Problems and New Approaches for Crypto-Agility in Operational Technology

Tobias Frauenschläger, Jürgen Mottok

Problems and New Approaches for Crypto-Agility in Operational Technology

Tobias Frauenschläger and Jürgen Mottok
Laboratory for Safe and Secure Systems (LaS³)
Technical University of Applied Sciences Regensburg
93053 Regensburg, Germany
{tobias.frauenschlaeger, juergen.mottok}@oth-regensburg.de

***Abstract*—In recent years, cybersecurity has also become relevant for Operational Technology (OT). Critical systems like industrial automation systems or transportation systems are faced with new threats, and therefore require the implementation of thorough security measures. Regulations further mandate the deployment and regular verification of these security measures. However, OT systems differ from well-known systems of classic Information Technology (IT), such as mission times spanning decades, infrequent updates only during on-site maintenance, or diverse devices with varying support for security measures.**

The growing field of crypto-agility examines approaches to integrate security measures in an agile and flexible way, making updates easier and, therefore, encouraging a more frequent deployment of them. This paper contributes to this research field in the context of secure communication in two ways. We first examine the current state of crypto-agility by providing an overview of existing measures for OT systems. Then, we propose a new architecture concept with different deployment approaches to integrate security measures in a crypto-agile way. Based on a security library with a generic interface and a flexible proxy application, our architecture is capable of securing both new OT systems and existing ones via retrofit.

Keywords*—*Security, Crypto-Agility, Automation, Industrial Control Systems, SCADA, Transportation, Real Time, Communication Systems, Cryptography, Proxy, Gateway, Retrofit

I. Introduction

Cybersecurity is an emerging topic in the field of *Operational Technology* (OT). Nowadays, automation and control systems like *Industrial Control Systems* (ICS) are threatened by cyberattacks just like enterprise systems. This is especially important for automation systems within industrial processes or critical infrastructures, such as transportation or supply systems. Due to these increasing threats, various new regulations have been passed [1], [2] to improve resilience against cyber threats. Among various topics, operators of critical infrastructures in Germany have to prove the effectiveness of the implemented security measures to the Federal Office for Information Security (BSI) on a periodic basis. Due to the limited expected lifetime of cryptographic algorithms published by regulation bodies, e. g. by the BSI [3], and the vast amount of potential vulnerabilities [4] within these systems, deployed security measures have to be updated and maintained continuously.

Compared to IT enterprise systems, however, OT systems differ greatly in terms of updates and maintenance operations. Typical OT systems are designed with a mission time of up to 25 years, until the equipment is replaced on schedule. Software updates are installed only when strictly necessary during infrequent on-site maintenance (based on the common „*never change a running system*“ mentality). Hence, security measures can barely be updated or newly integrated after initial deployment. This inertial technological enhancement is aggravated by the systems consisting of devices from numerous manufacturers with vastly differing security capabilities. As a result, the effort to create a secure system based on a common set of state-of-the-art measures and best-practices is complicated and expensive, if at all possible.

To overcome this problem, the arising research field of *crypto-agility* examines solutions to simplify the maintainability of security measures. Especially for the considered OT systems with their long service life, thorough measures of crypto-agility are key to future-proof the systems against cyberattacks and to increase the expected lifetime of the security features, and, hence, of the overall systems. In the first part of this work, we provide an overview of the current state-of-the-art regarding crypto-agility in OT systems, focusing on secure communication within OT systems. We analyze existing security measures for OT communication and then identify problems and limitations thereof, which could be improved by thorough crypto-agility capabilities. Based on the identified state-of-the-art, we propose a new solution concept to increase the level of crypto-agility in OT systems. A key aspect of our concept is to create a fast migration path for existing systems to retrofit security features with agility capabilities to counteract the described inertia. Lastly, we verify the viability of our concept in a case study to examine the impact on overall system performance metrics.

In summary, the paper makes the following contributions:

1) We give an overview of the current state of security measures within OT communication systems, elaborating on problems and limitations regarding crypto-agility capabilities.
2) Then, we present our concept to add crypto-agility capabilities into both existing and new OT systems with its various deployment approaches.
3) In a case study, we verify the viability of the concept and compare the different deployment approaches, with a focus on retrofitting existing systems.

The remaining paper is structured as follows. In Section II, the current state-of-the-art regarding crypto-agility in OT communication systems is presented. Based on a general overview of OT security and a definition of crypto-agility, current problems from both a software and a hardware perspective are discussed. Section III presents related work that already approaches the identified problems of current systems, and introduces approaches to improve crypto-agility comparable to ours. In Section IV, we present our concept and explore different approaches to actually deploy the concept within real OT systems. To verify the concept, the results of our case study are discussed in Section V. Finally, Section VI draws a conclusion and hints at future work.

II. Current state-of-the-art

In this chapter, we give an overview of security measures in OT communication systems and elaborate on problems

and limitations regarding crypto-agility. Initially, we briefly introduce OT communication systems (Part II-A) and present currently available security measures (Part II-B). Thereafter, we provide a definition of crypto-agility and elaborate the different branches of this research field for our work in Part II-C. Lastly, we analyze the identified problems and limitations of current measures from a software (Part II-D) and hardware perspective (Part II-E).

A. OT Communication Systems

Before we dive into the research field of crypto-agility, an overview of current OT communication systems is given. In general, an enterprise system can be split into multiple layers based on the *Purdue Enterprise Reference Architecture* (PERA) model [5], covering both its IT and OT domain. Makrakis et al. presented a modern adaptation of this model [6], depicted in Figure 1.

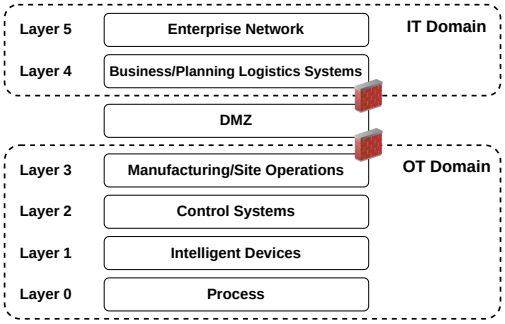

Figure 1. Modern adaptation of the Purdue Enterprise Reference Architecture (PERA) model, based on [6]. The layers are divided into IT and OT domain, separated by the DMZ (demilitarized zone) to secure the communication between the two domains.

Figure 1 shows the separation of IT and OT domains with their respective layers. Each layer implements a specific functionality of the overall system and further abstracts the underlying layers (e. g., a manufacturing process, or a complex transportation system). As the IT layers typically contain functionality that requires external communication (e. g. customer access or cloud services over the public internet), there is a strict isolation of the IT from the OT domain to protect the OT from malicious actors. Only absolutely necessary connections between IT and OT are allowed for the overall system to work, which are supervised and controlled by a special DMZ layer that integrates thorough security measures (indicated by the two firewall icons in Figure 1).

This separation and isolation of the OT domain with only very limited access from the IT is a key security aspect of current systems, as cyberattackers have to gain access to the OT before they can attack it. However, current developments regarding *IT/OT-Convergence* weaken this strict separation, and more connections are allowed to enable new and innovative functionality within the overall system (e. g. live monitoring of a process from the IT domain, predictive maintenance). Furthermore, OT systems grow more and more complex, resulting in a larger potential for configuration errors that ultimately lead to an increased number of entry points for attackers. On top of this, in many systems, cellular connections over public networks are used nowadays in addition to the wired network, breaking the isolation between the IT and OT domain. In total, the attack surface for the OT domain is growing. As a result, more thorough security measures have been created and deployed in OT systems, primarily derived from established IT security solutions.

