A Mechanism for Seamless Cryptographic Rekeying in Real-Time Communication Systems

Heiko Bühler, Andreas Walz, Axel Sikora
Institute of Reliable Embedded Systems and Communication Electronics (ivESK)
Offenburg University of Applied Sciences
Badstrasse 24, 77652 Offenburg, Germany
{heiko.buehler, andreas.walz, axel.sikora}@hs-offenburg.de

***Abstract*—Cryptographic protection of messages requires frequent updates of the symmetric cipher key used for encryption and decryption, respectively. Protocols of legacy IT security, like TLS, SSH, or MACsec implement rekeying under the assumption that, first, application data exchange is allowed to stall occasionally and, second, dedicated control messages to orchestrate the process can be exchanged. In real-time automation applications, the first is generally prohibitive, while the second may induce problematic traffic patterns on the network. We present a novel seamless rekeying approach, which can be embedded into cyclic application data exchanges. Although, being agnostic to the underlying real-time communication system, we developed a demonstrator emulating the widespread industrial Ethernet system PROFINET IO and successfully use this rekeying mechanism.**

***Index Terms*—industrial communication, security, rekeying, industrial Ethernet, PROFINET IO**

I. Introduction

Modern operational technology (OT) imposes demanding requirements regarding the reliability and real-time capabilities of the communication link. As in legacy information technology (IT) systems, Ethernet is often used as the underlying transport medium [1]. Several protocols (e.g., EtherCAT, PROFINET IO RTC/IRT, and POWERLINK) have been developed to meet the real-time and reliability needs of industrial networks [2]. Cryptographic protection enables security regarding the integrity and confidentiality of messages. Although, it requires frequent updates of the symmetric cipher key used for encryption and decryption, respectively. Such an update of a cipher key is called rekeying.

Protocols of legacy IT security, like TLS, SSH, or MACsec implement rekeying under the assumption that, first, application data exchange is allowed to stall occasionally and, second, dedicated control messages to orchestrate the process can be exchanged. In real-time automation applications, the first is generally prohibitive, while the second may induce problematic traffic patterns on the network. As shown in [3], existing IT security solutions require dedicated adaptations to meet the OT specific needs.

One specific difference between IT and OT is the expected lifetime of a connection between two entities. For example, in legacy IT an HTTPS connection is closed after a typical request to the web server. In contrast, the connection between a programmable logic controller (PLC) and an actuator (e.g., motor driver) may be kept alive for several months or even years without interruption.

To protect the communication between two parties, symmetric cipher schemes are widely used for performance reasons. Those schemes require a shared secret key, known to both parties to perform encryption and decryption operations.

In [4] Luykx and Paterson establish bounds on the success probability of an attacker as a function of the amount of data processed under a single key using the AES-GCM authenticated encryption scheme. These bounds depend on the number of plaintext bytes encrypted and the number of forgery attempts by an adversary. Another reasons are nonce depletion [5], or because of elapsed time since key activation allowing an adversary to brute-force the key in use [6]. Considering those boundaries in our industrial Ethernet scenario, we conclude that rekeying is approximately required every six hours.

In the following we propose an approach for rekeying, with continuous data flow between two participants and enable two different types of rekeying procedures, one of which allows a message-less, 0-RTT (zero round trip time), rekeying without additional message exchanges. An important prerequisite to achieve 0-RTT rekeying is the independence between the key generation and the key activation. The former is less time critical, whereas the latter requires very low latency to not interrupt the flow of data, and to enable a seamless rekeying. This separation allows to implement these two functionalities into different units. Although this separation is not mandatory we consider the key activation to be implemented in hardware, while the implementation of the key generation can be kept flexible (i.e., either in software or hardware).

Especially in a typical OT setting, a single PLC may communicate with many other components (e.g., 128 devices), via many communication relations, each of which is protected with its own key. The 0-RTT rekeying is a viable approach to reduce the computational and network load to a minimum and additionally schedule the workload, i.e., preventing all communication relations to rekey at once, in those OT scenarios. This approach takes advantage of the existing cyclic communication between peers, by embedding required metadata into those cylic messages enabling seamless rekeying without additional management messages.

