# Towards Secure Fieldbus Communication

Felix Wieczorek[1], Christoph Krauß[2], Frank Schiller[1], and Claudia Eckert[3]

[1] Beckhoff Automation, Scientific Safety & Security
Ostendstraße 196, D-90482 Nuremberg
{f.wieczorek,f.schiller}@beckhoff.com
[2] Fraunhofer Research Institution AISEC
Parkring 4, D-85748 Garching
christoph.krauss@aisec.fraunhofer.de
[3] Technische Universität München
Boltzmannstraße 3, D-85748 Garching
claudia.eckert@in.tum.de

**Abstract.** In this paper, we present an approach to secure fieldbus communication of automation systems used in security-critical applications. We propose a protocol that applies a scheme combining a stream cipher and a Message Authentication Code (MAC) to ensure integrity, confidentiality, authenticity, and freshness of transmitted telegrams over a fieldbus while maintaining real-time constraints. The security discussion shows that the protocol is secure against an adversary attacking the fieldbus communication. A first proof-of-concept implementation for the EtherCAT fieldbus protocol is implemented to perform some initial runtime analyses.

**Keywords:** fieldbus, security, protocol.

## 1 Introduction

Industrial automation systems use fieldbus communication for real-time distributed control of systems such as water supply, energy distribution, or manufacturing. Security for fieldbus communication was not considered to be an important issue, since these systems were deployed typically in closed environments. However, since fieldbus installations become more and more automated and cross-linked, security becomes more and more important. For example, the cables of the fieldbus-connections in a wind park used to connect the wind turbines with a central control system can be accessed by an adversary since it is not possible to protect the whole area. If wireless fieldbuses [3] are used, attacks such as eavesdropping on the communication are even much easier for an adversary. Thus, security mechanisms have to be applied. To enable the compliance with real-time requirements as well as to provide transparent security to higher layers, security mechanisms have to be integrated into the fieldbus layer.

These security mechanisms have to protect the confidentiality of the fieldbus communication to prevent an adversary from eavesdropping to get sensitive information such as the temperature profile of a beer brewing process, which may

F. Ortmeier and P. Daniel (Eds.): SAFECOMP 2012, LNCS 7612, pp. 149–160, 2012.

be confidential intellectual property. Furthermore, the authenticity of the communication has to be ensured, i.e., the origin of data has to be genuine. This prevents an adversary from injecting false data such as false temperature settings in order to destroy the mash. To detect any unauthorized modification of data, mechanisms for integrity protection are required. The detection of replayed old data requires mechanisms to ensure the freshness of data. Finally, the availability of the communication should be ensured, i.e., all authorized entities have access to services or information as intended. However, this is hard to achieve since an adversary may always perform Denial-of-Service (DoS) attacks, e.g., by simply cutting the wire or by performing a jamming attack if wireless fieldbuses are used. Thus, security mechanisms should at least not influence the availability of the fieldbus communication during operation.

In this paper, we present a protocol to ensure the security goals integrity, confidentiality, authenticity, and freshness of telegrams transmitted over a fieldbus. The protocol is based on a scheme combining a stream cipher (SC) and a Message Authentication Code (MAC). It is designed in such a way that the used SC and MAC primitives can be easily substituted, e.g., if they become insecure or more efficient primitives have been developed. The protocol is able to maintain real-time requirements when integrated into a fieldbus as long as no attacks such as DoS are performed, i.e., it guarantees secure telegram transmission within defined boundaries. We discuss the security of our protocol and how it meets the security goals. In addition, we present a first proof-of-concept implementation for the EtherCAT fieldbus and some initial results of our runtime analyses.

## 2 Related Work

Since most fieldbus systems have been used in closed systems, only a few approaches are designed to provide security, e.g., [12] which is based on IEEE 802.15.4 [11]. Here, Block Ciphers (BCs) in CCM-mode are used, which are padded to full block length. This is a major disadvantage when many short telegrams are transmitted like in typical fieldbus communication. Adding security mechanisms such as IPsec for Internet Protocol (IP)-based fieldbuses is discussed in [18]. Introducing security mechanisms at higher levels is also discussed in [19]. Secure industrial communication using Transmission Control Protocol/Internet Protocol (TCP/IP) is addressed in [4], where the necessary reaction times of automation fieldbuses cannot be reached.

