PROFINET Real-Time Protection Layer: Performance Analysis of Cryptographic and Protocol Processing Overhead

Thomas Müller, Hans Dermot Doran

Institute of Embedded Systems (InES)
Zurich University of Applied Sciences, Winterthur, Switzerland
Email: {mulh, donn}@zhaw.ch

Abstract—**Recent times have seen an increasing demand for access to process-data from the field level through to the Internet. This vertical integration of industrial control systems into the IT infrastructure exhibits major drawbacks in the context of security. Such systems now suffer exposure to cyber security attacks well-known from the IT environment. Successful attacks on industrial control systems can lead to downtimes, malfunction of production machinery, cause financial damage and may present a hazard for human life and health. Current automation communication systems generally lack a comprehensive security concept. PROFINET is a widespread Industrial Ethernet standard, fulfilling general communication requirements on automation systems as well as explicit real-time requirements. We elaborate the challenges of protecting the real-time component of PROFINET. We specify the requirements and a concept for ensuring integrity and authenticity using a keyed-hash message authentication code (HMAC) in combination with the cryptographic hash algorithm SHA-3. With a proof of concept implementation of a PROFINET RT protection layer, the performance overhead for generation and transmission of this HMAC and other required data fields, e.g. to prevent replay attacks, could be analyzed. Based on these data the limitations of security technology on real-time systems were explored as was the optimization potential of hardware acceleration.**

I. Introduction

Ethernet-based communication plays an increasingly important role in the field of industrial control systems. Industrial Ethernet standards as PROFINET enable the integration of IT network technologies as cloud connection or web servers directly on fieldbus level. Product machinery applying such standards are attached to a company's local area network (LAN) and therefore, potentially, accessible from the Internet. While in standard IT environment authentication schemes, end-to-end encryption, firewalls and intrusion detection systems are state of the art and ensure a certain level in security, in most of the productive automation systems such countermeasures are more or less completely inexistent even though almost all known cyber attacks can be performed towards them. Indeed, components of an industrial control system nowadays are not able to verify the origin of a received message nor exclude the possibility that it was manipulated by a malicious intermediary. Since automation communication systems often control the operation of critical infrastructure as, e.g. nuclear power or water treatment plants [1], the lack of security solutions can no longer be ignored.

For this reason, a PROFINET security guideline was published [2] in which PROFINET specific requirements are linked to requirements on appropriate security solutions. Solutions proposed are on a higher abstraction level, i.e. instead of adaptions on protocol level, measures as physical network segmentation as well as management processes for training and awareness-raising of employees are described. Focusing on protocol security, the performance of different components of an IT security layer for PROFINET as symmetric and asymmetric encryption and block cipher- as well as hash-based message authentication mechanisms was investigated in [3]. In a related publication, the performance of different message authentication code techniques was analyzed more detailed and compared to theoretical estimations [4]. Within the scope of the same research project, considerations on the application of public key infrastructures (PKI) for automation systems [5] as well as on the establishment and storage of cryptographic key material [6] were made. Including a prior threat analysis, all the results of these publications are collected in a final report [7].

The novelty of our approach lies in the evaluation of suitability for protection of PROFINET traffic under strict real-time requirements with cycle times lower than 1 ms that are customary for high-performance application as for instance motion control. We therefore designed and implemented a proof of concept of a PROFINET real-time (RT) protection layer based on the keyed-hash message authentication code scheme HMAC applying the latest member of the Secure Hash Algorithm (SHA) family SHA-3 on a system on chip platform including programmable logic blocks (FPGA) and a resource constraint processing system. Based on this implementation, a thorough performance analysis was carried out and potential optimizations towards the use of custom hardware acceleration were elaborated to match strict timing requirements.

Section II starts with a short theoretical background on PROFINET before the main body of this work is described, which are: The specification of the requirements of a network security solution for PROFINET in consideration of existing requirements on industrial control system networks; a concept of a protection layer for PROFINET real-time communication including the description of the required cryptographic elements as well as the corresponding fundamentals; and details on the implementation of the proof of concept on a relevant platform. In Section III we then present the results of the performance analysis and elaborate potential future optimizations. Section IV depicts further work planned in this area and summarizes the results.

II. Methodology

A. PROFINET

The Industrial Ethernet standard PROFINET[1] defines real-time communication between decentralized peripherals within the PROFINET IO perception. Devices in a PROFINET IO network are divided into three classes: IO-Supervisor represents the engineering stations for projection, configuration and monitoring, IO-Controllers perform as communication masters and are responsible for establishing application relations (AR) and communication relations (CR) to an IO-Device. An IO-Device is generally a sensor or an actuator [8, p. 34]. An AR represents a virtual channel for grouping multiple CR between two components. There are three different types of CR, the Record Data CR, a non real-time (NRT) communication channel for transmission of parametrization data, the cyclic real-time (RT) IO Data CR as well as the acyclic RT Alarm CR [8, p. 42]. NRT Communication combines the RPC (Remote Procedure Call) protocol for start-up commissioning residing in the application layer relying on UDP/IP for transportation with utility protocols directly built upon the data link layer as, e.g. the Discovery and basic Configuration Protocol (DCP) (see right column in Table I). The cyclic RT communication is again divided into three classes: RT-Class 1 (RTC1) for cycle times between 5-10 ms, RT-Class 2 (RTC2) for typical cycles times of around 1 ms, which requires special switch hardware and RT-Class 3 (RTC3), suitable for high-performance isochronous real-time (IRT) motion-control applications with configurable cycle times down to the lowest possible resolution unit of 31.25 µs. The latter also requires special switches as well as a prior topology planning [9, pp. 48-49]. The acyclic RT communication is realized with the RTA protocol, used for transmission of alarms (left column in Table I).