The remainder of the paper starts with a presentation of related work (Section II). It is followed by the description

©2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. DOI of original article: https://doi.org/10.1109/WFCS46889.2021.9483613.

of the underlying communication model we build our work upon (Section III). Based on that, we present a generic approach to enable seamless rekeying (Section IV) with some protocol extensions (Section V), and showcase its real-world OT applicability within the domain of PROFINET IO (Section VI) with a prototypical implementation on an FPGA-based hardware platform (Section VII). Finally, a conclusion is drawn and outlook on future work is made (Section VIII).

II. Related Work

Within the IT domain, several protocols for cryptographically securing communications do already exist. However, the rekeying mechanisms they implement use either explicit and additional messages to initiate rekeying and potentially lead to an interruption of the message flow. For example, the well-known DTLS 1.2 [7] uses a session key to protect data in transit. Occasionally it is required to update this key. To do so an entire DTLS handshake procedure is required, thus the communication is interrupted and no real-time requirements can be guaranteed anymore. The current draft of DTLS 1.3 [8] allows rekeying without the necessity of a handshake. However, an additional message is sent to indicate the switch to a new key, which is currently the best existing protocols can offer. Other protocols like MACsec [9], SSH [10] and IPsec/IKEv2 [11] do already require additional messages transferred between two peers to even initiate rekeying.

In [12] Runde et al. present an approach to negotiate a key between two peers in an industrial automation system by introducing protocol extensions for the IKEv2 [11] protocol. This rekeying always involves additional messages between two peers, for creating a child SA and agreeing on a new key.

In US patent US5940508A [13] the inventors perform seamless rekeying by using counters, which keep track of the amount of bits processed using the current key. At a configured threshold the switch to the other key is performed. The inventors assume the amount of bits sent is always the same amount of bits received at the other peer. Given this assumption, the switch indeed happens synchronized without interruptions. In case of message loss a re-synchronization of the counter values needs to be performed. In contrast to our work, there is no possibility to signalize different kinds of rekeying.

To the best of our knowledge there is no existing protocol, which allows to embed the signalization of an imminent rekeying into cyclic messages, while also managing different types of rekeying.

III. Communication Model

In this section we introduce a communication model for the intended use cases. First, we present a basic model representing OT communication systems. Then we specify the requirements to support cryptographic operations.

A. Basic OT Communication Model

We assume two equivalent peers communicating with each other over an unreliable channel, which may induce loss, reordering, duplication and delaying of messages. Additionally,

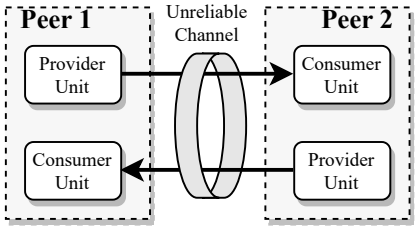

Figure 1. This basic communication model describes two peers exchanging data asynchronously with a lower and an upper bound for the time between two consecutively sent messages.

we assume this channel is under an active attackers control, that is, messages may be dropped, manipulated or forged by a malicious attacker. Any communication occurs asynchronously as unicast messages from one peer to the other. Both peers include a provider unit as a data source, and a consumer unit as a data sink as shown in Fig. 1. The provider unit of *Peer 1* communicates to the consumer unit of *Peer 2* and vice versa. The time t_s between two consecutively generated messages is bounded to a range between $t_{s,min}$ and $t_{s,max}$, that is, messages are not necessarily cyclic but may also be acyclic, as long as the time between two consecutive messages is within this range. These time bounds can be specific for each provider unit. In a real-world scenario one peer might be a PLC and the other peer a field device.

B. Cryptographic Requirements

Both peers exchange messages, which shall be protected. In contrast to legacy IT communication the confidentiality of the transferred data is not as much as important as its authenticity. Authenticated encryption with associated data (AEAD) [5] provides both data confidentiality and authenticity. In the following we use the terms encryption and decryption as synonyms for authenticated encryption and authenticated decryption respectively. This, in particular, includes the use of authenticity-only protection without any confidentiality. AEAD distinguishes plaintext (PT) and associated data (AD). While on PT both confidentiality and authenticity are provided, on AD only authenticity without confidentiality is achieved. This allows AD to be transmitted in cleartext, while still being protected (e.g., header fields, which cannot be transferred encrypted).