Within the OT domain, each layer consists of different components connected by a communication system, autonomously handling a part of the system or process to abstract it to the next higher layer. This leads to a tree-like structure, with each layer being connected to the one above and below. Communication happens both vertically and horizontally. Vertical communication is primarily used by an upper layer for supervising and managing tasks of a lower layer. For tasks that are split onto multiple components, horizontal communication enables coordination and synchronization. Typically, the lower the layer, the greater the requirements for real-time behavior and reliability.

Within the OT layers, there exist both unicast point-to-point and multipoint communication flows, using various application protocols. Point-to-point connections are typically based on the client-server model, whereas the multipoint flows use a publish-subscribe model over a multicast system (also often referred to as producer-consumer). In both categories, application protocols building upon different layers of the OSI reference model can be found. Protocols building directly upon layer 2 (“L2” in the following) are used for time-critical communication, typically using Ethernet links. Application protocols building upon layer 4 (“L4” in the following) are rather used for less demanding connections regarding timing requirements. Among the most common point-to-point protocols are (non-exhaustive):

- OPC UA Client-Server (L4: TCP/IP)
- IEC 61850 MMS (L4: TCP/IP)
- Modbus TCP (L4: TCP/IP)
- Siemens S7 Comm (L4: TCP/IP)
- PROFINET RT (L2: Ethernet)

In the multipoint category, widely used protocols are (non-exhaustive):

- OPC UA Publish-Subscribe (L4: UDP/IP)
- MQTT (L4: TCP/IP)
- IEC 61850 GOOSE (L2: Ethernet)

Many of the application protocol specifications consist of multiple actual protocols with different characteristics for different use cases (e. g., an L2-based publish-subscribe protocol for real-time traffic together with an L4-based client-server protocol for supervisory functionality).

B. OT Security

A key security aspect of OT systems is the strict separation from the IT domain, with only very limited flows in between. The approach of separating subsystems where possible is further adapted within the OT domain by thoroughly utilizing network segmentation. Both within a layer of the PERA model and also across multiple layers, separated functional *islands* are formed with only very limited and properly secured connections between them. This approach is often further improved by adding virtual private network (VPN) technologies into the systems. These segmented system architectures limit the attack surface to a minimum, as lateral movement of a malicious actor is considerably restricted.

From a cryptographic perspective, most protocols have been adapted to support security measures to protect communication flows from malicious actors. Within the OT domain, common security goals to be achieved are:

- Availability (no malicious actor can prevent access to a system service for a legitimate entity)
- Integrity (transmitted data is protected against malicious modification)

- Entity Authenticity (a malicious actor cannot impersonate a legitimate entity)
- Data Authenticity (transmitted data originates from an authentic source)
- Confidentiality (a malicious actor is unable to eavesdrop sensitive information from the system)

Most of the protocols build upon well-established standards to achieve these goals. For example, L4-based protocols typically recommend using the *Transport Layer Security* (TLS) protocol to integrate security measures without a direct modification of the actual application protocol (e. g., for IEC 61850 MMS [7], for Modbus TCP [8], and also for the multipoint protocol MQTT for connections to a central message broker [9]). One notable L4-based exception is OPC UA, which does not build upon TLS, but rather integrates security measures directly into the application protocol [10]. Conceptually, however, the integrated cryptographic measures are very similar to the ones in TLS (same cryptographic algorithms, similar authentication mechanisms using digital certificates).

Multipoint protocols, however, cannot integrate typical security protocols like TLS, as those are designed for exactly two peers that perform a handshake during connection establishment. Within a multipoint group, however, there are typically more than two peers (one or more publishers, various subscribers), which all require the same cryptographic keys to apply the security measures. Hence, a key distribution mechanism is required, typically implemented using an additional node in the system. Each participating node in a specific multipoint group first contacts this *key server* to obtain the current group keys. This initial point-to-point connection between the node and the key server is secured using mechanisms already described above, both protecting the exchanged group keys and properly verifying the peer's legitimacy to join the group. For example, such mechanisms are specified for OPC UA [10] and for IEC 61850 GOOSE [7].

For L2-based protocols with real-time requirements, however, security measures are only cautiously specified and recommended, as the overhead of cryptographic operations may break the strict timing requirements. For instance, no cryptographic measures are officially specified or recommended for the PROFINET RT protocol by standard bodies due to this reason. However, research in this direction already proposed technically viable solutions to achieve both security and real-time behavior (e. g. in [11]).

In general, there are standardized security measures for various protocols available. Together with a properly designed, segmented system architecture, secure OT systems can be created and deployed today. However, to properly integrate the described security measures, all participating devices have to support the specific features. As already indicated in the introduction, OT systems and their devices are typically designed for mission times of up to a few decades, resulting in a slow deployment and adoption of new (security) features in existing systems. Furthermore, even if a completely secure OT system is newly deployed today, the ongoing development in the area of crypto-analysis and the increasing number and capabilities of cyberattackers necessitate ongoing effort to keep the initially achieved level of security over the long mission time. Hence, update capabilities and fast migration concepts are required to change and adapt security measures easily within deployed systems.

C. Crypto-agility Definition

Based on the literature works by Alnahawi et al. [12] and Mehrez & Omri [13], the following general definition of the term *crypto-agility* is derived: crypto-agility describes the capability of updating and replacing security measures during the lifetime of a component. Specifically, this means:

- Update the implementation of existing security measures (e. g. to fix a vulnerability)
- Update the list of supported cryptographic algorithms and their security parameters (e. g., add new algorithms, remove old ones, increase the key size)
- Incorporate and adapt to new functionality transparently (e. g. use hardware acceleration instead of a software implementation)
- Incorporate regional security regulations and comply with regional peculiarities (e. g. ShangMi ciphers for the Chinese market)
- Create transition mechanisms to enable safe and secure migrations to new security measures

The research field of crypto-agility is boosted by the ongoing effort to define and integrate new, quantum-resistant cryptographic algorithms into systems to protect them against the threat of quantum computers, known as *Post-Quantum Cryptography* (PQC) [14]. As elaborated, for example, by Ott & Peikert [15] or Paul [16], the migration to PQC demonstrates the necessity of crypto-agility, especially for OT systems. However, it is also a perfect opportunity to create and implement appropriate capabilities to achieve long-term security. Furthermore, proper transition mechanisms are a key aspect for OT systems, as the long mission times and the scarce update capabilities of equipment complicate the deployment of new security functionality on a system-wide level. Hence, crypto-agility must also consider retrofit options for already deployed systems.

Another important aspect of crypto-agility is further elaborated on by Sikeridis et al. [17]. They state that crypto-agility should not only be considered as a capability of a single instance within a system (e. g., a software library, a used protocol, or a single device), but also as an attribute of a complete enterprise system. Hence, all layers of the PERA model (see Figure 1) have to be considered. Their „enterprise-level“ view of crypto-agility considers the overall system and infrastructure, with all devices and the integrated security measures. Furthermore, they added key aspects like central control and maintenance, and overall documentation of deployed measures to their view. Paul and Niethammer also identified that an expanded view of crypto-agility is required in automation systems to provide a real benefit for long-term security [18].

For the remainder of our work, we define two dimensions of crypto-agility, between which we differentiate in the following analysis parts and in the presentation of our concept. On the one hand, there is *Implementation-Agility*. This dimension considers ways to easily update, extend or replace the implementation of security measures without adapting or changing the higher-level application using it. Hence, this dimension covers the list of supported security measures of equipment within a system, as well as the retrofit of security measures into devices and systems without prior support of those. On the other hand, we define *Configuration-Agility*, which encompasses all aspects regarding the configuration of the currently used set of security measures within a device or system from the list of supported ones. This covers the

maintenance and coordination of events like migrating a system to a new algorithm once all nodes support it.

