(i),0}}(C[i]) \oplus \sigma_{k^{\text{write}}_{\delta(i),0}}(C[i]) \right)$$

Each segment is authenticated (σ represents a classical message authentication algorithm such as HMAC-SHA256) twice, with the corresponding reading and writing keys, respectively. All computed tags (*i.e.*, reading and writing tags for all segments) are then XOR-ed together, which does not reduce their security [6]. A verifier now needs all keys that were used to compute the individual tags to verify the aggregated tag. This aggregated tag is then appended to the message and transmitted to the first middlebox with limited read or write access to the message.

All middleboxes alter this tag according to their access rights in a deterministic way. A middlebox j that has read access to the segments $\mathbb{S}^{\text{read}}_j$ updates the tag of the message as follows:

$$t \overset{\oplus}{=} \bigoplus_{i \in \mathbb{S}^{\text{read}}_j} \left(\sigma_{\varphi(k^{\text{read}}_{\delta(i),j})}(C[i]) \oplus \sigma_{k^{\text{read}}_{\delta(i),j}}(C[i]) \right)$$

For each segment i in $\mathbb{S}^{\text{read}}_j$, the first part of the equation removes tags from the last entity that had read access to that message segment. This removal works exactly when the segment $C[i]$ has not been changed between the two readers, as only then do the old partial segment tag and the newly computed partial segment tag cancel out. The second part of the equation then computes and integrates a new segment tag with the new key $k^{\text{read}}_{\delta(i),j}$ not known to the previous middlebox.

Besides read access, some middleboxes may also have write access to certain segments $\mathbb{S}^{\text{write}}_j$. Here, we assume that write access implies read access. Formally, this means that $\mathbb{S}^{\text{read}}_j \cap \mathbb{S}^{\text{write}}_j = \emptyset$, *i.e.*, write access to a context cannot be combined with explicit read access. Here, the procedure to amend the authentication tag is similar. First, the old *reading and writing* tags are removed before they are replaced by the new tag computed over the changed segment data $C'[\cdot]$. Formally, this procedure looks as follows:

$$t \overset{\oplus}{=} \bigoplus_{i \in \mathbb{S}^{\text{write}}_j} \Big(\sigma_{\varphi(k^{\text{read}}_{\delta(i),j})}(C[i]) \oplus \sigma_{k^{\text{read}}_{\delta(i),j}}(C'[i]) \oplus \sigma_{\varphi(k^{\text{write}}_{\delta(i),j})}(C[i]) \oplus \sigma_{k^{\text{write}}_{\delta(i),j}}(C'[i]) \Big)$$

Finally, the receiver receives the message as well as the transmitted and updated authentication tag t. Based on the received data, it can then compute t^* as follows:

$$t^* = \bigoplus_{0 \le i < n} \left(\sigma_{\varphi(k^{\text{read}}_{\delta(i),e-1})}(C[i]) \oplus \sigma_{\varphi(k^{\text{write}}_{\delta(i),e-1})}(C[i]) \right)$$

If no unauthorized manipulation of transmitted data took place, t and t^* are identical, and thus the integrity of the message is verified successfully. Otherwise, at least one communication entity has been served with unauthentic data.

5.5 Self-Verifying Middlebox

By default, MADTLS operates in the most resource-conscious mode where only the final receiver verifies a message. Notably, this verification not only covers the authenticity of the message as received at the final destination but also ensures the authenticity of the message as received by each on-path middlebox (with read or write permission). However, further efforts are required to ensure on-path authenticity if the processing of messages by middleboxes causes *side effects*, *i.e.*, has influences beyond the current message, *e.g.*, the injection of a control command.

Here, a middlebox cannot always exclusively rely on the final receiver to authenticate a message. In some cases, *optimistic processing* [59, 63] (*i.e.*, the idea of processing a likely genuine message with the knowledge that maliciousness is guaranteed to be detected within a short time span) still allows offloading authentication to the final receiver for reliable connections (*e.g.*, TCP). But at the very least, middleboxes with side effects that communicate over lossy channels must authenticate the data they process themselves. Otherwise, an attacker could manipulate a message before a middlebox processes it and prevent it from being received at its final destination, *e.g.*, by jamming a wireless channel.

In cases where *immediate* verification of authenticity is required (*e.g.*, for irreversible critical decisions), MADTLS enables such middleboxes to *self-verify* authenticity by specifically adding an additional

16-byte authentication tag t_j for the middlebox j. This tag t_j is computed over the subset of partial tags σ relevant to the middlebox's access rights, such that no additional cryptographic processing is necessary. Formally, it is computed as follows:

$$t_j = \bigoplus_{i \in \mathbb{S}_j^{\text{read}} \cup \mathbb{S}_j^{\text{write}}} \left(\sigma_{k^{\text{read}}_{\delta(i),0}}(C[i]) \right) \oplus \bigoplus_{i \in \mathbb{S}_j^{\text{write}}} \left(\sigma_{k^{\text{write}}_{\delta(i),0}}(C[i]) \right)$$

Like the main authentication tag t, these tags t_j are modified by preceeding middleboxes. These modifications are, however, restricted to the subset of contexts both middleboxes have access to, which is communicated to the middleboxes during the handshake. For write access, a middlebox k would modify the tag t_j as follows:

$$t_j \overset{\oplus}{=} \bigoplus_{i \in \mathbb{S}_j^{\text{write}} \cap \mathbb{S}_k^{\text{write}}} \Big(\sigma_{\varphi(k^{\text{read}}_{\delta(i),k})}(C[i]) \oplus \sigma_{k^{\text{read}}_{\delta(i),k}}(C'[i]) \oplus \sigma_{\varphi(k^{\text{write}}_{\delta(i),k})}(C[i]) \oplus \sigma_{k^{\text{write}}_{\delta(i),k}}(C'[i]) \Big)$$

Analogously, the intermediary middlebox k only updates the read tags if it and middlebox j have access to a context:

$$t_j \overset{\oplus}{=} \bigoplus_{i \in \mathbb{S}_j^{\text{read}} \cap (\mathbb{S}_k^{\text{read}} \cup \mathbb{S}_k^{\text{write}})} \left(\sigma_{\varphi(k^{\text{read}}_{\delta(i),k})}(C[i]) \oplus \sigma_{k^{\text{read}}_{\delta(i),k}}(C'[i]) \right)$$

Finally, upon reception of the message by middlebox j, this middlebox can verify the authenticity of the data it has access to by recomputing t_j and comparing it to the transmitted tag. When forwarding the message to the final receiver, middlebox j removes the tag t_j as it is no longer needed. After learning about self-verifying middleboxes, we can now look at how MADTLS enables limited message injection as required by some industrial use cases.