Symmetric cryptography depends on the existence of a secret key shared among both peers. As shown by Luykx and Paterson in [4], an update of the cryptographic key is required from time to time because of key degradation. However, the ongoing communication flow shall never be interrupted. A rekeying procedure requires computational resources (i.e., computing power and time) and possibly additional message exchanges. Thus, the rekeying is no immediate action. For a speedup of the rekeying a second key may be prepared for future use in the background. For every message, the legitimate receiver must be able to identify and look-up the cryptographic key for message decryption. Several requirements apply to this process. First, it shall be unambiguous and deterministic, that is, there must not

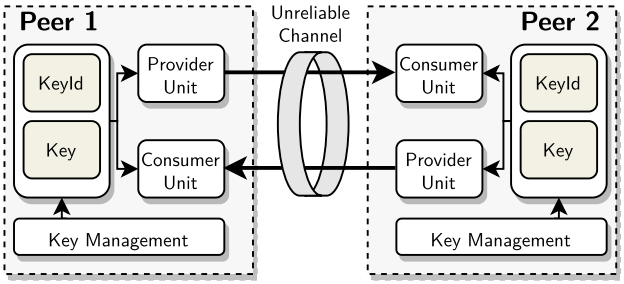

Figure 2. The extended communication model introduces cryptographic keys to enable encryption and decryption operations. These operations happen in the provider/consumer units and use the provided key, which is referred to by a KeyId. The key management, provides functionality to perform rekeying.

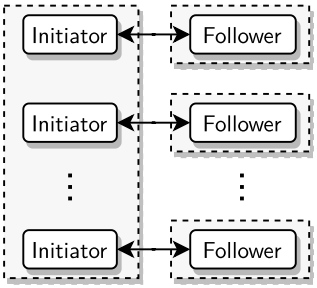

Figure 3. In an OT network a PLC (left) might have many one-to-one communication relations to field devices (right). The PLC fulfills the role of the initiator to schedule the workload of rekeying.

be the need for multiple decryption attempts under different possible keys, because real-time requirements forbids this. In [14] the inventors describe a seamless rekeying approach, which is based on testing all keys at hand. Second, it shall be low-latency and local, that is, it must not involve complex computations or interactions. Additionally, any rekeying shall happen synchronized between two peers, that is, the usage of a new key shall be allowed only if the other peer is able to process a message being encrypted with that key.

In order to distinguish keys of different generations from each other, they are designated by a Key Identifier (*KeyId*). As shown in Fig. 2 the *key management* is responsible for providing new symmetric keys with an associated KeyId. We assume the KeyId refers to a single key only. However, it is possible to use the KeyId to refer to several keys or other information.

IV. Generic Seamless Rekeying Mechanism

A. Rekeying Roles

In the following, we break the symmetry of the model to represent the OT reality, in which a single PLC may communicate with many devices, while a single device may only have a single connection to a PLC. Such an architecture is shown in Fig. 3, where each gray box represents a peer within an OT network. The communication always occurs in a one-to-one manner, however a single device (e.g., PLC) may have many of those relations. To allow the peer with many other relations to schedule the, possibly computationally expensive, rekeying procedures, we define it as an *initiator*, which always initiates the rekeying. The other peer is called the *follower*, which follows the initiated rekeying procedure.

B. Provision of Multiple Keys

Given the requirements stated in the previous section, we postulate: First, the switch to a new key shall happen instantaneously (i.e., low-latency) and second, it shall be possible to process delayed messages, which were potentially encrypted with an old key. In order to meet these requirements, one possible solution is to maintain at least two *slots*, of which one contains an active key. A slot is a container for a cryptographic key and potentially other information, and is referred to by the KeyId. The other slot(s) may either be used for future key generations and may be prepared in the background without interrupting the active communication or for old keys, which are still valid for a pre-defined period of time. The amount of slots within the initiator and the follower may be different. However, there must exist a way to unambiguously and deterministically decide which key has been used for the cryptographic operation. In addition, both peers shall know which slot will be used next in case of a rekeying.

A single slot, like shown in Fig. 2 is insufficient, because the flow of data might be interrupted during a rekeying procedure and any message encrypted with the previous key can not be decrypted anymore. Using two slots is sufficient to enable a seamless rekeying and allows decrypting messages of the previous key generation, for as long as no rekeying overwrites this old key. To support the usage of the previous key continuously or even older keys, a third or n-th slot may be involved. However, the KeyId must be unique along all slots to allow an unambiguous look-up. For further explanation we will use two slots (A and B) as shown in Fig. 4 of which one slot is in use, while the other is free for preparation.