In the next two parts, we take a more detailed look at the current state of crypto-agility capabilities within the presented security measures for OT communication systems and identify problems from both a software and hardware perspective.

D. Software-related Problems

Analyzing crypto-agility capabilities of secure OT communication from a software perspective, we identified problems in both of our defined dimensions. The main problem regarding implementation-agility is the tight coupling of security features to specific applications. To integrate security measures into an application, developers have to use common software libraries like OpenSSL, Botan, or WolfSSL directly from within the application itself. Those libraries commonly offer an extensive *Application Programming Interface* (API) for specific features (e. g., to establish TLS connections or to verify signatures). This setup results in various problems regarding crypto-agility that are already elaborated on in the literature, e. g., by Green and Smith [19] or by Georgiev et al. [20]. Available APIs are commonly very complex, especially for software developers without profound knowledge of security. In addition, proper and thorough documentation of the APIs is oftentimes lacking. Hence, using the libraries is very error-prone, leading to potential security vulnerabilities.

Furthermore, the resulting tight coupling between application and security functionality complicates an update of the security library (e. g. with support for a new algorithm) or a configuration adjustment (e. g. key length of an algorithm or logic to verify a peer). When a library update changes the API, updating the application is necessary, too. This increases the effort associated with an update for both the manufacturer of the component and the operator of the system, resulting in a slower deployment of fixes and new features. Ultimately, this leads to many systems being vulnerable to existing attacks and using older, potentially insecure implementations or measures. To improve implementation-agility, the application and the security implementation needs to be decoupled to enable independent changes to the security features without modifications to the application.

Regarding configuration-agility, the biggest problem is the static integration of a fixed set of security measures into the application without a way for external configuration of important parameters (e. g., algorithm selection, key length). The application manufacturer is solely responsible for the selection of supported functionality and its configuration in their individual product. This means that operators have no means of adapting the security measures running on their equipment to their needs without the manufacturer's support. Furthermore, the user-facing interfaces to configure security measures typically offer only limited options, and are typically integrated into the general maintenance interface of the application. Hence, each individual device must be configured individually in case of a security configuration change, drastically increasing effort and also coupling security maintenance to application maintenance. In typical systems with equipment from multiple manufacturers and of different releases (i. e. older and newer devices), this ends up in a „best-possible“ security configuration using the common set of supported measures. The result is unlikely to follow the current state-of-the-art, considering the slow deployment of new equipment within OT systems and the dependency on all manufacturers of used equipment to provide updates with new features for their products.

To improve configuration-agility, a centralized and easy way for operators to configure and manage the used security measures is necessary, independent of the application and with a manufacture-independent interface. This allows for a uniform, system-wide selection and an easier-to-coordinate migration process.

E. Hardware-related Problems

The hardware capabilities of deployed equipment also influences the achievable level of crypto-agility. This is especially important in OT systems, as the devices are oftentimes based on resource-constrained embedded hardware rather than powerful enterprise hardware. Furthermore, dependability and functional safety regulations, and the possibly rough environmental conditions of these systems further limit the options regarding hardware designs. We identified two hardware-related problems, both within the dimension of implementation-agility.

Firstly, the constrained processing resources of the devices limit the possibilities of software updates with newer cryptographic algorithms. This is especially relevant in the context of PQC, as the new algorithms have significantly greater requirements regarding CPU processing power and memory usage. For example, on a medium-sized microcontroller with 192 kB of available RAM, a TLS handshake using PQC algorithms alone occupies around 35 % of RAM, compared to around 1 % using classical elliptic-curve-based algorithms [21]. Even if a software update is possible to integrate these algorithms into deployed devices, the remaining available resources could be insufficient to make the new setup work properly. Hence, available hardware resources can impose restrictions on software-based implementation-agility.

Secondly, next to the hardware implications on software, there also arise problems due to hardware-assisted cryptography. In more recent microcontrollers and microprocessors, hardware peripherals are available to accelerate the calculations of cryptographic algorithms. This hardware acceleration greatly improves performance and sometimes enables the integration of security into these kinds of resource-constrained devices in the first place, as timing requirements could be achieved that are otherwise infeasible. Those peripherals, however, only support a defined static set of algorithms without a way to update them once a newer algorithm should be deployed. Furthermore, in case a vulnerability within the hardware implementation is identified, deployed chips cannot be upgraded, and fixing the flaw in the chip design is also very costly for the manufacturer. Hence, once the algorithms supported by an accelerator within a device are considered insecure or outdated, the device loses the hardware acceleration and has to fall back to software implementations, which suffer from the earlier described resource limitations.

Therefore, hardware-assisted cryptography needs to be upgradable after deployment. This enables integrating hardware acceleration of new cryptographic algorithms or fixing identified vulnerabilities in the design.

Next to the processor-internal hardware acceleration, there also exist external hardware modules that can be integrated into a device to add security measures with hardware assistance. These modules are commonly referred to as *Secure Elements* or *Trusted Platform Modules* (TPM) and are commonly available as either solder-down chips, small pluggable PCBs, or exchangeable plastic chip cards. Generally, those modules are used to store long-term keys related to the

cryptographic identity of the connected host device. The viability of this approach for OT systems has already been verified in different experimental investigations [22]–[24]. When soldered on the devices' PCB, those modules, however, suffer from the same problem as the internal peripherals because their fixed functionality cannot be upgraded once deployed. The exchangeable modules, on the other hand, significantly improve crypto-agility, as both hardware capabilities (implementation-agility) and also stored cryptographic keys with different parameters (configuration-agility) could be upgraded after deployment. Depending on the flexibility of the API that is used on the host application to access the module's functionality, crypto-agility could be further improved, as it is possible to swap the pluggable module without the need for updating the host application.

Such architectures, however, posses their own specific attack surface. As the communication interface between such a module and the host using it is typically not cryptographically secured, a malicious actor could tamper with the transmitted data and bypass the security features. Furthermore, as there is no cryptographic coupling between the module and the host, theft of a module results in a malicious actor taking over the valid cryptographic identity stored on the module. This opens the door for impersonation attacks. There are measures available for some types of external modules to protect the setups from the threats described above (e. g. [25]). However, to the best of our knowledge, there is currently no adaption of such protective measures that is suitable for OT systems, as these are typically based on human interaction with the setup when the module is used. In OT systems with no on-site human presence besides infrequent maintenance work, those measures are, therefore, not feasible. With those shortcomings addressed in the future, however, such modular setups would greatly benefit crypto-agility, as they would enable easy upgrades of hardware-based cryptography.

Based on the identified problems of current security measures for OT communication regarding both software- and hardware-based crypto-agility, the next section presents related work that already addresses those shortcomings.

III. Related Work

Before we introduce our concept to improve crypto-agility within OT systems, related work regarding the identified problems is presented. We split the presentation into solutions for the development of new equipment (Part III-A) and solutions for retrofitting existing systems (Part III-B).

A. Proposed Solutions for New Developments

To address the tight coupling between an application and security features, there are publications to provide simpler and more generic interfaces between the two to achieve better separation. For example, O'Neill et al. have two publications to outsource security functionality from applications to the operating system. In [26], they introduce the *Secure Sockets API* that enables applications to establish TLS connections using the common POSIX socket API. All configuration of the TLS connections is done on the device-level using a configuration file independent of the individual application. In [27], they propose the same idea for authentication functionality as an operating system service. Hence, applications pass certificates to the OS via a generic API to verify them. Both presented separation approaches simplify updating the underlying security functionality independent of the application (implementation-agility) and enable central management of the actual used cryptographic algorithms (configuration-agility).