In the area of Building Automation Control (BAC), an approach for secure fieldbus communication is presented in [16] using Data Encryption Standard (DES) and Hashed MAC (HMAC) with SHA-1 on smartcards. In [7], the security of wireless BAC networks is discussed. BAC networks have smaller bandwidth and the presented solutions are not fast enough for general fieldbuses in automation, where the data rate is much higher and the real-time constraints tighter than in BAC applications.

In [17], a multicast authentication protocol for fieldbuses based on MACs is proposed. The focus is on automotive buses such as CAN. Mechanisms to provide confidentiality are not discussed.

The bus systems used in the automotive domain such as CAN, LIN or Flexray do not provide any security mechanisms [22,23]. In [22], an approach to secure automotive bus communication is proposed, where the communication always involves a central gateway as intermediary.

Some popular schemes that combine authentication and encryption on basis of block-ciphers are OCB [15] or EAX [2].

We decided to use stream ciphers, since they can be implemented in hardware easily to reach high performance and do not require padding of data. Stream cipher schemes, providing combination of encryption and authentication, are VMPC-MAC [25], Helix [6] and Phelix [20]. They create a Message Authentication Code (MAC) that depends on the specific design of the stream-cipher. However, Phelix and Helix are considered insecure [24]. Another interesting approach is ASC [21]. However, the receiver has to decrypt first and then check the integrity, which leads to more effort, than our approach, if telegrams are corrupted.

To the best of our knowledge, there exists no efficient security protocol which can be used in fieldbuses with high bandwidth and hard real-time requirements for telegrams with various length. Previously proposed approaches cannot be transferred without adaption, due to restrictions of the performance requirements. Thus, we developed a new approach which is based only on universal properties of stream ciphers enabling the use of well evaluated stream ciphers as well as an easy way to exchange them in case they become insecure.

## 3 Protocol Description

In this section, we describe our proposed protocol in detail. We first discuss the requirements we address. Then we describe our scheme to combine a SC with a MAC. Finally, we describe the protocol steps in detail.

### 3.1 Addressed Requirements

The protocol is designed to meet real-time requirements which are necessary in fieldbus communication. Real-time in this context means the guarantee of telegram transmission shorter than a fixed delay. This is commonly reached by cyclic communication, which also allows detection of lost telegrams. The worst-case execution time of all security algorithms, which is relevant for the fieldbus performance, has to be limited to a fixed upper bound. Our proposed protocol meets this requirement in the regular operation phase, which is usable as long as all telegrams are correctly transmitted, by using only algorithms with deterministic runtime. The tasks of the (not real-time capable) initial phase need to be accomplished once, to distribute trust-anchors and keys, afterwards the key-exchange has to be carried out periodically in maintenance intervals where real-time constraints do not apply. Hybrid techniques similar to this one are widely in use.

Furthermore, our protocol ensures authenticity, integrity, freshness, and confidentiality of the fieldbus communication assuming an active attacker attacking the fieldbus communication. Availability is not considered, since protection against an active attacker is usually not possible.

An important design principle of our protocol is the exchangeability of the used Stream Cipher (SC) and Message Authentication Code (MAC) primitives and adaptability of security levels. If an underlying primitive becomes insecure during the long life-time of automation systems, easy substitution is required to fix those systems.

### 3.2 Generic SC and MAC Scheme

The generic SC and MAC scheme (cf. Figure 1) uses two distinct parts of the output of only one SC. One part is used for encryption, the other as input of a MAC scheme. We assume that the underlying MAC construction and the SC are secure and have deterministic runtime.

The inputs of the scheme are

- payloads $pl(0..n)$, all of the same fixed length ($|pl(i)| = |pl(j)| \; \forall \, 0 \leq i, j \leq n$),
- a key $k$ and
- an initialization vector $iv$.

The outputs of the scheme are

- ciphertexts $c(0..n)$, and
- integrity protecting tags $mac(0..n)$

where each $(c(i), mac(i))$ pair corresponds to one payload $(pl(i))$.