Each consumer and provider unit map their view onto the data within a single slot into one of the following properties:

- **Unused:** The slot contains no valid key.
- **Pending:** The slot contains a valid, yet unused key, which is ready for subsequent usage.
- **Active:** The slot contains a valid key, which is actively used for encryption and decryption, respectively.
- **Obsolete:** The slot contains a still-valid key, which is phasing out, i.e., it will become unused after a specified amount of time.

When using two slots, the state space becomes the Cartesian product of the state space of two single slots. It yields 16 different states, of which, however, nine correspond to situations, which should not occur in practice (e.g., two active slots). The remaining set of valid states is shown in Fig. 5 as a state diagram of a finite state machine (FSM). Changing the state is done by one of the following transitions: (a) *Push*, which loads a new cryptographic key into a currently unused slot, (b) *Switch*, which marks the pending slot as active and

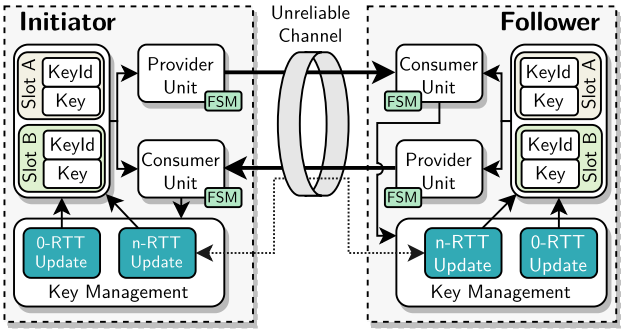

Figure 4. Both peers incorporate two slots containing a cryptographic key identified by the KeyId. Each provider/consumer unit maintains its own state of the slots via an FSM. Key renewal may occur either by a message-less 0-RTT or a message-based n-RTT update.

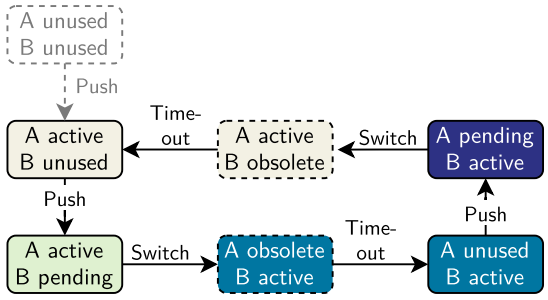

Figure 5. Valid state combinations of a view onto slots A and B. Transitions are either performed by pushing a new key into a slot, switching to a pending slot, or by the occurrence of a timeout for an obsolete key.

the currently active slot as obsolete and (c) a *Timeout*, which invalidates an obsolete key after a specified amount of time.

As shown in Fig. 4 each consumer unit and each provider unit consists of such an FSM to maintain the state of the shared slots independently of each other. That allows the provider and consumer unit of a peer to use different keys for encryption and decryption respectively. This situation may occur during the time interval in which outgoing frames are encrypted with the new key, however incoming frames have to be decrypted using the previously used key. The same holds true for the opposite direction, that is the follower, which might receive delayed messages that were encrypted with the old key, while the outgoing frames use the new key.

C. Identification of the Used Key

Both communicating peers must be able to identify the key used for protecting the message. This could be either performed explicitly by communicating the KeyId of the involved key, or by signaling the used slot. However, the latter requires both peers to know the state of the slots from each other. That is, each peer is able to derive in which slot the current key is situated on the other side. As both peers may have a different amount of slots, this approach becomes even more difficult.

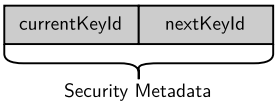

Figure 6. The seamless rekeying introduces a header containing metadata necessary to perform rekeying. This metadata is sent along the payload, which may be the actual application data.

D. Signaling of Rekeying

Any rekeying originates from the initiator, based on domain-specific triggers that are, for example the current sequence number or the key degradation status. For this purpose a signal indicating the kind of key update, is sent from the initiator to the follower. In return the follower sends a signal when the generation of the new key is done, and it is ready to be used subsequently. We incorporate two different kinds of rekeying:

- **0-RTT rekeying:** Each peer executes a deterministic KDF (key derivation function) locally. No dedicated message exchange is required, however no additional entropy is added to the key.
- **n-RTT rekeying:** By communicating with each other the peers agree on a new key (e.g., Diffie-Hellman key exchange). It is computationally more complex than a KDF and requires additional messages, however adds new entropy to the key.