The Ethernet payload of a PROFINET real-time frame (RTA and RTC1-3) is composed of a FrameID (2 bytes) to identify the telegram type, the actual RT payload (padded to minimal length of 36 bytes in case the Ethernet frame contains a VLAN tag, otherwise at least 40 bytes of length), a 16-bit cycle counter representing the relative transmission period in multiples of 31.25 µs and the data and transfer status fields (1 byte each, compare to Table II).

[1]PROFINET (acronym for Process Field Network): Industrial Ethernet standard (IEC 61158 and IEC 61784) maintained by PROFIBUS & PROFINET International (PI) with its headquarter in Karlsruhe, Germany.

	OSI Layer	Real-Time (RT)	Non-Real-Time (NRT)
7a	Application	PROFINET IO Services	
7b		**PROFINET RT Protection Layer**	RPC
6	Presentation		
5	Session		
4	Transport		UDP
3	Network		IP
2	Data Link	RTC1-3 / RTA	DCP / ...
1	Physical	100BASE-TX / 100BASE-FX	

TABLE I
PROFINET PROTOCOLS DIVIDED BY REAL-TIME (RT) AND NON REAL-TIME (NRT): SINCE RT PROTOCOLS DIRECTLY BUILD UPON THE DATA LINK LAYER (2), PROTECTION MECHANISMS CAN BE PLACED ANYWHERE BETWEEN THE APPLICATION LAYER AND THE DATA LINK LAYER.

Length $[Bytes]$	2	0-1440	2	1	1
Field Name	FrameID	Payload Data	Cycle Counter	Data Status	Transfer Status

TABLE II
PROFINET RT FRAME STRUCTURE [8, PP. 100-101][9, PP. 62-66].

B. Requirements Specification

Altough no specific security mechanisms are standardized for PROFINET yet, there exists a PROFINET security guideline [2], in which general requirements on security solutions, that shall not be violated after integration into a PROFINET or general a automation systems, are described. These are in short

1) no impact on the ability to meet real-time requirements;
2) straightforward and cost-efficient integration; and
3) robustness against temporary high communication load and unexpected input.

Highest priority is allocated to the availability of industrial control systems, but also ensuring device replacement shall be possible regardless on whether security functionality is implemented or not. Extremely important is the interoperability between legacy and security-aware devices. In addition, it must always be kept in mind that automation systems are generally operated in non-stop mode, i.e. periodic shutdowns for necessary security updates as they are common in the office IT environment are both unlikely and undesirable. This in turn stipulates extensive testing of implemented security mechanisms before the integration of such security-enhanced devices into an automation network. As it is to be expected that security weaknesses remain unpatched over longer periods.

a) Security Objectives: With respect to the general requirements mentioned above, the basic security objectives can be elaborated according their deployment on PROFINET:

- Device authentication: As a first step in the key negotiation procedure, both endpoints mutually authenticate themselves to each other to ensure a shared secret will be negotiated with a trusted communication partner.
- Message authentication: Using the negotiated key, a message needs to be cryptographically enriched in a way that a receiver can be sure the message originates from

a known and trusted sender. This prevents the processing of a malicious packet sent by an attacker.

- Integrity: Besides authentication, a receiver needs to be able to detect manipulation of messages, i.e. tampering by a man-in-the-middle.
- Confidentiality: To mitigate information disclosure, messages would need to be fully encrypted. This is not only very time consuming but also may not be of highest priority since in most use cases the transmitted data may not be confidential.

b) Implementation Principles: Almost all existing security solutions are designed and implemented for the use in standard office IT environment and therefore quite certainly not suitable for the usage in operation technology (OT) networks without adaption [10]. Nevertheless, solutions to protect PROFINET communication should build upon well-established and widespread standards. Which allows benefit from long-term experience and proven implementations to be drawn. Also notable is a broad acceptance by the community if a solution relying on familiar standards can be announced.

c) Integration Principles: As soon as a security solution for PROFINET is standardized, device and stack vendors will integrate the defined functionality into their products. To reduce the risk of erroneous implementations and security weaknesses, the process of adoption to the security standard should be clearly defined and as easy as possible. Also, incompatibility between different vendors shall be prevented by ensuring that the number of alternative realizations is minimal. It is in the very nature of network applications, that, considering the OSI[2] reference model, the level of complexity of individual parts increases the higher the layer they rely on. Although an introduced layer for the protection of PROFINET communication theoretically could be placed on any layer between the data link layer and the application layer (see Table I), choosing lower layers allows integration into less complex structures. The ITU-T X.800 [12] recommendation on security architectures within the OSI model proposes the placement of both message authentication and integrity services in the network or transport layer. In regard to possible extensions of the protection layer for also supporting protection of NRT communication, this recommendation should be taken into account.