Whenever the SC outputs a cipher-stream of the length $|otp_{enc}| + |otp_{mac}|$, the internal state of the SC is updated ($s(i+1) = f(s(i))$). The initial state is derived from the key $k$ and the initialization vector $iv$ ($s(0) = f_{init}(k, iv)$). The cipher-stream is partitioned into the two parts $otp_{mac(i)}$ and $otp_{enc(i)}$, which serve as inputs of the MAC and the encryption algorithms, respectively.

Note that usually different keys are used for different purposes, e.g., one key for encryption and one key for the MAC. Using one key could enable an attacker to get information from a possible relation between the ciphertext and the MAC [13, pp. 418, 567]. The disadvantage of using multiple keys is the additional overhead for key distribution and storage. However, when carefully designed, a cryptographic scheme can still be secure although only one key is used. Examples are Grain-128a [27,26] and CCM mode for BCs [5].

### 3.3 Protocol Steps

In this section, we describe the protocol steps in the two phases of our protocol, i.e., initialization and operational phase.

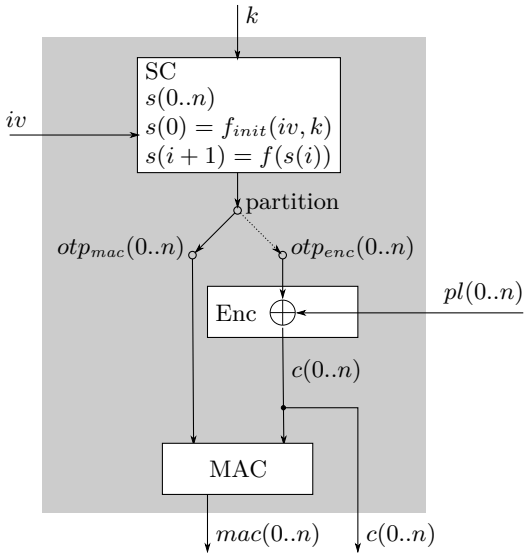


**Fig. 1.** Concept of the generic SC and MAC scheme (sender)

**Initialization Phase.** This phase does not require real-time, therefore the use of asymmetric cryptography is possible. During this phase, trust is established, parameters, such as cipher choice, key- and MAC-length, are negotiated and keys are exchanged. The key-exchange of the communicating parties has to be triggered in advance by one party knowing the network topology, which is usually the master. A Diffie-Hellman key-exchange using trust anchors for authentication can be used as key-exchange protocol.

The SCs of every party are initialized using the exchanged key (and other parameters) and $iv := 0$, resulting in same states $s_0$ of the SCs.

**Operational Phase.** In this phase, real-time restrictions apply. If the algorithms and state-updates of the security layer run in real-time, the secure transmission of data is done in real-time itself, because the underlying fieldbus provides real-time transmission of telegrams. All protocol steps are based solely on symmetric algorithms to have a short runtime.

*States of a Participant.* Each party has to keep a state per communication relationship consisting of:

- secret key $k$,
- current $iv$,
- current state $s$ of the SC,
- fixed payload length $|pl|$,
- fixed MAC length $|mac|$, and
- maximum retries of windowing $w_{max}$.

During resynchronization, the following additional variables are required:

- temporary state $s^*$,
- counter for telegrams not correctly verified $w$, and
- temporary initialization vector $iv^*$.