After performing the rekeying, this new key is stored into the currently unused slot. In consequence that slot is marked as pending. Refer to fig. 5 with its *Push* transitions. At the follower's side, the necessity of a rekeying is announced by the initiator. However, an immediate switch to that new key is not valid, as the other slot is not ready yet on follower's side. Thus, the current slot is still in use for cryptographic operations. A switch to the pending slot will be valid, when both peers are ready to use the new key. Consequently, the follower updates the key as requested by the initiator and stores it into the unused slot assigning the new KeyId and marks that slot as pending. Afterwards it signalizes to the initiator that it is ready to use the new key. Now, both participants are ready to use the new key and eventually perform a seamless switch to the new key without interrupting the ongoing communication. The signalization is embedded into the cyclic messages and requires only a few bits.

V. Seamless Rekeying Protocol

Subsequently, we introduce a protocol to realize the prior mentioned concept of a seamless cryptographic rekeying. That protocol consists of a header containing metadata. This header information is sent with every cyclic message to synchronize the preparation and usage of cryptographic keys. By involving AEAD schemes it is treated as AD and therefore being transmitted in the clear, however, still protected against undetected manipulation.

One way to realize the signaling of rekeying and which key was used for encryption is to use the KeyId and additionally encoding the kind of intended rekeying into it. For this purpose,

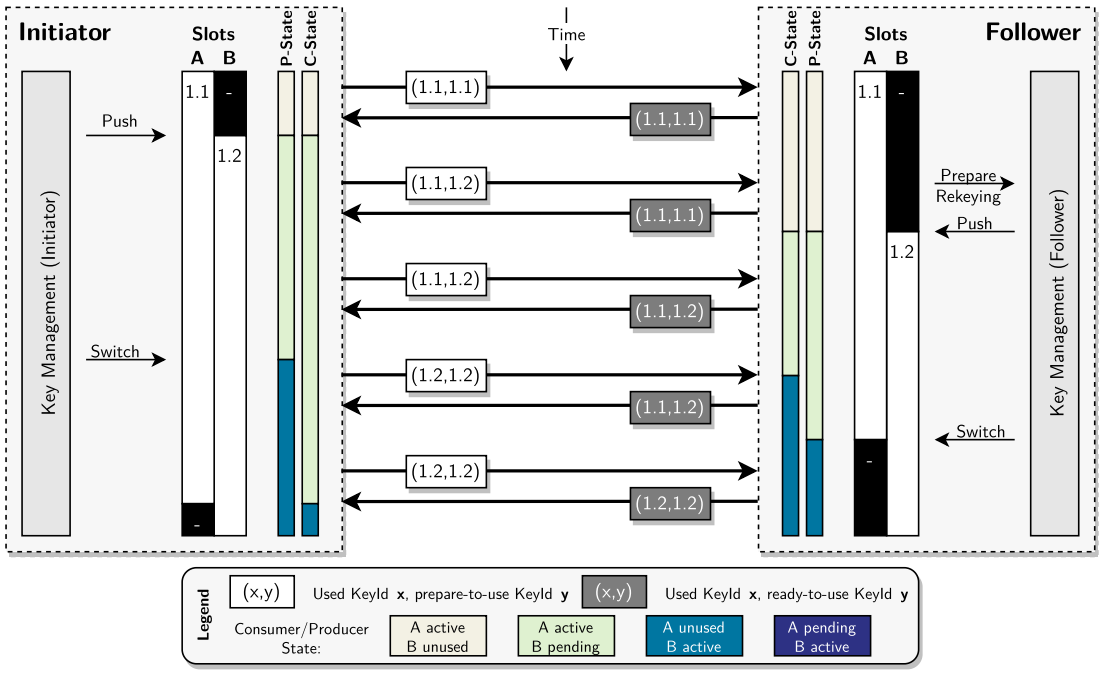

Figure 7. Rekeying is commanded by the initiator by incrementing the nextKeyId. The follower indicates its readiness by setting the nextKeyId to the prepared KeyId. P-State and C-State referrers to the view on the slots from the perspective of the producer and consumer unit respectively.

we include two different KeyIds into the header as shown in Fig. 6. The former *currentKeyId* is used to indicate which key has been used for the encryption operation. The latter *nextKeyId* is used to signal rekeying. Its semantics depend on the role of the peer (either initiator or follower). In case of initiator, it is used to announce an imminent rekeying to the follower. In contrast, the follower uses the *nextKeyId* to indicate which key is ready to be used.