The already mentioned architecture by Sikeridis et al. [17] also features a novel *Crypto Provider* software component that provides a flexible and abstract API for the application to isolate security from it, thereby facilitating independent updates (implementation-agility). Another central aspect of their presented software module is an additional control-plane interface for service and management that enables an operator to centrally configure all Crypto Providers within a system at once (configuration-agility).

Regarding improvements to implementation-agility from a hardware perspective, there is literature on integrating *Field Programmable Gate Arrays* (FPGAs) into devices and synthesize security functionality on them (e. g. for VPN functionality [28]). This would enable both fast hardware acceleration of algorithms and future update capabilities, as the FPGA could be reprogrammed in the field through a software update of the device. Furthermore, work within the enterprise context currently moves security functionality away from a server onto an attached programmable network interface card (NIC), so-called *SmartNICs* (e. g., [29], [30]). This modularization improves crypto-agility, as the SmartNICs are independent hardware modules connected to the server via generic hardware and software interfaces, enabling partial hardware upgrades in case of a new security measure that requires an updated SmartNIC, without affecting the main server.

B. Retrofitting Existing Deployments

All presented approaches to address the problems of current systems require substantial modifications of the applications or even of the whole device to integrate the proposed Crypto-agile security measures. In many currently deployed systems, however, it is not possible to update the applications or devices in such a fundamental way. Hence, retrofit approaches are necessary to provide a fast migration path for existing systems. This means that crypto-agility capabilities have to be integrated transparent to the application.

In the context of secure communication, a very promising approach for this integration is a design based on *proxies* that are integrated into the existing communication paths. They then intercept the traffic from the existing devices and apply the desired security measures. Such proxy designs have already been demonstrated in the OT context within the literature [31]–[34]. The traffic is typically intercepted either on the underlying transport layer (OSI layer 4) or directly on the application layer (OSI layer 7). The proxy is either deployed within the existing device as a software service (if that is possible, for example within an Embedded Linux system) or within an additional *Bump-in-the-Wire* (BitW) device that is integrated into the physical communication path. This integration of security measures, which is completely independent of the existing application or device, enables a retrofit of existing systems, but also generally achieves independence of the manufacturers of actual OT equipment. This final advantage of the proxy approach is a key aspect of our concept, as it results in a faster deployment of crypto-agility compared to the dependency on various manufacturers to integrate said features into their devices.

Each presented related work addresses some identified problems and limitations of security measures for OT communication regarding their crypto-agility capabilities and, thus, creating a solid foundation for our concept to build upon.

IV. Proposed architecture

In this section, we present our architecture concept to improve crypto-agility within OT systems. Our concept is similar to the work of Sikeridis et al. [17], however, adapted to meet the requirements of OT systems not considered in their enterprise-focused work. Our design-goal is to create a solution that improves crypto-agility both for new devices and also existing ones via a retrofit approach. The retrofit aspect is a key feature, as already deployed OT systems and devices need agile security measures to create a fast migration path to long-term security. For this process, however, new developments have to be considered simultaneously to create a reusable solution that can be widely adopted. We have designed various software modules to provide security measures for OT devices, which contain capabilities for both implementation-agility and configuration-agility to improve the achieved level of crypto-agility. In the following, we first present the detailed integration into actual devices, separated into deployment in new applications (Part IV-A) and retrofit for existing ones (Part IV-B). Afterwards, Part IV-C contains a discussion of the concept and its various approaches.

A. Deployment in new Applications

We created the *Agile Security Library* (ASL) to provide common features for secure communication behind a small, generic API for applications to use, similar to the Crypto Provider in [17]. The deployment approach is depicted in Figure 2. Newly developed applications simply use the generic interface to communicate with external peers. All functionality of the library is based on well-established standards and protocols to guarantee interoperability with other implementations. The small, generic API is the key feature to improve implementation-agility, as more functionality can be changed and upgraded within the library without modifying the interface for the application. Hence, it is easier to update the library independent of the application.

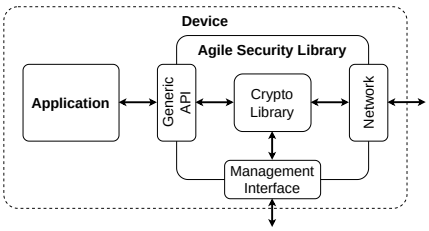

Figure 2. Deployment of agile security measures as a software library that can be used by applications. This approach requires the application to use the new small, generic API.

Along the interfaces for actual data processing, the library contains an additional management interface for the configuration of all security related functionalities. On the one hand, this moves security configuration away from the application onto the device-level, further improving their mutual independence. On the other hand, a distinct management interface simplifies the configuration for an operator as well as the implementation of a centralized control and management platform (i. e. fleet management). Those measures improve configuration-agility for devices incorporating the ASL.

The actual configuration of endpoints for applications is based on *profiles*. A profile contains all relevant parameters for an endpoint related to secure communication. For example, an endpoint profile describes the configuration of a TLS server with its supported cipher suites and its certificate chain. Via the management interface, profiles can be created and modified. The profiles and their associated data (e. g. certificates or private keys) are stored in an isolated and secured location within the device. On application startup, a specific profile is loaded with an API call to get an endpoint context. If no profile is specified, a default profile is used. This behavior makes sure that safe defaults are implemented. Using the obtained context, secure sessions can be created within the application for connections with peers.

Due to the generic API of the library, security functionality can be provided for communication systems on different OSI layers. Depending on the profile configuration, the desired security features are applied transparently to the application data without considering other OSI layers. The library further supports both synchronous and asynchronous program flows to provide flexibility for application developers.

Finally, the library is optimized for deployment in OT devices by focusing on aspects like minimal resource footprints and minimal dependencies on other libraries. Furthermore, it is designed to be used within various operating systems to offer flexibility and increase deployment possibilities.

B. Retrofit Deployment

To retrofit the crypto-agility capabilities of the ASL into already deployed OT systems without modifications to applications, we created an architecture design using the aforementioned transparent proxies. The proxies are placed within the communication path to intercept the application traffic and integrate security measures transparently for the existing peers. Within a proxy, the ASL is used to incorporate the improvements regarding implementation-agility and configuration-agility. Due to the transparent deployment of the proxy next to the application without any direct coupling, both dimensions of crypto-agility are even further improved compared to the direct integration of the ASL into applications. We elaborated on two integration approaches for the proxy, as described below.

The first approach is to deploy the proxy within the existing device as an additional software service. Depending on the particular device, this integration may require assistance from the manufacturer (for the installation of the supplementary application) or may be accomplished by the operator themselves, provided that the device is not locked down (i. e. the operator can install the supplementary services independently). As the proxy is a standalone service running besides the application, independent updates and configuration are straightforward.

The internal design of the proxy within the device is shown in Figure 3. The proxy application uses the ASL to incorporate both implementation-agility and configuration-agility. To manage the proxy itself, an additional management interface is present.

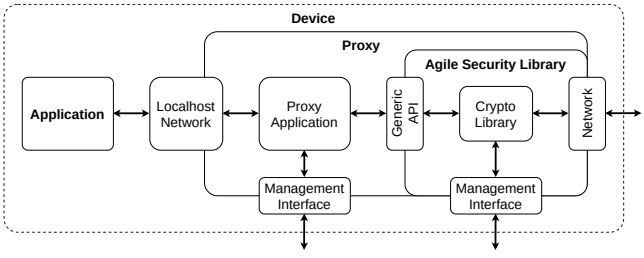

Figure 3. Deployment of the agile security measures within a proxy application deployed on the device itself. In this setup, the application does not need any modifications, but the proxy must be deployed on the device and be transparently integrated into the network path.