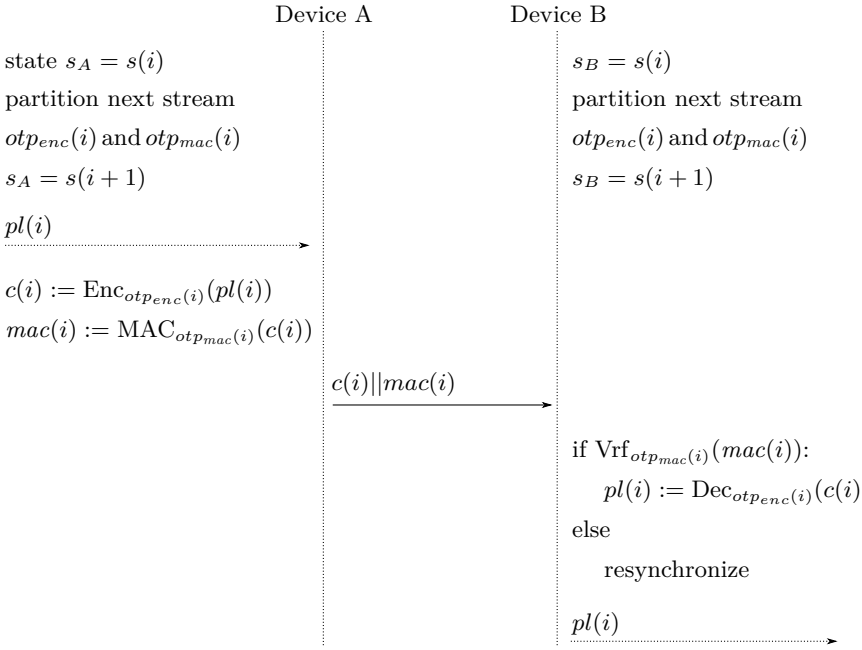


**Fig. 2.** Regular real-time operation

*Regular Operation.* The regular operation is sketched in Figure 2. All parties share the same state of the SC. Since each payload has the same length, the execution of the security algorithms consumes the same amount of cipher-stream for each payload. Given the actual state $s(i)$ of the SC, each successor state $s(j)\,(j > i)$ and the corresponding $otp_{mac}$ and $otp_{enc}$ are computable in advance without knowing the payloads (cf. Figure 1).

Each time a payload $pl$ is passed to the security layer, the cipher-stream is first used as $otp_{enc}$ for encryption and afterwards as $otp_{mac}$ for integrity protection. A ciphertext is build with the encryption algorithm Enc (which computes $c := otp_{enc} \oplus pl$). Then the MAC secures the authenticity and integrity of the ciphertext consuming $otp_{mac}$. The telegram transmitted over the fieldbus consists of the ciphertext concatenated with the *mac*.

The receiver uses the same cipher-stream, thus resulting in the same state as the sender. It first checks the correctness of the *mac* with the verification algorithm Vrf. If the *mac* is verified successfully, the ciphertext is decrypted (by

using bitwise XOR with the same $otp_{enc}$ the sender had used to encrypt the ciphertext). The resulting payload is then passed to the application.

Regularly, the initialization vector is changed due to a specific schedule (e.g., every fixed number of telegrams). If the states of the communication parties are no longer synchronized, mechanisms for resynchronization are required. At a data rate of 100 Mbit/s of the fieldbus, the SC Grain-128a, which we used in our implementation, can be used with one initialization vector over a typical operational uptime, so renewing the initialization vector is only necessary if synchronization is lost.

*Resync.* When a telegram is lost, resynchronization is needed, since the states of the SCs are not equal anymore. The resynchronization is not real-time capable, just as it would be in plain fieldbus communication without the security-layer. The asynchronous state is recognizable by an unsuccessful verification of the *mac*. Because of this, an asynchronous state is not distinguishable from a manipulated telegram. If only a few telegrams were lost, it is possible, due to the fixed telegram length, to resynchronize only the receiver.

The Resync with windowing resynchronizes the receiver after a few lost messages. If messages are lost, the receiver tries to catch up by verifying the received telegram with the following stream-cipher states as temporary states. This mechanism is limited to a predefined number of retries. An example is given in Figure 3. Both parties A and B share the same SC state $s_0$ in the beginning. One telegram gets lost during transmission, the next payload is passed to A in state $s_A = s_2$, resulting in the correctly transmitted telegram $tel_2 = c_2||mac_2$. This telegram is received by B in state $s_B = s_1$, not corresponding to the senders state, and not verified correctly by B. B enters the windowing mechanism and steps one state forward temporarily, correctly verifying the integrity of $tel_2$. A posteriori, the one lost message is detectable. The temporary state is applied, resulting in synchronous states of both parties, again.