5.6 Limited Message Injection Capabilities

So far, MADTLS allows authorized middleboxes to read and write to well-defined segments of transmitted messages. Still, certain industrial use cases for middleboxes additionally require capabilities to actively inject new messages, *e.g.*, to enable low latency control of robot arms (cf. Section 2.2). However, giving such middleboxes the ability to inject *arbitrary* messages breaks the least-privilege principle as a middlebox could take control beyond what is necessary for its dedicated task. Consequently, MADTLS needs to enforce *limited injection capabilities* where middleboxes can only inject those messages relevant to its intended functionality.

MADTLS realizes such limited injection capabilities through the use of pre-defined *message templates*. In essence, a template is a message with dedicated placeholders that is authenticated by the endpoint and sent to the middlebox in advance. The middlebox then has restricted write access to the placeholder segments of this message before it is forwarded (*e.g.*, to only be able to transmit specific control commands).

However, DTLS uses nonces that are explicitly transmitted to prevent replay attacks. Either way, for messages generated via message templates, we must ensure that *(1)* the used nonce is unique and *(2)* the receiver knows and accepts the nonce. In DTLS, nonces are a combination of a sequence number and an epoch. To not interfere with this procedure, MADTLS defines a new content type (`0x1F`) to mark middlebox-injected messages. These messages use the same encryption and authentication keys as normal messages but start at any unique epoch in the future. This epoch should be chosen according to how many injected messages are expected in relation to normal messages and how many message-injecting middleboxes exist in a given communication session.

While a middlebox with the mere task of sending emergency stops in critical situations only needs the ability to send a few messages, a middlebox responsible for low-latency control adjustments continuously injects a significant number of messages. As the sequence numbers of injected and regularly transmitted messages (from the original sender) are decoupled, the impersonated endpoint can asynchronously provide multiple authentication tags in advance for increasing sequence numbers of the same message template without having to retransmit the template itself. Hence, MADTLS achieves the same fine-grained access control over injected messages as for read and write access to existing messages.

6 THE MADTLS HANDSHAKE PROTOCOL

MADTLS requires several keys to enable fine-grained access control for middleboxes. These keys can be pre-configured or distributed ad-hoc by a trusted party. In many cases, it is, however, beneficial if the involved parties can agree upon these keys by themselves. Therefore, we adapt the DTLS 1.2 handshake to additionally exchange the keys required for MADTLS to the endpoints as well as all involved middleboxes. For simplicity, we assume that all communication entities have pre-established shared secrets, which is often reasonable as all entities are known in advanace and managed by a single operator in industrial networks. To enable certificate-based authentication of all entities, we can employ the `ServerKeyExchange` message proposed in step 3 of the mcTLS handshake [46], which we, however, do not require due to our pre-shared keys. Figure 7 highlights the necessary additions to the DTLS 1.2 handshake in blue. In the following, we discuss these changes step by step.

`ClientHello`. The `ClientHello` announces the desire to establish a new communication session and proposes a set of cipher suites. The `ClientHello` contains a pre-exchanged cookie to thwart DoS attacks as per the DTLS 1.2 standard, which we do not change. One important change is that the `ClientHello` message passes through all middleboxes, which are thus informed about the new session and may remove cipher suites from the announced list. We extend it by the information concerning middleboxes and their access rights. As additional information, MADTLS first appends a list of middleboxes identified by their IP address. Then, the `contexts` field defines all contexts needed by that session, *i.e.*, combinations of read and write access for the different middleboxes. Finally, the client appends a list of possible message templates. First, the number of templates is announced, followed by a null-byte terminated enumeration of `segmentation info` fields, as introduced in Figure 5. If the receiver accepts the request and supports a requested cipher suite, it responds with a `ServerHello` message.

`ServerHello`. The receiver responds with a `ServerHello` handshake message, agrees on a selected cipher suite, and replays the MADTLS-specific *contexts* and *template* fields. These fields must be replayed as the server may add more contexts or templates to it.

`ServerHelloDone`. The `ServerHelloDone` does not require any modification as it only signals the end of the `ServerHello`.

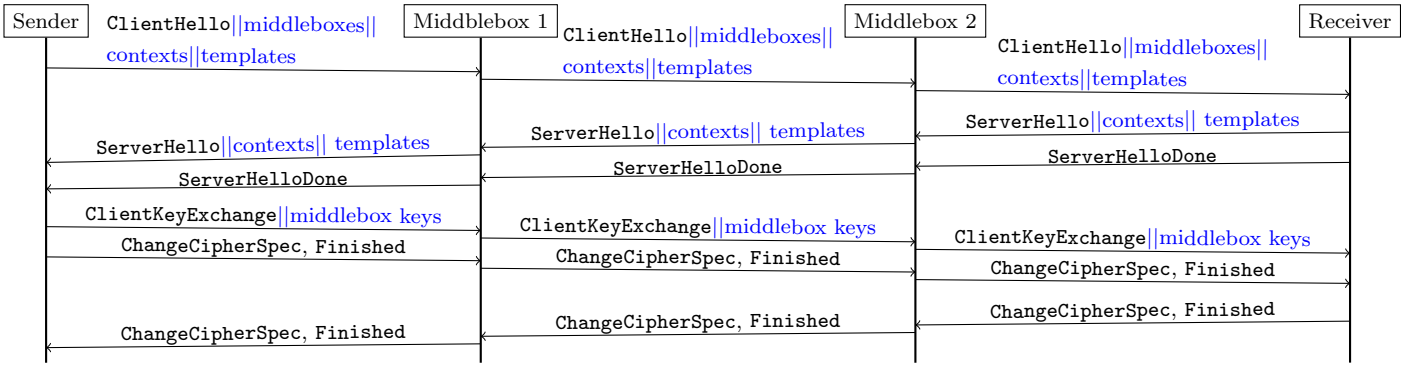


Figure 7: For MADTLS, we extend the DTLS 1.2 handshake and add additional data to select messages (highlighted in blue).