To implement the transparent integration into the communication path, the proxy intercepts the network traffic on the desired OSI layer from within the networking infrastructure of the operating system. This deployment results in the messages already wrapped within a suitable transport protocol (e. g., TCP, UDP, or a Layer 2 Ethernet frame). There are then two strategies for the proxy to handle the messages received from the application.

The first strategy is to extract the upper layer payload from these messages, secure them with the configured cryptographic measures, and finally pack them in a new message of a transport protocol. This results in the proxy terminating the connection with the application, requiring it to simultaneously establish a second connection with the remote peer. This setup breaks the end-to-end connection between the two applications and, therefore, influences protocol functionality like flow control. However, this strategy is otherwise fully transparent for both existing peers, providing maximum compatibility.

The alternative strategy is to wrap the complete received message including the metadata of the transport protocol within a secure message and send it to the peer. This mode of operation does not influence the end-to-end flow control of the existing connection between the peers, as each message is *tunneled* over the secure connection. Such tunnel behavior is default in regular VPN systems. However, both peers have to support the tunnel functionality (either by deploying a proxy or by processing the tunnel messages themselves), limiting interoperability and increasing the deployment effort.

Adding proxies to a system to intercept network traffic and add functionality this way is well-established within the cloud infrastructure domain. There exist various proxy applications for cloud servers to act as load-balancers or as security endpoints (e. g. NGINX or Envoy). However, those applications are optimized for cloud environments with their specific protocols and are built upon various technologies common for these systems (e. g. containerized applications and powerful enterprise hardware). Hence, using these available proxy applications in OT systems is not viable, especially considering the retrofit of existing devices.

Consequently, we created our own proxy application specifically for OT systems. This application can work both as a forward proxy (actively establishing connections to a peer) and as a reverse proxy (waiting for incoming connections). Furthermore, it has minimal software dependencies and its memory footprint is optimized to be as small as possible to maximize portability onto existing devices while simultaneously delivering the highest possible performance to limit interference with the communication flow.

Some available, more powerful commercial devices for OT systems nowadays also already feature application designs based on containers (e. g. using Docker) to increase the flexibility of application development and deployment. For such systems, the proxy could also be deployed as a container to be integrated into the containerized application complex.

For cases where an existing device is locked down or has not enough resources available to deploy the proxy as a service directly, we created an alternative approach based on a *Bump-in-the-Wire* device. In this scenario, depicted in Figure 4, the BitW device is integrated into a communication path as an interceptor. On this additional device, our proxy application is deployed to provide the same overall functionality for the system as the approach above.

This deployment approach further increases the achieved

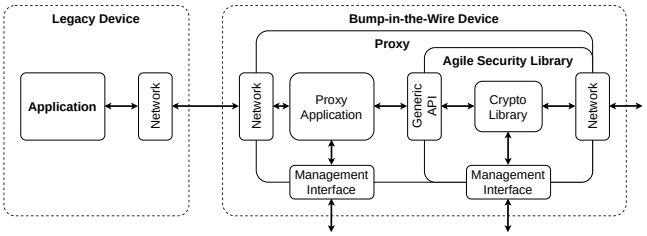

Figure 4. Deployment of the agile security measures within a standalone BitW device within the network path. In this setup, the legacy does not need any modifications. Only the additional device must be transparently integrated into the network path.

level of crypto-agility, as both hardware and software are independent of the existing device. Furthermore, basically any system can be retrofitted by deploying BitW devices. In one of our previous work, a BitW security gateway has already been presented for a very specific use case within energy supply systems [34], which served as the foundation for the more general and flexible BitW proxy approach in this work.

Besides using a standalone external device to deploy the proxy application, other already existing devices within the OT systems could also be considered to host the proxy. For example, modern network equipment like switches or firewalls feature powerful processors and are typically based on common operating systems. Hence, it would also be possible to deploy the proxy on those devices. This setup would result in equal crypto-agility capabilities, while avoiding the additional BitW devices within the system.

C. Discussion

The presented concept to integrate crypto-agility capabilities for both new and existing devices with its different deployment approaches covers all use cases within OT systems. However, each approach has its individual advantages and disadvantages. For that reason, we present a discussion of the three approaches in the following.

1) Achieved level of crypto-agility: Comparing the three deployment approaches regarding the achieved level of implementation-agility, the BitW proxy deployment wins. It offers the most extensive separation of security measures from an application and enables both independent hardware and software updates without any modification of the application. Deploying the proxy within an existing device, we achieve a high degree of software independence, but the proxy is still bound to the available hardware resources on the device. Finally, the direct integration of the ASL within a newly created application offers the least amount of implementation-agility, as we still have a direct coupling between application and security functionality via the small, generic API. Considering configuration-agility, all three approaches feature the same management interface, resulting in the same capabilities.

2) Impact on Communication behavior: When considering the impact of the deployment approaches on the communication behavior of a single device and of the overall system, the sole usage of the ASL is to be preferred. Due to the direct integration into an application, the overhead for each message is kept minimal, resulting in the smallest latency increase and the smallest bandwidth reduction. Comparing the two proxy deployments, the resulting system influence depends on the hardware and software environment. In case the existing device has enough resources available to deploy the proxy directly, the resulting influence can be small (however, still

larger than the direct library integration due to networking overhead). When resources are limited, the influence can grow larger, negatively impacting the communication behavior. The same applies to the BitW deployment: when powerful hardware is used, the influence can be quite low. With less resources available, the negative impact on system performance increases.

3) Costs and Deployment Effort: In this category, we have to clearly differentiate between new developments and retrofit deployments. For newly developed devices, the integration of the ASL directly into the application is the most viable approach, as it limits costs and effort for the manufacturer and also requires no additional integration steps or maintenance overhead for the operator after installing a new device with these capabilities. Considering the retrofit of existing devices, the two proxy approaches score differently, depending on the perspective of the manufacturer or the operator. For the manufacturer, the BitW deployment is more attractive, as it involves no work regarding existing devices. Furthermore, new devices can be sold to customers, even in case they use existing devices from other manufacturers. For operators, on the other hand, the direct integration into existing devices is better suited, as additional hardware costs are avoided. The deployment effort for both proxy approaches from a maintenance-perspective of the operator is comparable.

4) Safety and Reliability: Finally, the influence of the three deployment approaches regarding the overall safety and reliability of the system is considered. Assuming that all newly created devices incorporate measures for secure communication and, hence, a software library to implement those, anyway, the direct integration of the ASL performs best of all three approaches. The small, generic API for the application decreases the effort for developers to safely and securely integrate the security measures into their applications and devices. For an operator, a common management interface for various devices also decreases the effort to properly configure the security measures, resulting in less errors.

Both proxy deployment approaches for retrofit add additional software and possibly also additional hardware (BitW approach) directly into the main communication paths within the OT systems. As a result, both the hardware and software of the proxy have to be developed with a high level of assurance regarding safety and reliability to not negatively influence the overall system (e. g. a significant increase of the Mean-Time-to-Failure). Due to the limited scope and complexity of the proxy application and the possibly dedicated hardware development for exactly the desired use case, those requirements could be met with moderate effort, however.

In summary, the presented concept with its three deployment approaches enables the integration of agile security capabilities into both existing and new devices. With the key focus on the retrofit aspect of existing systems, a fast migration path is created until all deployed devices are capable of crypto-agility themselves. Using the dedicated management interface, an operator can configure the system based on the individual threat model and security goals, independent of the actual OT application.