To encode the kind of rekeying into the KeyId the following approach is used: The KeyId consists of the concatenation of the amount of n-RTT key updates, and the amount of 0-RTT key updates since the last n-RTT key update. For example, KeyId $= 2 \parallel 3$ is considered to be the key resulting after executing two n-RTT key updates and three 0-RTT key updates since the last n-RTT key update.

A full example of such a seamless rekeying is shown in Fig. 7. In contrast to existing solutions, our approach features an explicit signalization of an imminent rekeying procedure via a static embedding of a few-bits header into cyclic messages.

VI. Integration into PROFINET IO

Within the realm of industrial Ethernet protocol families, one commonly used technology is PROFINET. For cyclic data (like process data) the PROFINET IO RTC (real-time cyclic) protocol is used. We show a proposal on integrating the aforementioned seamless rekeying protocol into the RTC frame structure. For that purpose, the header is integrated into the RTC frame header after the frameID. We assume the existence of adequate mechanisms to cryptographically protect RTC messages, which may use the presented rekeying protocol. The resulting structure is shown in Fig. 8.

From the perspective of a legacy security unaware participant or intermediary, the security header is interpreted as part of the payload (e.g., IO data). However, a security aware participant is able to interpret the relevant bits as previously described. In terms of PROFINET vocabulary the IO-Controller (IOC) possesses the role of the initiator, while the IO-Device (IOD) the follower. Between those components exist two reciprocal communication relations (CRs), which both provide a one-way communication from a provider protocol machine (PPM) to a consumer protocol machine (CPM). The PPM maps to the provider unit, and the CPM to the consumer unit mentioned above.

Furthermore, we assume a key hierarchy consisting of a master key (result of a key agreement function) and a communication key (derived from the master key by a KDF). Each KeyId referrers to a specific combination of a master key and a communication key. Hence, each update of either the master key or the communication key implies an update of the KeyId. An update of the communication key increments the communication key index and performs a 0-RTT rekeying. Similarly, an update of the master key increments the master key index, however, resets the communication key index to zero and performs an n-RTT rekeying. The KeyId is a concatenation of these two indices.

From the perspective of the PPM, valid KeyIds are the currently active key only. In contrast, from the perspective of the CPM the current KeyId, the preceding KeyId and the next KeyId, either by 0-RTT or n-RTT rekeying, are valid values. To differentiate between the current, preceding and following KeyId from each other, at least two bits of information are

Figure 8. PROFINET IO RTC frame with additional currentKeyId and nextKeyId fields for rekeying purposes.

required. As the KeyId is now a concatenation of two key indices, it results in a minimum width of 4 bits. Combining currentKeyId and nextKeyId we fill a single octet.

VII. FPGA-based Demonstrator

Given the concept of the seamless key update in theory we developed a demonstrator that consists of several dummy PROFINET components. One of these dummies acts as an IO controller, whereas one or more act as IO devices. Each component is implemented on a DIGILENT Zybo Z7 development board, using the Zynq 7000 architecture, which incorporates a programmable logic (FPGA) and an ARM-based processing system on a single chip. This design choice allows us to offload less time-critical task into software, while performing time-critical tasks in hardware.

For demonstration purposes no entire PROFINET protocol stack, but only PROFINET IO RTC with the modifications mentioned in the previous section has been realized for real-time cyclic communication. To this end, a suitable packet parser and packet generator has been implemented in programmable logic for being able to receive and transmit security-enabled PROFINET IO RTC packets. In addition, a module responsible for slot management and handling of the security metadata has been implemented in hardware because of its time critical nature. Also, an interface for performing the cryptographic operations has been defined, however a real crypto engine is not implemented yet, but is intended for future work. Currently, a dummy engine is used, which takes a block of input data, performs a XOR operation with the key, and outputs the result. In case of authentication-only the previous block is XORed with the current result, thus producing a simple ICV. The absence of a real crypto engine however does not affect the validation results, as the rekeying procedure is independent of the actual cryptographic operations. Although, any error in key handling (e.g., usage of the wrong key) would have even been detected by our dummy engine. The generation and validation of payload, key derivation, generation of new security contexts and configuration of the hardware modules is performed by software modules. For this purpose either FIFO buffers or mapped memory is used to access the content of hardware registers by software. This demonstrator is used to test the concept of a seamless key update in a real-world application. Several tests have been successful and provide a platform for further work, e.g., including a real cryptographic engine.