`ClientKeyExchange`. The client then shares the final information the server needs to derive the symmetric keys used in this session and shares all middlebox keys k^{read} and k^{write} with the respective involved middleboxes through a `ClientKeyExchange` message. For this key distribution, the sender first computes a key to encrypt and authenticate data to the respective middleboxes. Based on $\text{secret}_{s,m}$ shared between the sender s and a middlebox m, these key distribution keys k^{kd} are derived as:

$$k_m^{kd} = \text{KDF}(\text{secret}_{s,m}, \text{nonce}),$$

where the nonce is derived from the standard DTLS handshake. Then, the context encryption keys k^{enc} are derived independently of any secrets shared with a middlebox, as

$$\text{KDF}(\text{secrets}, r, \text{nonce}, \text{context}, \text{'encrypt'})$$

from the secret shared between sender s and receiver r, the nonce, an identifier of the respective context and a unique string. Finally, the context authentication keys k^{read} and k^{write} are derived as

$$\text{KDF}(\text{secret}_{s,r}, \text{nonce}, \text{middlebox}, \text{context}, \{\text{'read'}, \text{'write'}\})$$

by including an identifier of the targeted middlebox and different strings for read or write keys. To ensure that a middlebox only gains access to its keys (and the respective preceding keys as given by $\varphi(\cdot)$), the client encrypts each middlebox's context keys with the respective key distribution key k_m^{kd} before appending them to the `ClientKeyExchange` message. Thus, if middlebox 2 has read access to context 1 and write access to context 3, the sender appends $\text{enc}_{k_{s,2}^{kd}}(k_{2,1}^{\text{read}} || k_{2,3}^{\text{read}} || k_{2,3}^{\text{write}} || \varphi(k_{2,1}^{\text{read}}) || \varphi(k_{2,3}^{\text{read}}) || \varphi(k_{2,3}^{\text{write}}))$ to the `ClientKeyExchange` message. The middleboxes can then decrypt these respective keys as the `ClientKeyExchange` passes. The subsequently transmitted `ChangeCipherSpec` and `Finished` messages are not changed for MADTLS.

Final Server Messages. In DTLS 1.2, the server sends a final copy of `ChangeCipherSpec` and `Finished` messages to conclude the handshake. This procedure is not changed by MADTLS as the receiver can compute and verify all key distribution keys that the sender transmitted in the `ClientKeyExchange` message. Overall, a MADTLS session can be efficiently established, and we now look at its performance in real-world scenarios.

7 PERFORMANCE EVALUATION

MADTLS fulfills all functional requirements expected of a middlebox-aware security protocol for industrial networks (*cf.* Section 3.2). To evaluate if its performance is suitable even for resource-constrained devices, we implemented a prototype for Contiki-NG 4.8, a popular operating system for IoT devices, by extending the *tinydtls* library.

7.1 MADTLS vs. the Current State-of-the-Art

First, we compare the processing latency of MADTLS to related approaches. These approaches consist of the tinydtls DTLS 1.2 implementation (which offers no middlebox awareness) and a custom mcTLS implementation[1] adapted to DTLS to avoid TCP overhead. For all protocols, we send payloads of lengths increasing from 1 to 256 bytes. MADTLS uses a single write context to emulate the same functionality as mcTLS. Our measurements run on a Zolertia RE-Mote (Cortex M3 @ 32 MHz, 32-bit CPU), a common device to represent the constraints of industrial hardware [50, 59, 60].

Figure 9 plots the processing time against the length of the transmitted payload for the different protocols. All protocols require more processing as the payload increases. However, these increases are discrete, marginally rising by about 0.1 ms whenever the payload fills up a new AES block (all 16 bytes), and more substantially jumping by about 0.4 ms when sha256 blocks of the HMAC computation are filled up (all 64 bytes). Thus, MADTLS is efficient even for large messages and even achieves a noticable performance gain against mcTLS which neither offers the same fine-grained data access as MADTLS nor ensures path integrity.

Beyond latency, reducing bandwidth usage is also imperative for MADTLS. Here, the DTLS 1.2 record protocol carries a header overhead (including 16 bytes tags) of 30 bytes while mcTLS's overhead is 63 bytes. Meanwhile, MADTLS only adds 1 byte to DTLS 1.2 for the `template id` that defines the message structure. Even for multiple contexts, MADTLS does not require more space.

MADTLS is thus attractive performance-wise for industrial networks, even if no fine-grained data access is needed. In the case of a single context, it saves over 22 % of processing latency (more for

[1] We could not source any implementations of the approaches discussed in Section 2.3 and therefore implement the closest competitor of MADTLS ourselves.

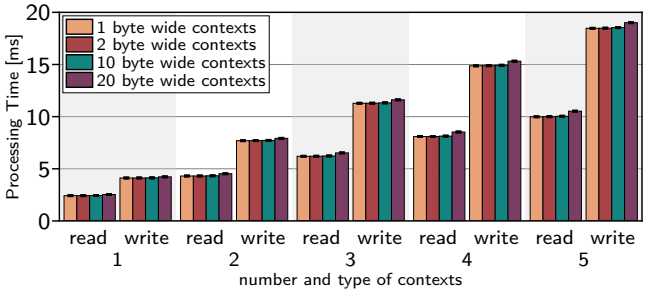


Figure 8: MADTLS 's performance scales linearly in the number of contexts, while their sizes only have marginal impact.

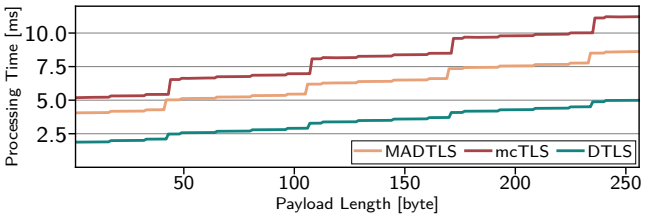


Figure 9: MADTLS performs better than mcTLS in comparable scenarios, where middlebox write access is given for selected messages. DTLS provides no such guarantees.

larger messages) and 32 bytes of header over mcTLS. Most importantly, MADTLS offers this performance while offering true least-privilege data access to middleboxes, ensuring path integrity, and allowing least privilege traffic injection, three crucial features not offered by mcTLS, or any other approach from related work.

Cipher Suite for Faster Processing. MADTLS performs adequately for many industrial applications. However, some applications require even faster processing in the sub-millisecond range. To assess MADTLS under these harsh requirements, we design a cipher suite to minimize processing latency based on recent work on preprocessed encryption and integrity protection. More concretely, we create a cipher suite based on antedated encryption [28] and BP-MAC [60]. Antedated encryption uses AES in counter mode and splits it into a precomputation phase for the computationally-intensive keystream generation and a fast online phase that only XORs a message with the cached keystream [28]. BP-MAC, in turn, achieves fast integrity protection for short messages by combining precomputed authentication tags for individual bits via a Carter-Wegman construction [60]. Using this cipher suite for MADTLS on our resource-constrained RE-Mote board, message encryption and authentication time reduces to 624 µs (compared to 3.975 ms) for a 5-byte message with a single write context spanning the entire message. Thus, with suitable cryptographic algorithms, MADTLS is apt for low-latency scenarios on constrained hardware.