V. Case Study: Securing IEC 61850 MMS Communication

To demonstrate the viability of our concept, we created reference implementations for both the ASL and the proxy application. Using these, we conducted a case study securing a test environment based on the IEC 61850 MMS protocol to evaluate the performance and system influence of our designs. In the following, we first present more details about the reference implementations in Part V-A. As indicated in the discussion above (see Part IV-C), the direct integration of the ASL into new applications should behave very similarly to an integration of a „classical“ security library, as only the API differs. Hence, we use the performance of this integration as a reference for the comparison of our proxy approaches within the case study. At first, we focus on the proxy application and measure its resource consumption (Part V-B) as well as the achievable bandwidth (Part V-C). Thereafter, we present various measurements to validate the timing influence of the proxy integration on system behavior compared to the ASL in Part V-D.

A. Reference Implementations

To maximize compatibility and portability of our approach, and also to demonstrate the viability in a broad device spectrum, we created our reference implementations for both Embedded Linux and the bare-metal Zephyr RTOS. Thus, we can demonstrate our approaches for both microcontroller-based and also more powerful microprocessor-based systems. For the underlying cryptographic library, we use WolfSSL, as this library supports a broad spectrum of different architectures and scales well from small microcontrollers up to powerful processors. For our agile security library, we created a wrapper around WolfSSL with our small, generic API and the additional management interface. Currently, the profiles with the endpoint configuration are implemented as JSON files which are loaded during initialization. A thorough user-facing interface with remote access is not yet implemented, however.

Our proxy application is implemented in C and runs on both Linux and Zephyr. It is currently limited to TCP/TLS functionality, acting as both a TLS forward and a reverse proxy. Thus, TCP connections can be intercepted and *upgraded* to TLS. The forward proxy waits for incoming TCP connections and upgrades them to TLS. The reverse proxy, on the other hand, waits for incoming TLS connections and forwards all traffic to a TCP connection. We only allow TLS in version 1.3 and only support mutual authentication, so both server and client have to authenticate using a valid certificate chain. Once all connections are established, data transmission is possible in both directions. The proxy can handle multiple parallel connections to multiple different hosts. The configuration is also done using a JSON file, with a proper management interface on the roadmap for future extension. Furthermore, other protocols are planned to be integrated in the future, e. g. UDP/DTLS or support for OPC UA Secure Channels.

For our case study, we created a setup using the IEC 61850 MMS protocol, as it is based on TCP and its security recommendations are built upon TLS (see Part II-B). The complete setup is depicted in Figure 5. We created both an MMS server and client using the open-source library libiec61850. The server contains a data model with various variables. After connection establishment (TCP handshake and MMS handshake), the client periodically reads single data objects to track value changes. The IEC 61850 library also offers optional support for TLS encryption directly integrated into the code, based on the library mbedTLS. This existing interface has been the basis for our integration of the ASL directly into the application.

To represent a realistic and reproducible test system in the laboratory, we deployed both the MMS server and client on two Raspberry Pi 4 with the Raspberry Pi OS lite, as this

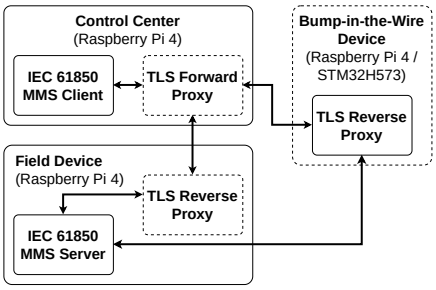

Figure 5. Test setup for our case study based on the IEC 61850 MMS protocol. Dashed lines indicate that the device or software module is only present in a subset of the performed tests.

hardware should roughly be on an equal performance level as typical commercial devices for OT systems and is widely available for validation of our measurements. The network between the devices is based on a single Gigabit Ethernet switch to minimize the influence of network latency.

For deployment of our proxy application on an external device, we limit the case study to a BitW proxy only for the server side in front of the Raspberry Pi to minimize the number of test cases. The BitW device in the test setup is based on either another Raspberry Pi 4 running Raspberry Pi OS lite or a microcontroller system based on the STM32H573i evaluation board from STMicroelectronics (Cortex-M33 with 250 MHz) running Zephyr RTOS.

B. Resource Consumption

The resource consumption of the proxy application is measured to estimate the overhead that is loaded onto a device when the proxy is deployed as an additional service besides the application. As this deployment approach is much more likely for Embedded Linux systems due to their better extensibility, we focus our measurements on this OS. To obtain the size of the executable, we compiled the proxy with statically linked dependencies. The resulting executable is roughly 2.3 MB in size. The proxy application for the Zephyr RTOS could not be measured in isolation, as it is embedded into a complete firmware image for the microcontroller. But the current firmware, containing the proxy and all supplementary code, fits onto a microcontroller with 1 MB of flash space.

To obtain the RAM footprint of the proxy during execution, the tool *Valgrind* with its heap profiler *massif* has been used. As stack space of Linux processes is also dynamically allocated, this measurement can obtain both Heap and Stack usage of a process. The peak memory usage of the proxy occurs during the TLS handshake, when the peer certificate chain is parsed. When only a single TLS handshake is performed using certificates based on the ECC secp521r1 algorithm, the peak memory usage reaches 117.6 kB. In a synthetic scenario with 10 simultaneous handshakes, the setup peaks at 389.6 kB. This footprint is considered to be small enough to make the proxy deployable on a typical Embedded Linux device. For microcontroller-based systems, the footprint is approximately the same, as the same code is used. On those systems, however, the number of parallel connections must probably be constrained to sustain viable RAM footprints.

Regarding CPU resources, the proxy application follows the *best-effort* principle. Hence, in case of limited free processing resources, the data processing performance of the proxy simply decreases. This effect, however, has to be validated on a per-device basis.

C. Bandwidth

To measure the achievable bandwidth, we used the well-known tool *iPerf*. Each of the two Raspberry Pi 4 run one instance of the tool, either as a client or as a server. We tested both the setup with the proxy deployed on the Pi itself and the one with the proxy on another Pi acting as a BitW device. Two test cases have been defined, one with a single connection for the bandwidth measurement and one with four parallel connections to use all CPU cores of the Raspberry Pis.

For both deployment approaches, we achieved nearly identical measurements. In the test case with only a single connection and transmission in one direction, both approaches achieve a bandwidth of around 180 MBit/s. When looking at the CPU resource consumption, only a single CPU core of the Pis is used, running at 100 %. The test case with four parallel connections utilizes all four CPU cores to 100 % each, as the proxy application creates a new thread for each connection, achieving a bandwidth of around 620 MBit/s (in both deployment approaches). Based on these results for both deployment approaches, we conclude that the AES implementation within the TLS library is the limiting factor regarding bandwidth. In the case of a test setup with full-duplex data transmission over the same connections, the achievable bandwidth per thread of around 180 MBit/s is split onto the two directions, cutting the total bandwidth roughly in half for each direction.

We also measured the setup with the STM32 microcontroller as BitW device. However, we identified the network stack on Zephyr to be a limiting factor regarding network bandwidth, achieving poor performance of only around 13 MBit/s in total using iPerf. Hence, those numbers are only a limited indication of the proxy performance running on the microcontroller.

In summary, the achieved bandwidth values show that the proxy approach is generally viable and reaches acceptable data rates. When deployed on a hardware platform with hardware acceleration for AES, the bandwidth should also reach the Ethernet link limit of 1 GBit/s.