If the number of lost telegrams exceeds the limit of windowing, interaction between the receiver and the sender is required. After the windowing has failed, the receiver determines a fresh initialization vector, by incrementing the current initialization vector. Then the SC is initialized with the current secret key and the fresh initialization vector, resulting in a new defined state. The initialization vector and a SyncLost notification is sent unencrypted but integrity protected to the other party, which can reach the same state with the knowledge of the initialization vector and verify the integrity of the telegram subsequently. The resync mechanism is shown in Figure 4.

## 4 Security Discussion

In this section, we discuss the security of our proposed protocol, i.e., how it protects the integrity, confidentiality, authenticity, and freshness of the transmitted messages. We assume an active adversary with access to the whole communication who can try to inject, eavesdrop, replay, drop, delay, or manipulate any

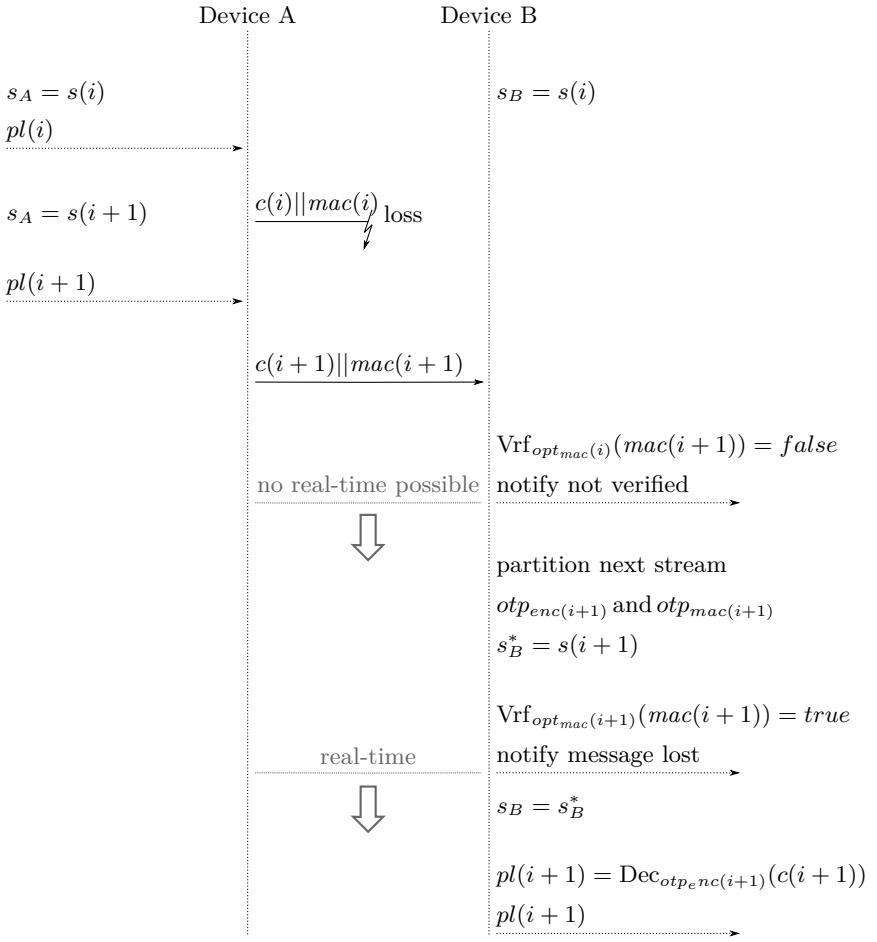


**Fig. 3.** Resynchronization with windowing

telegrams. However, we assume that the adversary has no access to the automation devices and to any stored data such as the cryptographic keys. This is a reasonable assumption since automation devices are expected to be mounted in physically secured installation environments. If that is not the case, i.e., systems could be compromised, additional mechanisms have to be taken into account. For example, the automation systems could be equipped with hardware security modules such as smartcards, which provide secure storage for cryptographic data, a secure execution environment for (cryptographic) computations, and often the support for additional features such as secure boot or remote attestation.