7.2 Impact of the Size and Number of Contexts

We must also understand the impact of MADTLS' fine-grained access control on its performance. Therefore, we evaluate the impact of the number and size of contexts on the processing time. We continue using the Zolertia RE-Mote as evaluation platform and measure the encryption and authentication time of a 100-byte long packet. We add 1 to 5 contexts of sizes varying between 1 and 20 bytes to each message. We repeat our measurements 20 times and show the results, including 99%-confidence intervals, in Figure 8.

We observe a linear growth of processing times with the number of contexts and that write contexts require more processing than read contexts. This scaling is expected as it is proportional to the number of calls to HMAC-SHA256. While processing one 20-byte read context takes 2.42 ms, the same operation over a write context lasts 4.22 ms. This behavior can also be explained by the number of HMAC-SHA256 calls, as write contexts require two calls per context compared to the one required for read contexts. Meanwhile, the size of the contexts only has a negligible impact on the processing time as all contexts individually fit into a single SHA256 block.

Overall, MADTLS performs adequately even for scenarios requiring many contexts for different middleboxes with diverse functionalities. As contexts are shared if the same data is accessed by multiple middleboxes, even complex chains of middleboxes can operate in a least-privilege mode with a low number of contexts. Thus, even resource-constrained devices, commonly seen as endpoints in industrial networks, can employ MADTLS.

7.3 MADTLS Across Different Hardware Classes

Middleboxes are typically more powerful as they have to process many messages from different sources. Therefore, we use the Raspberry Pi Zero (which uses the same Arm 11 chip that the Netronome NFP-4000 Flow Processor uses for general-purpose processing) to evaluate the performance of SmartNIC-based middleboxes without further optimizations. Moreover, we evaluate MADTLS on an AMD EPYC 7551 server CPU. Thus, we can learn how MADTLS performs over a wide variety of CPU classes and architectures.

For our evaluation, we process 10, 50, 100, and 200 byte messages with the middlebox accessing 80 % of a message's data via a single read context. The processing entails the decryption of the accessed data and updating the authentication tag. We repeat all measurements 20 times and report on the throughput with 99%-confidence intervals in Figure 10. Across all processors, the throughput grows significantly with longer messages as the fixed per-message overhead mainly impacts small messages.

Our evaluation shows that if devices like the Zolertia RE-Mote were used as a middlebox, they would still achieve a throughput of over 66 kbit/s. The Raspberry Pi Zero more than doubles this throughput, with a throughput between 154 and 1877 kbit/s, depending on the message sizes. Expectedly, SmartNICs (as well as programmable switches) can, however, not rely on a relatively slow general-purpose processing core to achieve gigabit throughput. Still, their performance can be significantly optimized with device-specific implementations of cryptographic primitives [32, 62]. On the other hand, our AMD EPYC 7551 processor achieves a throughput between 20 and 249 Mbits/s on a single of its 32 cores.

7.4 MADTLS in the Real World

To verify MADTLS's utility in real-world use cases, we conduct two case studies. First, we implement a middlebox that translates between local coordinates of robot arms and global coordinates as proposed by Kunze *et al.* [34]. Here, MADTLS allows us to ensure that

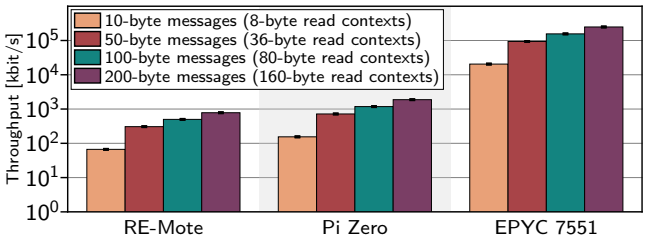


Figure 10: Throughput is limited on constrained devices but middleboxes can take advantage of more powerful hardware.

the middlebox only accesses the first six byte of a packet's payload where the x, y, and z coordinates are encoded in 16 bits each. The remaining payload, containing supplementary sensor readings (*e.g.*, timestamp, grip pressure, gripper rotation) in an additional 14 bytes, cannot be written to or read by the middlebox. This data layout, *e.g.*, corresponds to one observed in Modbus communications. Secondly, we use MADTLS to realize an IDS based on Snort rules. To this end, we use the 14 Quickdraw Snort rules [2] for industrial Modbus communication. Using MADTLS, we constrain the IDS' access to selectively reading 3, 5, or 6 bytes per Modbus frame, depending on the communication flow. Without restricting the IDS' capabilities, MADTLS thus blinds over 60 % of all bytes in the corresponding Modbus test traces [2].

8 LIMITATIONS OF MADTLS

MADTLS addresses many shortcomings of related work on secure middlebox-aware (industrial) communication. These achievements come, however, with a few drawbacks that must be considered before deploying MADTLS. First, MADTLS' handshake is significantly more complex than the standard DTLS 1.2 handshake. While it does not require additional round trips, more data must be exchanged between the entities. Per se, this is not a problem in industrial networks with relatively static connections since handshakes can be performed in advance during non-critical periods. However, the added complexity makes weaknesses in design and implementation more likely and may be restrictive in some scenarios.

Secondly, MADTLS needs more extensive key management and storage than simple end-to-end communication. While this drawback is inevitable when multiple entities with different access rights on a single communication channel are involved, it may impact embedded devices with limited storage capabilities. Also, the corresponding key exchange protocol, as exemplified in Section 6, becomes more error-prone in terms of design and implementation.

Thirdly, MADTLS offloads integrity verification to the final receiver of a message per default. However, it must be carefully considered whether middleboxes must additionally verify the integrity of processed data themselves. Still, MADTLS offers efficiently computed additional middlebox-specific authentication tags that require no additional cryptographic processing to append to a message.

Fourthly, while MADTLS is designed with performance in mind and even outperforms its closest competitor (mcTLS [46]), the added processing overhead is not negligible for resource-constrained devices in industrial scenarios. Fortunately, MADTLS's processing adds minimal jitter, such that a deterministic overhead can be considered when designing control algorithms where required [52].

Fifthly, MADTLS adopts DTLS 1.2 with our building blocks to bring middlebox-aware security to industrial communication. In reality, the industrial landscape and beyond also uses other security protocols (*e.g.*, TLS) that can and should not always be replaced with DTLS. While we see nothing preventing the adoption of other protocols with MADTLS's features, a concrete design must still be proposed to bring our advances to a wider variety of applications.

Sixthly, MADTLS is most efficient for predictable message structures as they are often found in the IIoT. Employing MADTLS on the traditional Internet would expose it to more dynamic content (*e.g.*, websites). While the explicit transmission of segmentation info enables such scenarios, the added bandwidth overhead must be considered. Moreover, privacy implications of metadata (*e.g.*, for templates) must be carefully considered in other scenarios, as devices no longer belong to a single operator in that case.