C. Concept

1) Message Authentication and Integrity: A keyed-hash message authentication code (HMAC) is a specific construction for calculating a message authentication code (MAC, further referred to as integrity check value, abbreviated to ICV) involving a cryptographic hash function in combination with a shared secret key. It is used to simultaneously verify both the data integrity and the authenticity of a message. HMAC was designed to be combined with any cryptographic hash function (see corresponding RFC2104 [13]). A previously negotiated shared secret key first has to be derived to fit into one block length (the basic cryptographically relevant unit, depending on the underlying hash function), i.e. it has to be padded by appending zeros up to the block length if it is shorter or otherwise hashed once (using the same hash function as the combined one) if its is larger than the block length (Eq. 1).

$$K' = \begin{cases} pad(K,0) & \text{if len(K)} < \text{B} \\ K & \text{if len(K)} = \text{B} \\ H(K) & \text{if len(K)} > \text{B} \end{cases} \tag{1}$$

The HMAC algorithm is composed of an inner and an outer hash function execution (see Eq. 2). The inner hash is calculated on the message appended to the derived key that is XORed with the inner padding block ($ipad$, 0x36 repeated for block length B). The output of the inner hash is then again appended to the key K', this time XORed to the outer padding block ($opad$, 0x5C repeated fro block length)[14, pp. 88-91].As long as the shared secret key remains valid, the key derivation and padding with the two blocks has only to be performed for the first initial iteration. This can lower the execution time significantly, especially if the shared key is longer then the block length B of the hash function. For long messages, HMAC should execute in approximately the same time as the embedded hash function[14, p. 91].

$$HMAC(K',m) = H[(K' \oplus opad)||H[(K' \oplus ipad)||m]] \tag{2}$$

To minimize the transmission time overhead and therefore the impact on strict real-time requirements, an ICV appended to the message shall not be longer than needed to fulfill a sufficient level of security. According to the RFC 2104, the generated output can be truncated down to half of the output size of the hash function but not less than 80 bits, to still have a robust protection against brute-force attacks for evaluating the key[13].

Common consensus of the relevant PROFINET working group considers SHA-3, the latest member of the Secure Hash Algorithm family (standardized in the FIPS[3] publication 202 by NIST[4][15]) as the optimal candidate for the use with HMAC. Originally announced under the name Keccak, this algorithm is based on a completely different structure, compared to its predecessors and a majority of other cryptographic hash functions, namely the sponge construction (see Fig. 1). This construction applies 24 rounds of fixed length permutations f on the state vectors $S_0 - S_n$ of length $b = 1600$ bits. The state vectors are divided into the capacity c (in case of SHA-3 corresponding to double the size of the digest output), and the rate r. The rate defines the input block length, what means the message will be padded up to a multiple of r bits, splitted up into blocks of length r ($P_0 - P_{n-1}$ in Fig. 1) and XORed to the rate segment of the corresponding state. This process is denoted by absorbing and is followed by the squeezing, i.e. the

[2]Open Systems Interconnection model: The conceptual partition of network systems into seven layers, developed by the International Organization for Standardization (ISO)[11]

[3]Federal Information Processing Standards

[4]National Institute of Standards and Technology

extraction of the digest out of the rate portion of state S_n (and subsequent if chosen digest size $n > r$, this is not the case for standardized SHA-3 versions and therefore faded in Fig. 1). SHA-3 is standardized for the four digest sizes 224, 256, 384 and 512 bits. Again to reduce the impact on performance, the 224 bit version shall be chosen. This then results in a HMAC algorithm with 1152 bits (144 Bytes) input block size and 224 bit digest size, truncated to 112 bit (14 Bytes) ICV that will be transmitted additionally to each frame. As already stated, even if the processing time of the HMAC can be minimized, the first initialization takes longer than all subsequent iterations using the same key. Therefore, to not interrupt the cyclic real-time communication, a fresh key has to be negotiated on a separate channel before the current key gets worn-out and the HMAC module has to be initialized with this new key prior the first use of it. To signalize to the receiver of a message, which key (and other associated data referred to by context here) was used for protection of the current frame, a specific identifier field is needed. Although in our working assumption we presume both communication endpoints to have already agreed on a shared secret key for a specific session without focusing on how, i.e. over what channel and how often, this negotiation was performed, we define the length of this additional field—the context identifier—to at least one byte. Considering the block length of the HMAC-SHA-3-224 algorithm, negotiation on session keys with length less than or equal to 144 bytes can omit the additional hash execution during initialization. The strength of hash functions depends on which property one wants to be broken [14, pp. 82-83]:

- Preimage resistance: Given a hash h, the effort required to find a message x such that $H(x) = h$ is proportional to 2^n.
- Second preimage resistance: Given a message x, the effort required to find a message $y \neq x$ such that $H(x) = H(y)$ is proportional to 2^n.
- Collision resistance: The effort to find an arbitrary pair of messages (x, y) such that $H(x) = H(y)$ is proportional to $2^{n/2}$ (this is referred to by the birthday paradox).

Since in the proposed protection scheme the message is transmitted in clear-text along with the digest, the only practical attack would be on the second preimage resistance, i.e. find a message of meaningful content (respecting the frame structure in Table II) other than the original but producing the same truncated ICV. Finding such a preimage would expectedly require about 2^{112} HMAC calculations, what would take several billion years on a single desktop machine.