In the following, we assume that the used stream cipher and the MAC are each secure, i.e., an adversary can neither decrypt messages encrypted with the stream cipher nor forge valid MACs without knowing the cryptographic

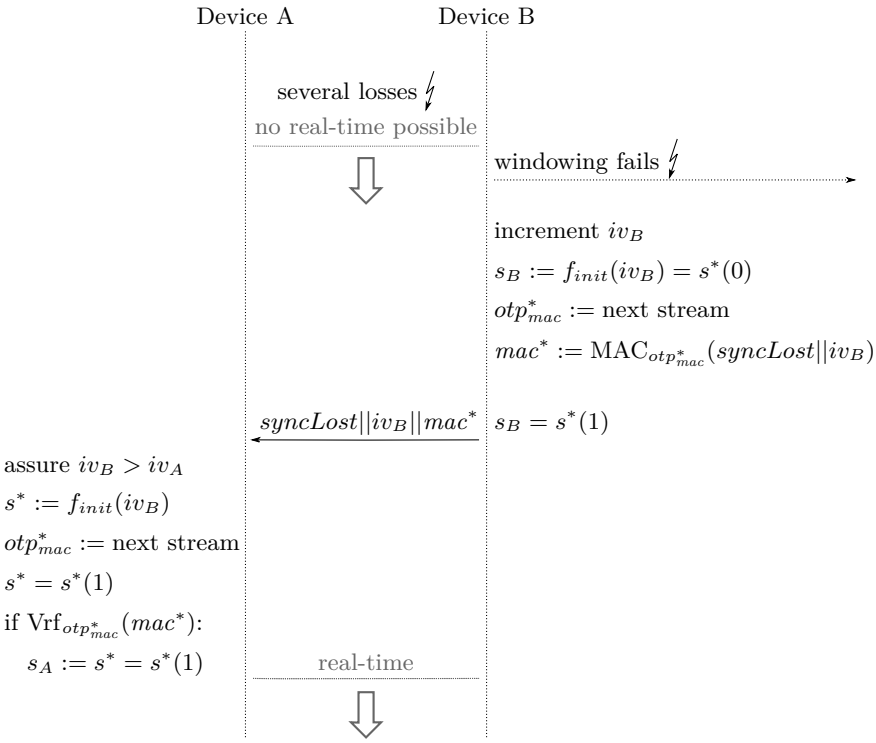


**Fig. 4.** Resync with $iv$

key. Since the basis of our protocol is the proposed generic construction, we show that using this construction is also secure. First, we show that given any set of pairs $(c(0..n), mac(0..n))$, where $c(i) = pl(i) \oplus otp_{enc}(i)$ and $mac(i) = \mathrm{MAC}_{otp_{mac}(i)}(c(i))$ for payloads $pl(i)$ $0 \leq i \leq n$, an adversary cannot get information about any $pl(i)$ by eavesdropping these pairs. Second, we show that an adversary is not able to forge a valid pair $(x, mac(i)) = (x, \mathrm{MAC}_{otp_{mac}(i)}(x))$ for any arbitrary binary string $x$.

In the first case, the generic construction loses its security properties if two different messages are ever encrypted with the same cipher-stream. Thus, $otp_{enc}(i)$ has to be different for each $pl(i)$. To achieve this, the SC changes the state for each transmitted telegram. The initial state $s(0)$ is calculated using the initialization vector $iv$ and key $k$: $s(0) = f_{init}(iv, k)$ and all subsequent states are calculated according to $s(i+1) = f(s(i))$. Each state results in a different output which is partitioned into $otp_{mac}(i)$ and $otp_{enc}(i)$. When the scheme is reinitialized, a new $iv$ is used by incrementing the old one. Assuming the size of the $iv$ is carefully chosen to prevent overflows, an $iv$ is only used once. Thus, for each $pl(i)$ always a different $otp_{enc}(i)$ is used. This reduces the security of the scheme up to the security of the used stream cipher. Since we assumed that the stream cipher is

secure and protects the confidentiality of the transmitted messages, this is also true for the generic scheme.