MADTLS thus mostly exhibits limitations outside the industrial domain. However, domain-specific adaptions can alleviate some of these constraints while benefiting from MADTLS's main contributions: Allowing least-privilege access control on a communication channel for middlebox processing.

9 CONCLUSION

This paper proposes MADTLS, a middlebox-aware enhancement of the DTLS protocol tailored explicitly to the IIoT. Hereby, MADTLS addresses multiple major limitations of the current state-of-the-art on middlebox-aware security that focuses heavily on the traditional Internet. Most importantly, MADTLS allows fine-grained access control for middleboxes on a bit-level and enables middleboxes to inject a restricted set of messages where this is desired (*e.g.*, for emergency shutdowns) while still operating more efficiently than mcTLS [46].

Specifically, MADTLS segments messages and assigns contexts (*i.e.*, read and write access rights) according to the middleboxes' needs. Each segment is encrypted and authenticated separately without expanding the packet. Middleboxes are permitted to read or write to specific segments and either verify integrity directly or defer this step to the final receiver for efficiency reasons. MADTLS processes packets in only a few milliseconds on heavily constrained hardware, while performance scales linearly with the number of contexts and message lengths. Meanwhile, MADTLS achieves up to 249 Mbit/s throughput on a single AMD EPYC 7551 core, enabling middleboxes to process many different communication streams.

MADTLS thus brings middlebox-aware security to the IIoT to solve the dilemma where middleboxes become increasingly popular and thus prevent secure end-to-end communication.

ACKNOWLEDGMENTS

We would like to thank René Glebke and our anonymous reviewers for their valuable feedback. This paper was supported by the EDA Cyber R&T project CERERE, funded by Italy and Germany. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2023 Internet of Production – 390621612. The authors are responsible for the contents of this work.

REFERENCES

[1] [n. d.]. Arrowhead, Ahead of the Future. http://www.arrowhead.eu. Last Accessed: 22-11-2023.

[2] [n. d.]. Quickdraw Snort Ruleset. https://github.com/digitalbond/Quickdraw-Snort/blob/master/modbus.rules. Last Accessed: 22-11-2023.

[3] [n. d.]. Snort. https://www.snort.org/. Last Accessed: 22-11-2023.

[4] Taehyun Ahn, Jiwon Kwak, and Seungjoo Kim. 2023. mdTLS: How to Make middlebox-aware TLS more efficient?. In *Proceedings of the International Conference on Information Security and Cryptology (ICISC'23)*.

[5] Tejasvi Alladi, Vinay Chamola, and Sherali Zeadally. 2020. Industrial Control Systems: Cyberattack Trends and Countermeasures. *Computer Communications* 155 (2020).

[6] Mihir Bellare, Roch Guérin, and Phillip Rogaway. 1995. XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions. In *15th Annual International Cryptology Conference (Crypto'95)*.

[7] Dan Boneh and Victor Shoup. 2020. A Graduate Course in Applied Cryptography.

[8] Pietro Borrello, Andreas Kogler, Martin Schwarzl, Moritz Lipp, Daniel Gruss, and Michael Schwarz. 2022. ÆPIC Leak: Architecturally Leaking Uninitialized Data from the Microarchitecture. In *USENIX Security Symposium*.

[9] Sébastien Canard, Aïda Diop, Nizar Kheir, Marie Paindavoine, and Mohamed Sabt. 2017. BlindIDS: Market-Compliant and Privacy-Friendly Intrusion Detection System over Encrypted Traffic. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (AsiaCCS'17)*.

[10] B. Carpenter. 1996. *Architectural Principles of the Internet.* RFC 1958. IETF.

[11] B. Carpenter. 2000. *Internet Transparency.* RFC 2775. IETF.

[12] B. Carpenter and B. Liu. 2020. *Limited Domains and Internet Protocols.* RFC 8799. IETF.

[13] Marco Caselli, Emmanuele Zambon, and Frank Kargl. 2015. Sequence-aware Intrusion Detection in Industrial Control Systems. In *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*.

[14] Marco Caselli, Emmanuele Zambon, Jonathan Petit, and Frank Kargl. 2015. Modeling Message Sequences for Intrusion Detection in Industrial Control Systems. In *Proceedings of the Internation Conference on Critical Infrastructure Protection (ICCIP 15)*.

[15] Fabricio E Rodriguez Cesen, Levente Csikor, Carlos Recalde, Christian Esteve Rothenberg, and Gergely Pongrácz. 2020. Towards Low Latency Industrial Robot Control in Programmable Data Planes. In *Conference on Network Softwarization (NetSoft'20)*.

[16] Markus Dahlmanns, Johannes Lohmöller, Jan Pennekamp, Jörn Bodenhausen, Klaus Wehrle, and Martin Henze. 2022. 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 (AsiaCCS'22)*.

[17] Xavier de Carné de Carnavalet and Paul C. van Oorschot. 2023. A Survey and Analysis of TLS Interception Mechanisms and Motivations: Exploring How End-to-End TLS is Made "End-to-Me" for Web Traffic. *Comput. Surveys* 55, 13s (2023).

[18] D. Dolev and A. Yao. 1983. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory* 29, 2 (1983).

[19] Huayi Duan, Cong Wang, Xingliang Yuan, Yajin Zhou, Qian Wang, and Kui Ren. 2019. LightBox: Full-Stack Protected Stateful Middlebox at Lightning Speed. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS'19)*.

[20] Ertem Esiner, Utku Tefek, Daisuke Mashima, Binbin Chen, Zbigniew Kalbarczyk, and David M Nicol. 2023. Message Authentication and Provenance Verification for Industrial Control Systems. *ACM Transactions on Cyber-Physical Systems* 7, 4 (2023).

[21] Marc Fischlin. 2023. Stealth Key Exchange and Confined Access to the Record Protocol Data in TLS 1.3. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS'23)*.

[22] Brendan Galloway and Gerhard P Hancke. 2012. Introduction to Industrial Control Networks. *IEEE Communications Surveys & Tutorials* 15, 2 (2012).

[23] Paul Grubbs, Arasu Arun, Ye Zhang, Joseph Bonneau, and Michael Walfish. 2022. Zero-Knowledge Middleboxes. In *USENIX Security Symposium*.

[24] Csaba Györgyi, Károly Kecskeméti, Péter Vörös, Géza Szabó, and Sándor Laki. 2021. In-network Solution for Network Traffic Reduction in Industrial Data Communication. In *International Conference on Network Softwarization (NetSoft'21)*.

[25] Csaba Györgyi, Péter Vörös, Károly Kecskeméti, Géza Szabó, and Sándor Laki. 2023. Adaptive Network Traffic Reduction on the Fly With Programmable Data Planes. *IEEE Access* 11 (2023).