VIII. Conclusion and Outlook

In order to cryptographically protect communication in real-time industrial communication systems rekeying is required from time to time. With the proposed protocol it is possible to perform rekeying without interrupting the data flow and without additional messages. All required metadata is sent piggyback with every message. Nonetheless, it is still possible to perform a message-based key exchange to add new entropy to the key. The protocol has been successfully tested on an FPGA-based demonstrator. For future work, the proposed protocol may be translated into a formal specification. This would allow performing model-checking to prove its correctness under all circumstances given the timing requirements. Automatic code generation based on a configurable model (e.g., amount of slots, time until timeout, ...) would make the adaptation and implementation easier, and might also be topic for future work.

Acknowledgment

Our work presented herein has significantly been informed by our participation in the Security Working Group of PROFIBUS & PROFINET International. We would like to thank the members for fruitful discussions. However, assumptions made herein regarding security extensions for PROFINET solely serve the purpose of showcasing our rekeying mechanism in a typical setting. There are no claims regarding official security extensions for PROFINET.

References

[1] M. Felser, M. Rentschler, and O. Kleineberg, "Coexistence Standardization of Operation Technology and Information Technology," vol. 107, no. 6, pp. 962–976, 2019.

[2] F. Klasen, V. Oestreich, and M. Volz, *Industrial Communication with Fieldbus and Ethernet*. VDE-Verlag, 2011.

[3] T. Müller, A. Walz, M. Kiefer, H. Dermot Doran, and A. Sikora, "Challenges and prospects of communication security in real-time ethernet automation systems," in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2018, pp. 1–9.

[4] A. Luykx and K. G. Paterson, "Limits on authenticated encryption use in tls," *Personal webpage*. [Online]. Available: https://www.atul.be/aelimits_2017_08_28.pdf

[5] D. McGrew, "An Interface and Algorithms for Authenticated Encryption," RFC 5116, Jan. 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5116.txt

[6] J. Longo, D. P. Martin, L. Mather, E. Oswald, B. Sach, and M. Stam, "How low can you go? Using side-channel data to enhance brute-force key recovery," Tech. Rep. 609. [Online]. Available: https://eprint.iacr.org/2016/609

[7] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2," RFC 6347, Jan. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6347.txt

[8] H. T. E. Rescorla, "The datagram transport layer security (dtls) protocol version 1.3," Internet Engineering Task Force (IETF), Tech. Rep. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-tls-dtls13

[9] "Ieee standard for local and metropolitan area networks-media access control (mac) security," *IEEE Std 802.1AE-2018 (Revision of IEEE Std 802.1AE-2006)*, pp. 1–239, 2018.

[10] C. M. Lonvick and T. Ylonen, "The Secure Shell (SSH) Transport Layer Protocol," RFC 4253. [Online]. Available: https://rfc-editor.org/rfc/rfc4253.txt

[11] C. Kaufman, "Internet Key Exchange (IKEv2) Protocol," RFC 4306. [Online]. Available: https://rfc-editor.org/rfc/rfc4306.txt

[12] M. Runde, S. Hausmann, C. Tebbe, B. Czybik, K.-H. Niemann, S. Heiss, and J. Jasperneite, "Sec_pro: sichere produktion mit verteilten automatisierungssystemen," 2014. [Online]. Available: https://serwiss.bib.hs-hannover.de/frontdoor/deliver/index/docId/499/file/SEC_PRO_Abschlussbericht_final.pdf

[13] C. A. Long, W. R. Worger, and B. R. Schaefer, "Method and apparatus for seamless crypto rekey system," Aug. 17 1999, US Patent 5,940,508.

[14] J. Åkerberg and L. Thrybom, "Methods and devices for security key renewal in a communication system," Jan. 5 2016, US Patent 9,231,929.

All links were followed on January 21, 2021.