2) Replay Protection: Through the integration of message authenticity and integrity services, it can be ensured that only trusted endpoints can send messages to a device that will be processed and also that they were not altered en route. Nevertheless, an attacker intercepting messages containing an ICV could store them and transmit them again later, i.e. perform a so-called replay attack. This means, without additional measures, an attacker can force a device to maliciously alter its behavior because the frame passes the verification

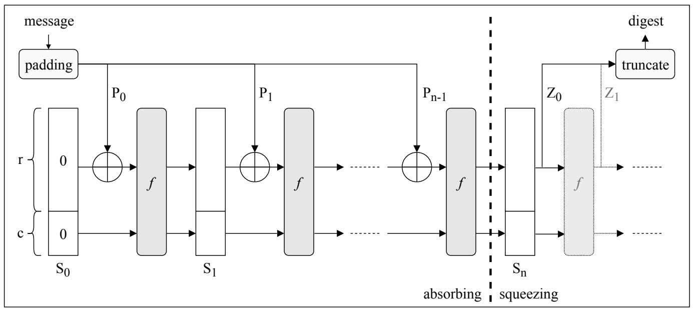

Fig. 1. SHA-3 (Keccak) sponge construction.

even if it was already sent long time ago. Such replay attacks can easily be mitigated by introducing a sequence counter that is incremented for each frame. Consequently, to completely prevent the possibility of replayed messages, the current session key has to be refreshed after an overflow of this sequence counter. This fact disqualifies the existing PROFINET Cycle Counter as standalone representative for a anti-replay sequence counter, since it consists of only 16 bits representing the transmission times in multiples of 31.25 µs and therefore overflows after approximately 2 s. It is infeasible to perform a complete key negotiation each 2 s, even within a seperate channel, without ever disturbing the normal operation. Therefore, another additional field has to be introduced that can be incremented on each overflow of the PROFINET Cycle Counter. Specifying a minimal length of 2 bytes for this counter extension, the period for key renewal can be enlarged to about 1 1/2 days, what we consider as enough time to schedule the establishment of new key material. One could state, that an even longer key could be chosen such that the key never has to be renewed within the complete operation period of a system or at least within a maintenance cycle. The longer a key is in use, the more data was protected using this key and therefore could be available to an attacker for deeper analysis that probably could reveal the key sooner or later. For this reason, the NIST has setup a recommendation for so called cryptoperiods of keys, i.e. how long the same key can be used before it wears out and therefore shall be refreshed. For symmetric authentication keys, this period is recommended to be shorter than 2 years (compare to Section 5.3.6 in [16]). With respect to this definition, there is no need to take a length of more than 4 bytes for the extended counter into account. Focussing on a minimal performance impact, 2 bytes of length are assumed for further considerations.

3) RT Protection Layer: The PROFINET RT protection layer applies the elaborated fields for counter extension, identification of the used context and the actual integrity check value—17 bytes in total—to an unprotected frame (Alg. 1). Generally, the steps that needs to be performed for protection of a frame are: (1) increment the extended counter on overflow of the PROFINET Cycle Counter, (2) get the latest negotiated context containing the currently valid key, (3) compare the current context identifier with the stored one to determine if the HMAC needs to be initialized with a new key, (4) calculate the ICV with HMAC on the RT payload, extended counter and context identifier, (5) truncate the ICV, (6) update the stored context identifier for being able to reinitialize the HMAC as long as the key remains valid, and (7) assemble the protected frame by adding the generated fields.

Algorithm 1 PROFINET RT Protection Layer: Protect Frame

```
1: procedure PROTECT FRAME($frame$)
2:     if PROFINET Cycle Counter overflowed then
3:         $ctr \leftarrow ctr + 1$                      ⇒ update extended counter
4:     end if
5:     $ctx_{current} \leftarrow getLatestContext()$
6:     $id_{current} \leftarrow getID(ctx_{current})$      ⇒ get latest context id
7:     $buf \leftarrow extractRTPayload(frame)||ctr||id_{current}$
8:     if $id_{current} \neq id_{last}$ then                ⇒ Initialize HMAC
9:         $key \leftarrow getKey(ctx_{current})$             ⇒ get new key
10:        $icv \leftarrow HMAC(key, buf)$
11:    else                                             ⇒ Reinitialize HMAC
12:        $icv \leftarrow HMAC(buf)$                      ⇒ key already stored
13:    end if
14:    $icv_{trunc} \leftarrow truncateICV(icv)$            ⇒ truncate ICV
15:    $frame_{protected} \leftarrow (frame||ctr||id_{current}||icv_{trunc})$
16:    $id_{last} \leftarrow id_{current}$                   ⇒ update stored context id
17:    return $frame_{protected}$
18: end procedure
```

Algorithm 2 PROFINET RT Protection Layer: Verify

```
1: procedure VERIFY($frame_{protected}$)
2:     $buf \leftarrow extractRTPayload(frame_{protected})$
3:     $ctr_{recvd} \leftarrow extractCounter(frame_{protected})$
4:     if $ctr_{recvd} \leq ctr_{stored}$ then             ⇒ verify the counter
5:         return $Fail$                                ⇒ replay attack detected
6:     else
7:         $ctr_{stored} \leftarrow ctr_{recvd}$              ⇒ update stored counter
8:     end if
9:     $id_{rcvd} \leftarrow extractContextID(frame_{protected})$
10:    $ctx_{rcvd} \leftarrow getContext(id_{rcvd})$
11:    if $ctx_{rcvd}$ not exists then                   ⇒ verify context id
12:        return $Fail$                     ⇒ context not exists (anymore)
13:    end if
14:    $icv_{rcvd} \leftarrow extractICV(frame_{protected})$
15:    if $id_{rcvd} \neq id_{last}$ then                    ⇒ Initialize HMAC
16:        $key \leftarrow getKey(ctx_{rcvd})$                 ⇒ get new key
17:        $icv_{calc} \leftarrow HMAC(key, buf)$
18:    else                                             ⇒ Reinitialize HMAC
19:        $icv_{calc} \leftarrow HMAC(buf)$                  ⇒ key already stored
20:    end if
21:    $icv_{calc} \leftarrow truncateICV(icv_{calc})$         ⇒ truncate ICV
22:    if $icv_{calc} \neq icv_{rcvd}$ then                  ⇒ verify ICV
23:        return $Fail$                       ⇒ ICV comparison mismatch
24:    end if
25:    $id_{last} \leftarrow id_{rcvd}$                       ⇒ update stored context id
26:    return $Success$
27: end procedure
```