[26] Juhyeng Han, Seongmin Kim, Jaehyeong Ha, and Dongsu Han. 2017. SGX-Box: Enabling Visibility on Encrypted Traffic using a Secure Middlebox Module. In *Proceedings of the First Asia-Pacific Workshop on Networking*.

[27] Kevin E. Hemsley and Dr. Ronald E. Fisher. 2018. History of Industrial Control System Cyber Incidents. (2018). https://doi.org/10.2172/1505628

[28] Jens Hiller, Martin Henze, Martin Serror, Eric Wagner, Jan Niklas Richter, and Klaus Wehrle. 2018. Secure Low Latency Communication for Constrained Industrial IoT Scenarios. In *Conference on Local Computer Networks (LCN'18)*.

[29] Jonathan Katz and Andrew Y Lindell. 2008. Aggregate Message Authentication Codes. In *Cryptographers' Track at the RSA Conference*.

[30] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. 2000. The Click Modular Router. *Transactions on Computer Systems* 18, 3 (2000).

[31] Thomas Kohler, Ruben Mayer, Frank Dürr, Marius Maaß, Sukanya Bhowmik, and Kurt Rothermel. 2018. P4CEP: Towards In-Network Complex Event Processing. In *Proceedings of the Morning Workshop on In-Network Computing*.

[32] Shaguftha Zuveria Kottur, Krishna Kadiyala, Praveen Tammana, and Rinku Shah. 2022. Implementing ChaCha Based Crypto Primitives on Programmable SmartNICs. In *Proceedings of the ACM SIGCOMM Workshop on Formal Foundations and Security of Programmable Network Infrastructures*.

[33] Ralf Kundel, Fridolin Siegmund, Jeremias Blendin, Amr Rizk, and Boris Koldehofe. 2020. P4STA: High Performance Packet Timestamping with Programmable Packet Processors. In *IEEE/IFIP Network Operations and Management Symposium (NOMS'20)*.

[34] Ike Kunze, René Glebke, Jan Scheiper, Matthias Bodenbenner, Robert H Schmitt, and Klaus Wehrle. 2021. Investigating the Applicability of In-Network Computing to Industrial Scenarios. In *International Conference on Industrial Cyber-Physical Systems (ICPS'21)*.

[35] Ike Kunze, Philipp Niemietz, Liam Tirpitz, René Glebke, Daniel Trauth, Thomas Bergs, and Klaus Wehrle. 2021. Detecting Out-Of-Control Sensor Signals in Sheet Metal Forming Using In-Network Computing. In *Proceedings of the 2021 IEEE International Symposium on Industrial Electronics (ISIE'21)*.

[36] Chang Lan, Justine Sherry, Raluca Ada Popa, Sylvia Ratnasamy, and Zhi Liu. 2016. Embark: Securely Outsourcing Middleboxes to the Cloud. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI'16)*.

[37] Hyunwoo Lee, Zach Smith, Junghwan Lim, Gyeongjae Choi, Selin Chun, Taejoong Chung, and Ted Taekyoung Kwon. 2019. maTLS: How to Make TLS middlebox-aware?. In *Network and Distributed System Security Symposium (NDSS'19)*.

[38] Jie Li, Rongmao Chen, Jinshu Su, Xinyi Huang, and Xiaofeng Wang. 2019. ME-TLS: Middlebox-enhanced TLS for Internet-of-Things Devices. *IEEE Internet of Things Journal* 7, 2 (2019).

[39] Xiaozhou Li, Raghav Sethi, Michael Kaminsky, David G Andersen, and Michael J Freedman. 2016. Be Fast, Cheap and in Control with SwitchKV. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI'16)*.

[40] Athanasios Liatifis, Panagiotis Sarigiannidis, Vasileios Argyriou, and Thomas Lagkas. 2023. Advancing SDN from OpenFlow to P4: A Survey. *Comput. Surveys* 55, 9 (2023).

[41] Chih-Yuan Lin and Simin Nadjm-Tehrani. 2019. Timing Patterns and Correlations in Spontaneous SCADA Traffic for Anomaly Detection. In *Proceedings of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID'19)*.

[42] Michele Luvisotto, Zhibo Pang, and Dacfey Dzung. 2016. Ultra High Performance Wireless Control for Critical Applications: Challenges and Directions. *IEEE Transactions on Industrial Informatics* 13, 3 (2016).

[43] Tianle Mai, Haipeng Yao, Song Guo, and Yunjie Liu. 2020. In-Network Computing Powered Mobile Edge: Toward High Performance Industrial IoT. *IEEE Network* 35, 1 (2020).

[44] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. 2020. Plundervolt: Software-based Fault Injection Attacks against Intel SGX. In *Proceedings of the 41st IEEE Symposium on Security and Privacy (S&P'20)*.

[45] David Naylor, Richard Li, Christos Gkantsidis, Thomas Karagiannis, and Peter Steenkiste. 2017. And Then There Were More: Secure Communication for More Than Two Parties. In *International Conference on emerging Networking EXperiments and Technologies (CoNEXT'17)*.

[46] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R López, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter Steenkiste. 2015. Multi-Context TLS (mcTLS) Enabling Secure In-Network Functionality in TLS. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM'15)*.

[47] Jianting Ning, Geong Sen Poh, Jia-Ch'ng Loh, Jason Chia, and Ee-Chien Chang. 2019. PrivDPI: Privacy-Preserving Encrypted Traffic Inspection with Reusable Obfuscated Rules. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS'19)*.

[48] Gennady Pekhimenko, Chuanxiong Guo, Myeongjae Jeon, Peng Huang, and Lidong Zhou. 2018. TerseCades: Efficient Data Compression in Stream Processing. In *USENIX Annual Technical Conference (ATC'18)*.

[49] Rishabh Poddar, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. 2018. SafeBricks: Shielding Network Functions in the Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*.

[50] Francesca Righetti, Carlo Vallati, Daniela Comola, and Giuseppe Anastasi. 2019. Performance Measurements of IEEE 802.15. 4g Wireless Networks. In *International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM'19)*.

[51] Fabricio Rodriguez, Christian Esteve Rothenberg, and Gergely Pongrácz. 2019. In-Network P4-based Low Latency Robot Arm Control. In *Proceedings of the 15th International Conference on emerging Networking EXperiments and Technologies (CoNEXT'22)*.