In the second case, the security of the MAC is solely based on the secrecy of the used key since we assumed that the used MAC construction is secure. Thus, an adversary can only forge a valid $(x, mac)$ pair if he can derive the key $k$ or the correct $otp_{mac}(x)$. However, since we assumed the applied stream cipher is secure, an adversary neither can get both of them by analyzing eavesdropped pairs $(c(0..n), mac(0..n))$.

Thus, an adversary cannot successfully eavesdrop on telegrams or inject new telegrams. Furthermore, an adversary cannot successfully replay telegrams, since the freshness is ensured by changing the internal state after each correctly verified telegram. Likewise the dropping of telegrams is detected and a resynchronization initialized.

## 5 Implementation and Runtime Analysis

In our implementation, we have used Grain-128a [27] with 128 bit key and 96 bit initialization vector as underlying stream cipher of the generic scheme. The Grain-128a cipher is based on the well-analyzed cipher Grain [8] which can be easily implemented in hardware, provides high performance, and has deterministic runtime. As MAC, we have chosen the Toeplitz matrix based approach, which is easy to implement in hardware and also has deterministic runtime. We chose a MAC length of 80 bit which provides a reasonable security level for most applications. The telegrams are embedded as process data in regular EtherCAT telegrams [10].

We have developed a corresponding prototype software implementation in C [14]. The implementation is currently not optimized for speed. In future applications it might be possible to run the security stack in hardware in order to reach higher performance. The master and slave were both running on the same Microsoft Windows XP Professional SP3, Intel Core2Duo T7400@2.16 GHz, 2 GB RAM machine during the runtime measurements. This configuration resembles widely used IPC. The slave controller is a Beckhoff FC1100 card [1]. For the proof-of-concept implementation, EtherCAT was used in a synchronous mode, triggering the slave application to run once on each incoming datagram. The master can not be executed in hard real-time (this is only possible with programming languages defined in [9]), as a replacement for the missing real-time capabilities, the multimedia timer of Microsoft Windows was used to achieve a de-facto cycle-time of 1 ms.

The first measurements show, that the security layer only generates negligible overhead, at a cycle-time of 10 ms, the non-secure slave application runs in 7 µs, compared to the execution time of 8 µs of the slave application with enabled security layer. The execution times for resynchronization are not significantly longer. Those measurements do not consider the transmission-time overhead of the MAC. More extensive measurements, also at shorter cycle-times, will be part of future work.

## 6 Conclusion

In this paper, we presented a protocol to secure the fieldbus communication of automation systems while maintaining real-time requirements. The basis of the protocol is a generic scheme which combines a stream cipher with a MAC to ensure integrity, confidentiality, authenticity, and freshness of transmitted messages using only one key for cipher and MAC to facilitate key management. The scheme relies solely on symmetric primitives, which are much more efficient than asymmetric primitives, to support the use in resource-constrained systems as well as to enable small cycle times for real-time communication. We chose a stream cipher since they typically execute at a higher speed than block ciphers and have lower hardware complexity. The security of our protocol relies on the security of the used stream cipher and MAC construction. By adjusting the key length, the protocol can be adapted according to the application requirements. Our proof-of-concept implementation and the first results of our performed performance analysis have shown the feasibility of our approach. As future work, we plan to implement the protocol in hardware and perform more detailed performance analyses. Another future topic is to provide exchangeability of SC and MAC in the prototype.