The protection layer also is responsible for the verification of the authenticity and integrity of a received frame (Alg. 2). Besides the obvious comparison of the calculated ICV over the received payload and the ICV attached to the protected frame (Line 22-24, Alg. 2), the verification procedure also has to make sure that the sequence counter (PROFINET Cycle Counter and extended counter) has incremented since the last received frame (Line 4-8, Alg. 2) and that the context used for protection exists, i.e. was not revoked in the meantime (Line 11-13, Alg. 2). In a first iteration we assume that the PROFINET RT protection layer can be configured with device internal configuration data and therefore is aware of the structure of the RT payload (to extract specific fields).

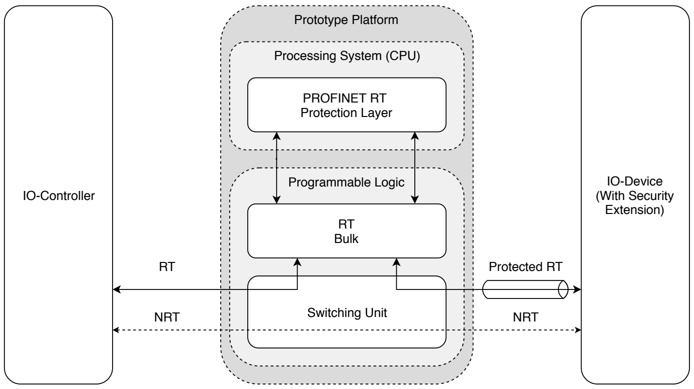

Fig. 2. Prototype architecture of PROFINET protection layer embedded in processing system of FPGA platform with switching unit and frame bulk optimized for PROFINET communication.

D. Implementation

The experimental framework must overcome the lack of specification of a key establishment process and with it the unavailability of security-aware IO-Controllers. This we achieve using an inline unit (Fig. 2) that is capable of transparently maintaining a secure communication relationship between a security-unaware IO-Controller and a security-aware IO-Device. That is, for instance, RT frames from the IO-Controller are converted to secured RT frames and forwarded to the IO-Device, and vice-versa. In a first iteration, only the RT frames are protected, in a second the proposed handshake protocol can be implemented on this inline unit and so the lack of suitable IO-Controllers can be compensated for. For the inline unit a transparent switch architecture based on an industry proven PROFINET IRT three port switch [17] was modified to allow the inline unit to receive unprotected PROFINET RT frames and add the protection on the fly. For the implementation, a Xilinx development board with a Zynq-7000 system-on-chip consisting of an ARM Cortex-A9 dual-core processing system and FPGA fabric was chosen. The three port switch was instantiated in FPGA fabric. This switch integrates a fast-forwarding unit and intelligent dynamic filtering based on information in the PROFINET frame (MAC address, FrameID, etc.). The prototypal implementation of the PROFINET RT protection layer resides initially in the processing system (ARM Cortex-A9) of the Zynq SoC platform - later in the FPGA fabric. Ingressing RT frames from the IO-Controller or IO-Device are filtered and forwarded to a frame buffer, the reception of a frame in the buffer triggers application of the previously described algorithms on that frame. Non-real-time traffic is forwarded unmodified from the IO-Controller to the IO-Device or vice-versa. The defining characteristic of this PROFINET protection layer is that it can be integrated in an IO-Device with minimum modification.

III. Results and Discussion

A. Performance Analysis

1) SHA-3: First, a software implementation of the 224-bit version of the SHA-3 alogrithm, implemented and unit tested on a host machine (intel Core i7 CPU), was analyzed respecting the influence of compiler optimizations on the performance. For this reason, the source code was compiled using different optimization options (for detailed description see the documentation for the GNU C Compiler (GCC) collection, version 6.3.0 [18]):

-O0 Default setting for reduced compilation time.
-O1 Enables optimization flags for reduced code size and execution time that do not have a great impact on the compilation time.
-O2 More optimizations that increase performance as well as compilation time. In addition to all options of *-O1*, specific options for algorithm reordering are used.
-O3 All options from lower optimization levels are inherited and additional flags, e.g. for function inlining, are used for even higher performance.

In addition to the flags already included within the mentioned optimization levels, loop unrolling can be enabled during compile time with the flags *-funroll-loops* and *-funroll-all-loops*. Fig. 3 shows the execution time of the SHA-3 algorithm under usage of the different optimization options during compile time for input sizes between 1 and 1500 bytes (all time measurements are averaged over 1000 iterations). The speedup between *-O1* and *-O2* is not large and also the size of the compiled binaries remains the same. Usage of level 3 optimization (*-O3*) is up to 10 times faster compared to the default. Using such compiler optimizations in the context of cryptography is not without risk. Even if the code is unit tested under optimized conditions, specially flags for function inlining and loop unrolling can expose the application to side-channel attacks, i.e. execution time probably differs depending on the input, which could help an attacker with access to a device to reveal the key (a detailed analysis on side-channel attacks for Keccak in combination with a message authentication scheme was done in [19]). Nevertheless, focusing on the best performance results, we decided on optimization level -*O3* and loop unrolling for the compilation of software running on the target platform.