[52] Jan Rüth, René Glebke, Klaus Wehrle, Vedad Causevic, and Sandra Hirche. 2018. Towards In-Network Industrial Feedback Control. In *Proceedings of the Morning Workshop on In-Network Computing*.
[53] Amedeo Sapio, Ibrahim Abdelaziz, Abdulla Aldilaijan, Marco Canini, and Panos Kalnis. 2017. In-Network Computation Is a Dumb Idea Whose Time Has Come. In *ACM Workshop on Hot Topics in Networks*.
[54] Syahril Ramadhan Saufi, Zair Asrar Bin Ahmad, Mohd Salman Leong, and Meng Hee Lim. 2019. Challenges and Opportunities of Deep Learning Models for Machinery Fault Detection and Diagnosis: A Review. *IEEE Access* 7 (2019).
[55] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. 2015. Blindbox: Deep Packet Inspection over Encrypted Traffic. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM'15)*.
[56] Bohdan Trach, Alfred Krohmer, Franz Gregor, Sergei Arnautov, Pramod Bhatotia, and Christof Fetzer. 2018. Shieldbox: Secure Middleboxes Using Shielded Execution. In *Proceedings of the Symposium on SDN Research*.
[57] Mostafa Uddin, Sarit Mukherjee, Hyunseok Chang, and TV Lakshman. 2017. SDN-based service automation for IoT. In *Proceedings of the 25th International Conference on Network Protocols (ICNP'17)*.
[58] Jonathan Vestin, Andreas Kassler, Sándor Laki, and Gergely Pongrácz. 2020. Toward In-Network Event Detection and Filtering for Publish/Subscribe Communication Using Programmable Data Planes. *IEEE Transactions on Network and Service Management* 18, 1 (2020).
[59] Eric Wagner, Jan Bauer, and Martin Henze. 2022. Take a Bite of the Reality Sandwich: Revisiting the Security of Progressive Message Authentication Codes. In *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec'22)*.
[60] Eric Wagner, Martin Serror, Klaus Wehrle, and Martin Henze. 2022. BP-MAC: Fast Authentication for Short Messages. In *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec'22)*.
[61] Konrad Wolsing, Eric Wagner, Antoine Saillard, and Martin Henze. 2022. IPAL: Breaking Up Silos of Protocol-Dependent and Domain-Specific Industrial Intrusion Detection Systems. In *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses (RAID'22)*.
[62] Sophia Yoo and Xiaoqi Chen. 2021. Secure Keyed Hashing on Programmable Switches. In *Proceedings of the ACM SIGCOMM Workshop on Secure Programmable Network Infrastructure*.
[63] Collin Zhang, Zachary DeStefano, Arasu Arun, Joseph Bonneau, Paul Grubbs, and Michael Walfish. 2023. Zombie: Middleboxes that Don't Snoop. In *IEEE Symposium on Security and Privacy (S&P'23)*.

APPENDIX

A SOUNDNESS OF MADTLS

We first prove the soundness of MADTLS's authentication scheme, *i.e.*, that without malicious manipulation all valid tags are accepted. First, we look at the case when no middlebox is included.

$$\begin{aligned} t^* &= \bigoplus_{0\le i<n} \left(\sigma_{\varphi(k^{\text{read}}_{\delta(i),j-1})}(X[i]) \oplus \sigma_{\varphi(k^{\text{write}}_{\delta(i),j-1})}(X[i])\right) \\ &= \bigoplus_{0\le i<n} \left(\sigma_{k^{\text{read}}_{\delta(i),0}}(X[i]) \oplus \sigma_{k^{\text{write}}_{\delta(i),0}}(X[i])\right) \\ &\overset{!}{=} t \end{aligned}$$

Then, we prove by induction that the inclusion of a reading middlebox k as last hop before the intended receiver does transform a valid tag t' into a valid final tag t^*.

$$\begin{aligned} t' &= \bigoplus_{0\le i<n} \left(\sigma_{\varphi(k^{\text{read}}_{\delta(i),k})}(X[i]) \oplus \sigma_{\varphi(k^{\text{write}}_{\delta(i),k})}(X[i])\right) \\ &\overset{\text{read}}{=} \bigoplus_{0\le i<n} \left(\sigma_{\varphi(k^{\text{read}}_{\delta(i),k})}(X[i]) \oplus \sigma_{\varphi(k^{\text{write}}_{\delta(i),k})}(X[i])\right) \oplus \\ &\qquad \bigoplus_{i\in\mathbb{S}^{\text{read}}_k} \left(\sigma_{\varphi(k^{\text{read}}_{\delta(i),k})}(X[i]) \oplus \sigma_{k^{\text{read}}_{\delta(i),k}}(X[i])\right) \\ &= \bigoplus_{i\notin\mathbb{S}^{\text{read}}_k} \left(\sigma_{\varphi(k^{\text{read}}_{\delta(i),k})}(X[i]) \oplus \sigma_{\varphi(k^{\text{write}}_{\delta(i),k})}(X[i])\right) \oplus \\ &\qquad \bigoplus_{i\in\mathbb{S}^{\text{read}}_k} \left(\sigma_{k^{\text{read}}_{\delta(i),k}}(X[i]) \oplus \sigma_{\varphi(k^{\text{write}}_{\delta(i),k})}(X[i])\right) \\ &= \bigoplus_{0\le i<n} \left(\sigma_{\varphi(k^{\text{read}}_{\delta(i),j-1})}(X[i]) \oplus \sigma_{\varphi(k^{\text{write}}_{\delta(i),j-1})}(X[i])\right) \\ &\overset{!}{=} t^* \end{aligned}$$

Finally, we do the same inductive proof for a writing middlebox.

$$\begin{aligned} t' &= \bigoplus_{0\le i<n} \left(\sigma_{\varphi(k^{\text{read}}_{\delta(i),k})}(X[i]) \oplus \sigma_{\varphi(k^{\text{write}}_{\delta(i),k})}(X[i])\right) \\ &\overset{\text{write}}{=} \bigoplus_{0\le i<n} \left(\sigma_{\varphi(k^{\text{read}}_{\delta(i),k})}(X[i]) \oplus \sigma_{\varphi(k^{\text{write}}_{\delta(i),k})}(X[i])\right) \oplus \\ &\qquad \bigoplus_{i\in\mathbb{S}^{\text{write}}_j} \Big(\sigma_{\varphi(k^{\text{read}}_{\delta(i),j})}(X[i]) \oplus \sigma_{k^{\text{read}}_{\delta(i),j}}(X'[i]) \oplus \\ &\qquad\qquad \sigma_{\varphi(k^{\text{write}}_{\delta(i),j})}(X[i]) \oplus \sigma_{k^{\text{write}}_{\delta(i),j}}(X'[i])\Big) \\ &= \bigoplus_{i\notin\mathbb{S}^{\text{write}}_k} \left(\sigma_{\varphi(k^{\text{read}}_{\delta(i),k})}(X'[i]) \oplus \sigma_{\varphi(k^{\text{write}}_{\delta(i),k})}(X'[i])\right) \oplus \\ &\qquad \bigoplus_{i\in\mathbb{S}^{\text{write}}_k} \left(\sigma_{k^{\text{read}}_{\delta(i),k}}(X'[i]) \oplus \sigma_{k^{\text{write}}_{\delta(i),k}}(X'[i])\right) \\ &= \bigoplus_{0\le i<n} \left(\sigma_{\varphi(k^{\text{read}}_{\delta(i),j-1})}(X'[i]) \oplus \sigma_{\varphi(k^{\text{write}}_{\delta(i),j-1})}(X'[i])\right) \\ &\overset{!}{=} t^* \end{aligned}$$