D. Timing Influence

Finally, we measured the influence of the proxy integration on key timing parameters within the test setup compared to the direct integration of the ASL. The test setup from above leads to multiple test cases in which we obtained the timing parameters: insecure TCP connection between client and server, direct integration of the ASL into the application, proxy deployment directly on both devices as a software service and server-side BitW proxy deployment (Raspberry Pi and STM32 microcontroller). All TLS implementations use ECC secp384r1 certificate chains with one intermediate certificate, and only mutual authentication is enabled. For each timing parameter and test case, we took 10000 measurements and calculated the mean value and the 99$^{\text{th}}$ percentile.

The first measurement is the time to establish an IEC 61850 MMS connection. This includes TCP connection establishment, TLS handshake (if present in the test case) and MMS handshake. The results are depicted in Figure 6. The microcontroller-based values are omitted, as the TLS handshake takes around 300 ms on average, which would impede the visualization in the violin plot.

As can be seen, the integration of TLS into the handshake massively increases the connection establishment time in general, compared to the plain TCP setup. Considering the

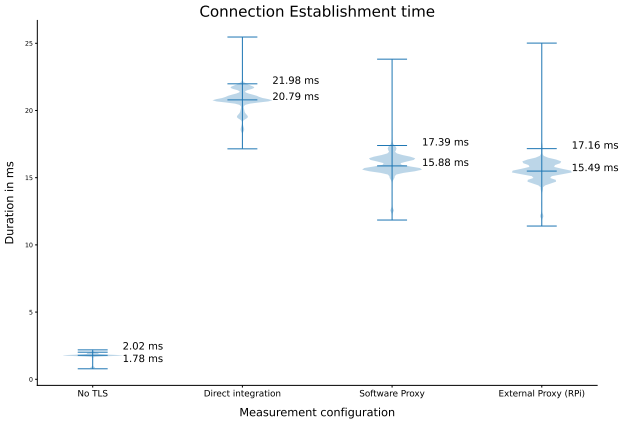

Figure 6. Violin plots of the connection establishment time for the different test cases (excluding the microcontroller-based BitW proxy). The written values indicate the mean value and 99th percentiles for each measurement.

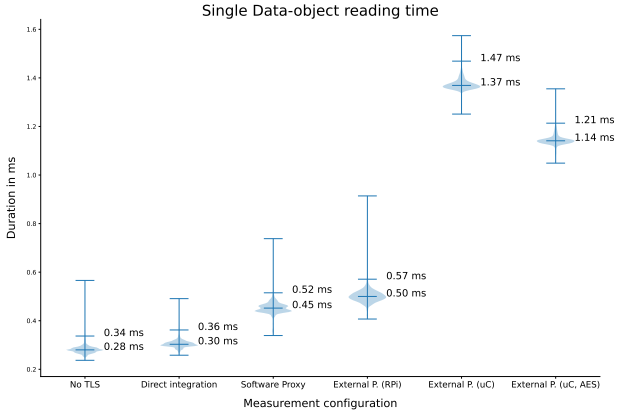

Figure 7. Violin plots of the time to read a single data object. The written values indicate the mean value and 99th percentiles for each measurement. The right-most measurement of the microcontroller-based deployment utilized the AES hardware acceleration of the microcontroller.

amount of data that is exchanged, and the cryptographic calculations performed during a TLS handshake with mutual authentication, however, such an increase is expected. Surprisingly, the measurements show that the two proxy deployments are even a bit faster than the direct integration of the ASL into the MMS application. This worse result of the direct integration is probably due to unoptimized data flows and worse event handling within the MMS application compared to the optimized proxy application, as the cryptographic code and the TLS configuration are identical in all deployments. The two proxy deployments show no noticeable difference, indicating that both approaches are viable. In total, the achieved handshake times with both proxies demonstrate that the proxy approach in general is a viable alternative to a direct integration. The much larger values from the microcontroller-based setup generally are not ideal. However, within OT systems, the connection is typically kept open for a long period, decreasing the negative impact of the longer connection establishment time. Hence, we also consider the proxy deployment on a microcontroller a viable approach regarding the connection establishment time.

The second measurement captures the time to read a single data object from the server. In this test, the connections are already fully established. Hence, we only measure the time overhead of the data processing and the resulting latency caused by the proxy. Figure 7 shows the results as violin plots.

The direct integration of the ASL only marginally increases the measured time to read a singe data object compared to the plain TCP setup. This results from the minor overhead of the TLS record protocol, mainly the AES encryption and decryption, running on a powerful processor of the Raspberry Pi. The two Raspberry Pi-based proxy deployments add only a minor latency overhead of 0.15 ms and 0.2 ms, showing that a proxy on a powerful hardware platform only marginally influences timing parameters. The minor latency increase in the case of an external deployment compared to a software deployment is probably caused by the additional physical network transmission over Ethernet compared to the in-memory-transmission over localhost within the software deployment.

The hardware of the STM32 microcontroller features a hardware accelerator for the AES algorithm. Hence, we created two versions of the Zephyr-based BitW proxy firmware: one with and one without utilizing this accelerator. As can be seen in the two right-most plots of Figure 7, both measurements add a noticeable amount of additional latency compared to the Raspberry Pi-based proxies. However, the measured values are still small compared to typical values of real-world network latency in Ethernet networks without hard real-time requirements, hence not causing a large additional delay. The hardware acceleration of the AES algorithm decreases the latency by around 0.2 ms. However, the value is still more than double the value of the Raspberry Pi based BitW proxy. This indicates that the software processing overhead of the TLS protocol is bigger than the required time to actually encrypt and decrypt the messages. Considering the much lower price and the much simpler hardware design of a microcontroller-based system compared to a Raspberry Pi, the deployment of the proxy on a microcontroller is definitely a viable approach in case the timing requirements of the system allow it.

To conclude, the results of the presented case study show the general viability of our proxy approach to retrofit agile security measures into existing systems, both within an existing device as a software service as well as within a BitW device with various hardware capabilities.

VI. Conclusion and Outlook

In this paper, we presented an analysis of the current state of crypto-agility in OT systems regarding secure communication and identified problems and possible improvements. Furthermore, we proposed a new architecture concept to integrate security measures into both current and future systems while improving crypto-agility. Finally, we demonstrated the viability of our concept with its deployment approaches and conducted a thorough analysis of the influence of the new setup onto system behavior.

In the future, we plan to further improve our reference implementations with support for more protocols (e. g. UDP and DTLS, OPC UA, IPsec, MACsec) and use cases (e. g. multicast communication). Furthermore, we want to examine the influence of the concept on systems with real-time requirements. Finally, we want to elaborate on further enhancements to our agile security library, both in terms of the coupling to the application (e. g. integration into the secure sockets API) and more thorough internal implementation agility (e. g. internal modularization).

Acknowledgement

The presented work is part of the research project *KRITIS Scalable Safe and Secure Modules* (KRITIS³M), which is funded by the Project Management Jülich (PtJ) and the German Federal Ministry for Economic Affairs and Climate Action (BMWK) under funding code 03EI6089A.

References

[1] Bundesamt für Sicherheit in der Informationstechnik. (2021, May) Zweites Gesetz zur Erhöhung der Sicherheit informationstechnischer Systeme (IT-Sicherheitsgesetz 2.0). Publication.

[2] C. Singh, "European cyber security law in 2023: A review of the advances in the Network and Information Security 2 Directive 2022/2555," *Cyber Security: A Peer-Reviewed Journal*, vol. 7, no. 1, pp. 82–92, September 2023. [Online]. Available: https://ideas.repec.org/a/aza/csj000/y2023v7i1p82-92.html

[3] Bundesamt für Sicherheit in der Informationstechnik. (2024, February) Technische Richtlinie TR-02102 - Kryptographische Verfahren: Empfehlungen und Schlüssellängen. Publication.