## References

1. Beckhoff Automation GmbH: FC1100 | PCI EtherCAT slave card (2011)
2. Bellare, M., Rogaway, P., Wagner, D.: The EAX Mode of Operation. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 389–407. Springer, Heidelberg (2004)
3. Brühne, M.: IEEE 802.1n und WLAN-Controller – Lohnt der Einsatz auch in der Industrie. In: SPS/IPC/DRIVES: Elektrische Automatisierung, Systeme und Komponenten (2011)
4. Damm, M., Leitner, S.H., Mahnke, W., Leitner, S.H.: Security. In: OPC Unified Architecture, pp. 1–51. Springer (2009)
5. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. NIST Special Publication 800-38C, NIST - Computer Security Resource Center (2007)
6. Ferguson, N., Whiting, D., Schneier, B., Kelsey, J., Lucks, S., Kohno, T.: Helix: Fast Encryption and Authentication in a Single Cryptographic Primitive. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 330–346. Springer, Heidelberg (2003)
7. Granzer, W., Reinisch, C., Kastner, W.: Future Challenges for Building Automation: Wireless and Security. In: Proc. IEEE Int Industrial Electronics (ISIE) Symp., pp. 4415–4467 (2010)
8. Hell, M., Johansson, T., Meier, W.: Grain – A Stream Cipher for Constrained Environments. International Journal of Wireless and Mobile Computing, Special Issue on Security of Computer Network and Mobile Systems 2(1), 86–93 (2006)
9. IEC: IEC 61131-3, Programmable controllers — Part 3: Programming languages, 2 edn. (2003)
10. IEC: IEC 61158, Industrial communication networks — Fieldbus specifications, 2 edn. (2010)

11. IEEE: IEEE 802.15.4, Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs) (2006)
12. ISA: ISA100.11a Wireless systems for industrial automation: Process control and related applications (2011)
13. Menezes, A.J., Vanstone, S.A., Oorschot, P.C.V.: Handbook of Applied Cryptography. Discrete Mathematics and Its Applications, 5th printing edn. CRC Press, Inc. (1996)
14. Microsoft Corporation: Microsoft Visual Studio, Ultimate, version 10.0.4.0129.1 SP1Rel (2010)
15. Rogaway, P., Bellare, M., Black, J.: OCB: A block-cipher mode of operation for efficient authenticated encryption. ACM Trans. Inf. Syst. Secur. 6, 365–403 (2003)
16. Schwaiger, C., Treytl, A.: Smart Card Based Security for Fieldbus Systems. In: Proc. IEEE Conf. Emerging Technologies and Factory Automation ETFA 2003, vol. 1, pp. 398–406 (2003)
17. Szilagyi, C., Koopman, P.: Flexible Multicast Authentication for Time-Triggered Embedded Control Network Applications. In: DSN, pp. 165–174. IEEE (2009)
18. Treytl, A., Sauter, T., Schwaiger, C.: Security Measures for Industrial Fieldbus Systems – State of the Art and Solutions for IP-based Approaches. In: Proc. IEEE Int Factory Communication Systems Workshop, pp. 201–209 (2004)
19. Treytl, A., Sauter, T., Schwaiger, C.: Security Measures in Automation Systems – a Practice-Oriented Approach. In: Proc. 10th IEEE Conf. Emerging Technologies and Factory Automation ETFA., vol. 2, pp. 847–855 (2005)
20. Whiting, D., Schneier, B., Lucks, S., Muller, F.: Phelix Fast Encryption and Authentication in a Single Cryptographic Primitive. Tech. rep., ECRYPT Stream Cipher Project Report 2005/027 (2005)
21. Wirt, K.T.: ASC – A Stream Cipher with Built–In MAC Functionality. World Academy of Science, Engineering and Technology 29 (2007)
22. Wolf, M., Weimerskirch, A., Paar, C.: Security in Automotive Bus Systems. In: Proceedings of the Workshop on Embedded Security in Cars, ESCAR 2004 (2004)
23. Wolf, M., Weimerskirch, A., Wollinger, T.: State of the Art: Embedding Security in Vehicles. EURASIP Journal on Embedded Systems (2007)
24. Wu, H., Preneel, B.: Differential-Linear Attacks against the Stream Cipher Phelix. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/056
25. Zoltak, B.: VMPC-MAC: A Stream Cipher Based Authenticated Encryption Scheme. In: Fast Software Encryption, Springer, Heidelberg (2004)
26. Ågren, M., Hell, M., Johansson, T.: On Hardware-Oriented Message Authentication with Applications towards RFID. In: Lightweight Security Privacy: Devices, Protocols and Applications (LightSec), pp. 26–33. IEEE Computer Society (2011)
27. Ågren, M., Hell, M., Johansson, T., Meier, W.: A New Version of Grain-128 with Authentication. In: Symmetric Key Encryption Workshop. European Network of Excellence in Cryptology II (2011)