B SECURITY OF MADTLS

Besides the soundness of MADTLS's authentication scheme, its security is also crucial. To prove its security, we first look at the security definition of traditional Message Authentication Code (MAC) schemes to show that MADTLS cannot be attacked by outsiders. Afterwards, we also prove that an insider, *i.e.*, a middlebox authorized to read or write some part of a packet, cannot manipulate a packet beyond what it is authorized to do.

B.1 Security of Traditional MAC Schemes

The security of a MAC scheme $\Sigma = (\text{Sig}, \text{Vrfy})$ over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$ is typically defined through a game between a challenger and an adversary $\mathcal{A}$ [7].

Attack Game 1.

- The challenger randomly picks a key k from $\mathcal{K}$.
- The adversary queries an oracle with a message m_i for a valid tag t_i, *i.e.*, t_i such that $\text{Vrfy}_k(m_i, t_i)$ returns `accept`. Denote $\mathbb{M}$ the set of all m_i queried by $\mathcal{A}$.
- Eventually, $\mathcal{A}$ outputs a candidate forgery $(m, t) \in \mathbb{M} \times \mathcal{T}$, with $m \notin \mathbb{M}$.

$\mathcal{A}$ wins the above game if $\text{Vrfy}_k(m, t)$ returns `accept`. We define $\mathcal{A}$'s advantage with respect to Σ, denoted as $\text{Adv}_{\mathcal{A}}[\Sigma]$, as the probability that $\mathcal{A}$ wins the game. A MAC scheme is considered secure if the advantage of all efficient adversaries $\mathcal{A}$ with respect to Σ is negligable, *i.e.*, $\text{Adv}_{\mathcal{A}}[\Sigma] = Pr[\mathcal{A} \text{ wins Attack Game 1 for } \Sigma] < \epsilon$.

B.2 MADTLS is Secure Against Outsiders

Our proof is built upon the fact that $\text{Adv}_{\mathcal{B}}[\Sigma^{\oplus}] \leq \text{Adv}_{\mathcal{A}}[\Sigma]$, where a tag t by $\Sigma^{\oplus}$ is computed as $t = \Sigma(X[1]) \oplus \cdots \oplus \Sigma(X[n])$, with $m = X[1]||\ldots||X[n]$. We first adapt Attack Game 1 to prove the security of a MAC scheme that allows the manipulation (of selected parts) of the message by authorized middleboxes.

Attack Game 2.

- The challenger randomly picks a key k from $\mathcal{K}$.
- The adversary queries an oracle with a segment $X_i[\cdot]$ for a valid partial tag $t_i[\cdot]$, *i.e.*, $t_i[\cdot]$ such that $\text{Vrfy}_k^{\Sigma}(X_i[\cdot], t_i[\cdot])$ returns `accept`. Denote $\mathbb{M}$ the set of all m_i that can be composed of partial messages queried by $\mathcal{A}$.
- Eventually, $\mathcal{A}$ outputs a candidate forgery $(m, t) \in \mathcal{M} \times \mathcal{T}$, with $m \notin \mathbb{M}$.

In Attack Game 2, the adversary is strictly more powerful than in Attack Game 1, as they can query partial authentication tags, but still (over multiple queries) learn the tag over every specific message. Thus, MADTLS is secure against outsider attacks if an adversary $\mathcal{A}$ cannot win Attack Game 2 with a non-negligible probability.

Still, the advantage of $\mathcal{A}$ to win Attack Game 2 for $\Sigma^{\oplus}$ remains negligible: For every $m \notin \mathbb{M}$, there exists at least one m_i for which t_i was not queried by $\mathcal{A}$. Thus, $\text{Adv}_{\mathcal{B}}[\Sigma^{\oplus}]$ can be at most $\text{Adv}_{\mathcal{A}}[\Sigma]$, as otherwise $\mathcal{A}$ could learn t_i for $X_i[\cdot]$ which were not queried, *i.e.*, $\text{Adv}_{\mathcal{B}}[\Sigma^{\oplus}] \leq \text{Adv}_{\mathcal{A}}[\Sigma]$. Consequently, MADTLS's authentication scheme is secure as long as the underlying MAC scheme is secure and the tags t_i computed for different message segments $X_i[\cdot]$ are independent.

B.3 MADTLS is Secure Against Insiders

We established thus far that only authorized writers can alter (segments of) messages such that they are successfully verified by the receiver. However, we have not considered the possibility of ephemeral changes by malicious actors, *i.e.*, the temporary altering of a message for only one or a few middleboxes' processing, before the message is altered back to a message accepted by the final receiver. To validate that such an attack is not possible, we have to take a deeper look at how a message's authentication tag changes over time.

When middlebox j accesses a segment, it alters the authentication tag in a deterministic way: $t \oplus \sigma_{\varphi(k_{\_j})}(X[\cdot]) \oplus \sigma_{k_{\_j}}(X[\cdot])$. The partial tags from the last accessing entity are first removed from the aggregated tag, before new partial tags is added by the current entity. This process might be done once or twice, depending on whether the accessing middlebox is only reading or also writing.

Consequently, if an attacker maliciously changes the message segment $X'[\cdot]$, they must be able to compute $\sigma_{\varphi(k_{\_j})}(X'[\cdot])$ and $\sigma_{k_{\_j}}(X'[\cdot])$ in order to remove the consequences of their attack from the authentication tag. It is, however, precisely the middlebox j doing these computations that has access to the required keys (besides the endpoints of the communication channel). The next middleboxes accessing that specific segment may compute $\sigma_{k_{\_j}}(X'[\cdot])$ for the attacker if the message has not been changed back yet. However, then $\sigma_{\varphi^{-1}(k_{\_j})}(X'[\cdot])$ needs to be intercepted by the attacker. In the end, the attacker needs to compute two partial authentication tags to ephemerally change a message in an unauthorized way. Thus, the collusion of at least two middleboxes having read access to a specific message segment is required for an attack that introduces ephemeral changes. Hence, no single actor, external or internal, can attack MADTLS's authentication scheme.