[4] S. D. D. Anton, D. Fraunholz, D. Krohmer, D. Reti, D. Schneider, and H. D. Schotten, "The Global State of Security in Industrial Control Systems: An Empirical Analysis of Vulnerabilities around the World," *IEEE Internet of Things Journal*, pp. 1–16, 2021.

[5] T. Williams, "The Purdue Enterprise Reference Architecture," *IFAC Proceedings Volumes*, vol. 26, no. 2, Part 4, pp. 559–564, 1993, 12th Triennal Wold Congress of the International Federation of Automatic control. Volume 4 Applications II, Sydney, Australia, 18-23 July. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667017485326

[6] G. M. Makrakis, C. Kolias, G. Kambourakis, C. Rieger, and J. Benjamin, "Industrial and Critical Infrastructure Security: Technical Analysis of Real-Life Security Incidents," *IEEE Access*, vol. 9, p. 165295–165325, 2021. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2021.3133348

[7] *IEC 62351: Power systems management and associated information exchange – Data and communications security*, International Electrotechnical Commission Std.

[8] Modbus Organization, "MODBUS/TCP Security Protocol Specification," 2018. [Online]. Available: https://modbus.org/docs/MB-TCP-Security-v21_2018-07-24.pdf

[9] OASIS MQTT Technical Committee, "MQTT Version 5.0 Specification," 2019. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf

[10] OPC Foundation, "OPC 10000-2 UA Part 2: Security," 2023. [Online]. Available: https://opcfoundation.org/developer-tools/documents/view/159

[11] T. Müller and H. D. Doran, "PROFINET Real-Time Protection Layer: Performance Analysis of Cryptographic and Protocol Processing Overhead," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2018, pp. 258–265.

[12] N. Alnahawi, N. Schmitt, A. Wiesmaier, A. Heinemann, and T. Graßmeyer, "On the State of Crypto Agility," in *18. Deutscher IT-Sicherheitskongress*. SecuMedia Verlags-GmbH, February 2022, pp. 103 – 126.

[13] H. A. Mehrez and O. E. Omri, "The Crypto-Agility Properties," in *Proceedings of the 12th International Multi-Conference on Society, Cybernetics and Informatics (IMSCI 2018)*, 2018. [Online]. Available: https://www.iiis.org/cds2018/cd2018summer/papers/ha536vg.pdf

[14] National Institute for Standards and Technology. Post-Quantum Cryptography. NIST. Accessed: March 27th, 2024. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography

[15] D. Ott and C. Peikert, "Identifying Research Challenges in Post Quantum Cryptography Migration and Cryptographic Agility," 9 2019. [Online]. Available: http://arxiv.org/abs/1909.07353

[16] S. Paul, "On the Transition to Post-Quantum Cryptography in the Industrial Internet of Things," Ph.D. dissertation, Technische Universität Darmstadt, Darmstadt, 2022. [Online]. Available: http://tuprints.ulb.tu-darmstadt.de/21368/

[17] D. Sikeridis, D. Ott, S. Huntley, S. Sharma, V. K. Dhanasekar, M. Bansal, A. Kumar, A. U. N, D. Beveridge, and S. Veeraswamy, "ELCA: Introducing Enterprise-level Cryptographic Agility for a Post-Quantum Era," 2023. [Online]. Available: https://ia.cr/2023/1539

[18] S. Paul and M. Niethammer, "On the importance of cryptographic agility for industrial automation: Preparing industrial systems for the quantum computing era," *At-Automatisierungstechnik*, vol. 67, pp. 402–416, 2019, crypto Agilität braucht:* Agile APIs* Secure Software Update* Dokumentation über Crypto Einsatz (welche Algos an welcher Stelle für welche Funktion).

[19] M. Green and M. Smith, "Developers are Not the Enemy!: The Need for Usable Security APIs," *IEEE Security and Privacy*, vol. 14, pp. 40–46, 2016.

[20] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, "The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software," *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 38–49, 2012.

[21] G. Tasopoulos, J. Li, A. P. Fournaris, R. K. Zhao, A. Sakzad, and R. Steinfeld, "Performance Evaluation of Post-Quantum TLS 1.3 on Resource-Constrained Embedded Systems," in *Information Security Practice and Experience*, C. Su, D. Gritzalis, and V. Piuri, Eds. Cham: Springer International Publishing, 2022, pp. 432–451.

[22] O. Kehret, A. Walz, and A. Sikora, "INTEGRATION OF HARDWARE SECURITY MODULES INTO A DEEPLY EMBEDDED TLS STACK," *International Journal of Computing*, vol. 15, pp. 22–30, 2016.

[23] R. Matischek and B. Bara, "Application Study of Hardware-Based Security for Future Industrial IoT," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*, 2019, pp. 246–252.

[24] O. Gilles, D. G. Pérez, P. A. Brameret, and V. Lacroix, "Securing IIoT communications using OPC UA PubSub and Trusted Platform Modules," *Journal of Systems Architecture*, vol. 134, p. 102797, 1 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138376212200282X

[25] M. Talamo, M. Galinium, C. H. Schunck, and F. Arcieri, "Secure Messaging Implementation in OpenSC," *Journal of Information Security*, vol. 03, pp. 251–258, 2012.

[26] M. O'Neill, S. Heidbrink, J. Whitehead, T. Perdue, L. Dickinson, T. Collett, N. Bonner, K. Seamons, and D. Zappala, "The Secure Socket API: TLS as an Operating System Service," in *27th USENIX Security Symposium*, 2018. [Online]. Available: www.usenix.org/conference/usenixsecurity18/presentation/oneill

[27] M. ONeill, S. Heidbrink, S. Ruoti, J. Whitehead, D. Bunker, L. Dickinson, T. Hendershot, J. Reynolds, K. Seamons, and D. Zappala, "TrustBase: An Architecture to Repair and Strengthen Certificate-based Authentication," in *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, 8 2017, pp. 609–624. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/oneill

[28] A. Salman, M. Rogawski, and J. P. Kaps, "Efficient hardware accelerator for IPSec based on partial reconfiguration on Xilinx FPGAs," *Proceedings - 2011 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2011*, pp. 242–248, 2011.

[29] D. Kim, S. Lee, and K. Park, "A Case for SmartNIC-accelerated Private Communication," in *4th Asia-Pacific Workshop on Networking*. ACM, 8 2020, pp. 30–35. [Online]. Available: https://dl.acm.org/doi/10.1145/3411029.3411034

[30] B. Pismenny, H. Eran, A. Morrison, D. Tsafrir, A. Yehezkel, and L. Liss, "Autonomous NIC Offloads," *ASPLOS '21: Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, vol. 18, 2021. [Online]. Available: https://doi.org/10.1145/3445814.

[31] T. A. Rizzetti, P. Wessel, A. S. Rodrigues, B. M. D. Silva, R. Milbradt, and L. N. Canha, "Cyber security and communications network on SCADA systems in the context of Smart Grids," *Proceedings of the Universities Power Engineering Conference*, vol. 2015-November, 11 2015.

[32] O. Givehchi, K. Landsdorf, P. Simoens, and A. W. Colombo, "Interoperability for industrial cyber-physical systems: An approach for legacy systems," *IEEE Transactions on Industrial Informatics*, vol. 13, pp. 3370–3378, 12 2017.

[33] W. Hupp, A. Hasandka, R. S. D. Carvalho, and D. Saleem, "Module-OT: A hardware security module for operational technology," *2020 IEEE Texas Power and Energy Conference, TPEC 2020*, pp. 1–6, 2020.

[34] T. Frauenschläger and J. Mottok, "Security-Gateway for SCADA-Systems in Critical Infrastructures," in *2022 International Conference on Applied Electronics (AE)*, 2022.