Even with these optimization options enabled, the performance of the SHA-3 algorithm on the target platform (ARM Cortex-A9 dual core CPU, 866 MHz) is around 40 times slower than on the intel machine (Fig. 4).

2) HMAC: The HMAC algorithm was implemented specifically for support of reinitialization if the same key is used as in a previous iteration. To get a feeling on how much time can be saved with reinitialization, the performance of a reinitialization

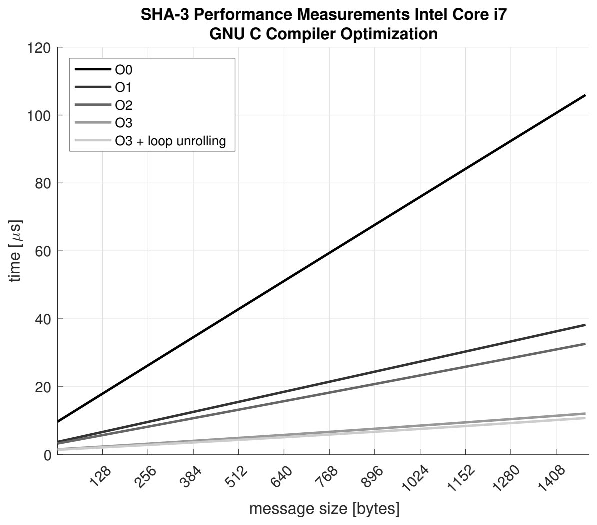

Fig. 3. Performance measurements of the SHA-3-224 implementation running on an Intel Core i7 platform compiled with different optimization flags.

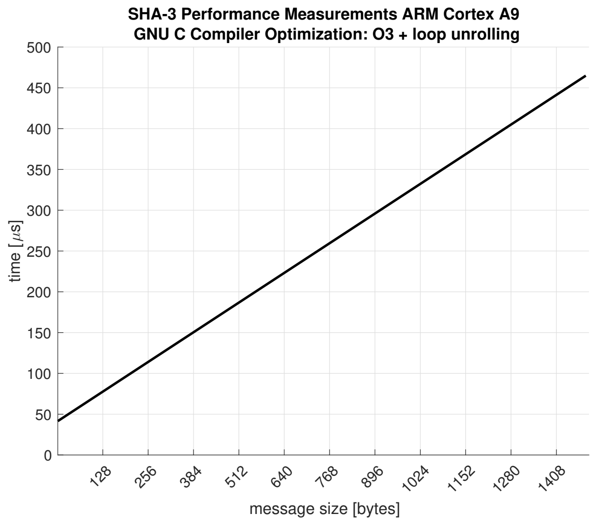

Fig. 4. Performance measurements of the SHA-3-224 implementation running on the prototyping platform with an ARM Cortex-A9 CPU.

execution is compared to an initial execution of the HMAC once with a key of length $<$ 144 Bytes (the block length of the underlying SHA-3-224), once with a longer key resulting in an additional hash execution, in Fig. 5. It can be seen that the performance of a reinitialized HMAC can be performed in less than 50% of the time for an initalization with a long key. The step function behavior can be explained by the padding up to a multiple of the block length of the underlying hash function.

3) RT Protection Layer: The PROFINET RT protection layer, implemented as described in Section II-C3, was analyzed respecting its performance in comparison to the pure execution of SHA-3 as well as HMAC (Fig. 6). As expected, for message sizes of length $(n \cdot r)$ with $n \; \epsilon \; (1 : \infty)$, the performance is the same as for pure SHA-3 execution. The

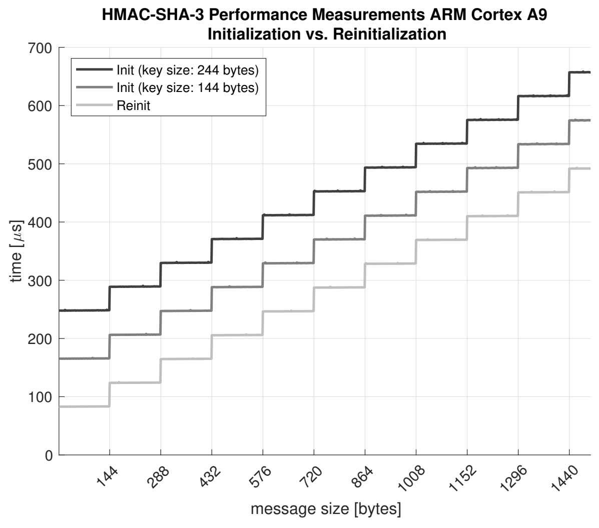

Fig. 5. Performance measurements of the HMAC-SHA-3 implementation: Execution time for first initialization with different key sizes is compared to a reinitialization using the same key again.

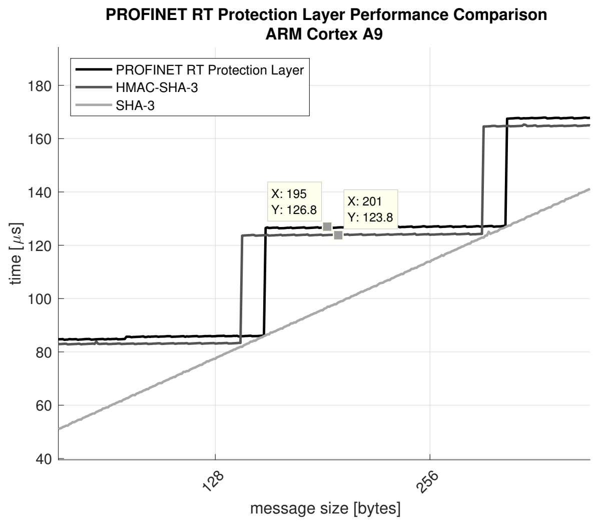

Fig. 7. The 3 µs processing overhead of the PROFINET RT protection layer is mainly caused by memory manipulation, the generation of additional fields and padding.

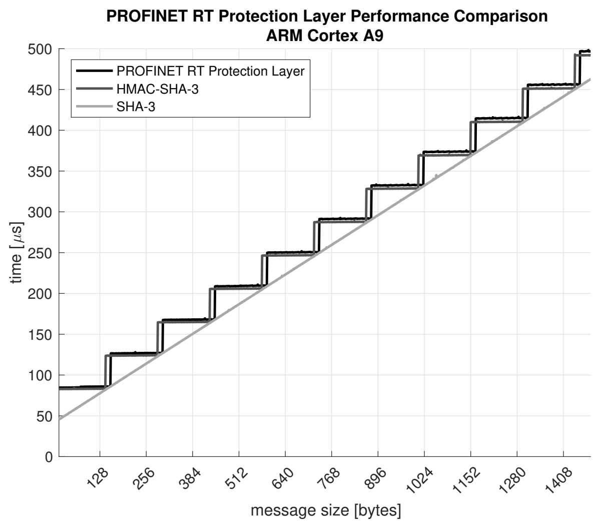

Fig. 6. Comparison of performance of the complete PROFINET RT protection layer to its subcomponents HMAC-SHA-3 and pure SHA-3.

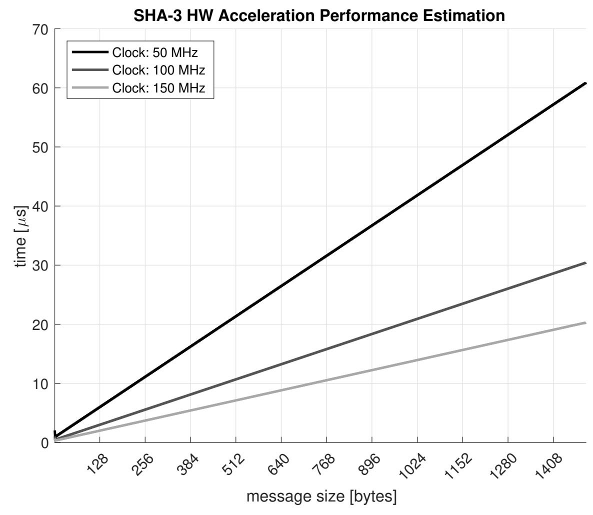

Fig. 8. Estimation of SHA-3 hardware acceleration performance: Depending on the clock speed, the SHA-3 performance can be estimated from the captured number of clock cycles needed in simulation.

plot of the protection layer performance is shifted 14 bytes with respect to the x-axis. This is caused by the Ethernet frame header (2 times MAC address and the EtherType), which is included in the length specification of the input message. Since we are interested in the suitability for high performance motion-control applications with typical payload sizes of 8 to 256 bytes and cycle times of 250 µs, 125 µs of it reserved for RT communication [17], we analyzed this part of the plot more precisely in Fig. 7. We can see that not even one single frame with a payload $\geq$ 144 bytes can be processed completely within the reserved RT bandwidth of a such a system with a minimal configured cycle time. If we assume a symmetrical setup, where an IO-Device transmits and receives one frame within a cycle, this would require the execution time of the protection layer to be less than half of the RT bandwidth, i.e. 62.5 µs. This statement assumes that execution times of the protection procedure (Alg. 1) and the verification (Alg. 2) are equal. Since the overhead for frame processing is just 3 µs (see Fig. 7), we rate this assumption as sufficient. This small overhead also lowers the priority for finding suitable optimizations on the processing (i.e. the generation of needed fields etc.), since most of the execution time is spent in cryptographic components of the protection layer.

B. Optimization Potential

The performance results of the software solution indicates that further optimization, especially on the cryptographic algorithms, is necessary to meet the strict timing requirements of high performance real-time automation systems. The most promising approach is outsourcing of cryptographic processing into the hardware, i.e. the programmable logic blocks. To estimate the potential speed up, we developed the SHA-3-224 algorithm in hardware description language and captured the number of clock cycles needed for processing of various length messages out of the testbench simulation. By multiplying this data with a suitable clock speed, we get a measure on the achievable execution times within a dedicated hardware core. Typical clock speeds of low cost FPGA platforms vary between 50-150 MHz, which we used to assess the comparison between the two implementations (Fig. 8). Depending on the clock speed, we achieve a speedup factor of 8 to 25. This would be sufficient to meet the timing requirements of systems with a minimum cycle time configuration of 250 µs.

IV. Conclusion and Further Work

In this publication the prototypal implementation of a PROFINET RT protection layer ensuring authenticity and integrity of real-time communication was presented. The chosen architecture of a transparent switch allows protection of frames on-the-fly. This device served as a platform for analyzing the performance with respect to very low cycle times of 250 µs. It could be shown that a protection layer purely software-based does not provide sufficient performance for the adoption into such systems, even under usage of a high optimization level during compile time. Nonetheless, the potential of compiler optimizations might not be fully exploited and this could be an interesting topic worth further investigation. To evaluate how much time is actually available for encryption, the CPU utilization when running a PROFINET stack combined with the protection layer must be studied by profiled on a per-device basis. We could show a potential performance increase of about factor 8 to 25 through the use of cryptographic hardware acceleration. This estimation can be compared to actual measurement results after successful integration of a PROFINET protection layer in an IO-Device. Further planned work includes the outsourcing of the complete HMAC component into programmable logic. The development of the transparent security switch will be continued, since besides for performance analysis, it can also serve as a verification and testing platform for device and stack vendors adopting their products to a future PROFINET security standard. This enables them to start the development process without being dependent on the progress of PROFINET master stack implementations.

Acknowledgment

The present study is a contribution to the PROFIBUS & PROFINET International working group CB/PG10 PN Security, founded to work out a security standard for PROFINET.

References

[1] "Das Honeynet-Experiment: Hackerangriffe auf virtuelles Wasserkraftwerk belegen Gefahren für Industrie 4.0," https://www.tuev-sued.de/management-systeme/newsletter/2015/4/das-honeynet-experiment-hackerangriffe-auf-virtuelles-wasserkraftwerk-belegen-gefahren-fuer-industrie-4.0, accessed: 2018-05-06.

[2] "PROFINET Security Guideline," Profibus Nutzerorganisation (PNO) e.V., Karlsruhe, Tech. Rep., Nov. 2013. [Online]. Available: https://www.profibus.com/download/profinet-security-guideline/

[3] M. Runde, C. Tebbe, and K. H. Niemann, "Performance Evaluation of an IT security Layer in Real-Time Communication," in *2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*, Sept 2013.

[4] B. Czybik, S. Hausmann, S. Heiss, and J. Jasperneite, "Performance Evaluation of MAC Algorithms for Real-Time Ethernet Communication Systems," in *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, July 2013.

[5] S. Hausmann and S. Heiss, "Usage of Public Key Infrastructures in Automation Networks," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012)*, Sept 2012.

[6] M. Runde, C. Tebbe, K. H. Niemann, and J. Toemmler, "Automated Decentralized IT Security Supervision in Automation Networks," in *IEEE 10th International Conference on Industrial Informatics*, July 2012, pp. 1234–1239.

[7] M. Runde, S. Hausmann, C. Tebbe, B. Czybik, K.-H. Niemann, S. Heiss, and J. Jasperneite, "SEC_PRO : sichere Produktion mit verteilten Automatisierungssystemen," Fakultät I - Elektro- und Informationstechnik, Tech. Rep., 2014. [Online]. Available: http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bsz:960-opus4-4995

[8] M. Popp and K. Weber, *Der Schnelleinstieg in PROFINET*. PROFIBUS Nutzerorganisation, 2004.

[9] R. Pigan and M. Metter, *Automatisieren mit PROFINET: Industrielle Kommunikation auf Basis von Industrial Ethernet*, 2nd ed. Publicis Corporate Publishing, Erlangen, 2008.

[10] T. Müller, A. Walz, M. Kiefer, H. D. Doran, and A. Sikora, "Challenges and Prospects of Communication Security in Real-Time Ethernet Automation Systems," forthcoming 2018 IEEE International Workshop on Factory Communication Systems Proceedings, June 2018.

[11] "ITU-T X.200 (07/1994) Information Technology Open Systems Interconnection Basic Reference Model: The Basic Model," International Telecommunication Union, Geneva, CH, Recommendation, Jul. 1994. [Online]. Available: https://www.itu.int/rec/T-REC-X.200-199407-I

[12] "ITU-T X.800 (03/1991) Security Architecture for Open Systems Interconnection for CCITT Applications," International Telecommunication Union, Geneva, CH, Recommendation, Mar. 1991. [Online]. Available: http://www.itu.int/rec/T-REC-X.800-199103-I/e

[13] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104, Feb. 1997.

[14] W. Stallings, *Network Security Essentials: Applications and Standards*, 5th ed., ser. Always learning. Pearson, 2013.

[15] "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," National Institute of Standards and Technology / Federal Information Processing Standards, Gaithersburg, Maryland, Standard, Aug. 2015. [Online]. Available: http://dx.doi.org/10.6028/NIST.FIPS.202

[16] E. B. Barker, W. Barker, W. Burr, T. Polk, M. Smid, and L. Zieglar, "SP 800-57. Recommendation for Key Management, Part 1: General (Rev. 4)," Gaithersburg, MD, United States, Special Publication, Jan. 2016.

[17] D. Gunzinger, C. Kuenzle, A. Schwarz, H. D. Doran, and K. Weber, "Optimising PROFINET IRT for Fast Cycle Times: A Proof of Concept," in *2010 IEEE International Workshop on Factory Communication Systems Proceedings*, May 2010, pp. 35–42.

[18] R. M. Stallman and G. DeveloperCommunity, *Using The Gnu Compiler Collection: For GCC Version 6.3.0*. Boston, MA: GNU Press, Free Software Foundation, 2016. [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc-6.3.0/gcc.pdf

[19] M. Taha and P. Schaumont, "Side-Channel Analysis of MAC-Keccak," in *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, June 2013, pp. 125–130.