Integration Approach for Communications-based Train Control Applications in a High Assurance Security Architecture*

Thorsten Schulz, Frank Golatowski, and Dirk Timmermann

Institute of Applied Microelectronics and CE, University of Rostock, Germany
{thorsten.schulz,frank.golatowski,dirk.timmermann}@uni-rostock.de

Abstract. The secure integration of model-based, safety-critical applications implemented in the programming suite Ansys SCADE is explained with the help of a demonstrator. The interoperability between the embedded devices of the demonstrator is achieved using the new TRDP middleware. Remote connections are secured using the WireGuard secure network channel. The demonstrator security concept addresses the different life cycles of its heterogeneous components by adoption of the robust MILS separation architecture. The goal of this open demonstrator is to show how these essential technologies can be composed to a secure safety-critical system.

Keywords: MILS · security · formal modeling · railway · CPS.

1 Introduction

From a systematic viewpoint, an application or device in a distributed system has inputs, the application logic and outputs. For a physical system, the application is hosted on hardware, typically abstracted from the application with a hardware abstraction layer and an operating system. In an interoperable, networked system, the inputs and outputs and their connection fabric are abstracted by a middleware using a standardized network protocol. Networks can range from dedicated, local connections to remote connections over untrusted networks. Untrusted access must be secured with adequate measures, defined by norms such as IEC 62443 or Common Criteria. Depending on the requirements of the application, certain levels of security assurance require different security measures. Railway devices must be maintainable with regards to EN 50126. Even on a dedicated network, they must secure their maintenance access and update features. The update/maintenance provider may not part a certified of the system.

This puzzle of heterogeneous technologies potentially induces three pitfalls: different life cycles, mismatching interfaces, complex integration. In this paper,

we describe our approach for an educational technology demonstrator that addresses these issues by applying the Multiple Independent Levels of Security (MILS) architecture on a separation kernel [11][14].

A real-world device is based on technologies, e.g., network protocols and hardware, with shorter life cycles compared to its core application logic. Different levels of modularity and a separation architecture are required to achieve a composed system certification. Such a separation architecture with different levels of security and safety can sustain security patches or updates to components without invalidating the safety certification of the application. This paper describes the security architecture in the next chapter 2, showing its feasibility with the help of a demonstrator in chapter 3.

In comparison to earlier approaches of our OpenETCS[1] On-Board-Unit (OBU) demonstrator [6], we have upgraded a basic middleware implementation with the new IEC 61375 standard based reference implementation of the Train Real Time Data Protocol (TRDP) from the TCNopen consortium. The adaptations and extensions to the TRDP Light library, we deemed necessary for a smooth integration with the critical application's interfaces, will be explained in chapter 4. Within the MILS architecture, the demonstrator also applies a novel, state of the art encryption mechanism with minimal additional overhead for embedded systems. We have selected this data encryption tunnel for communication between, in terms of the use-case, "non-local" devices. The motivation will be discussed in chapter 5.

2 Security Architecture for Critical Systems

Railway applications are required to be reliable, available, maintainable and safe (RAMS) according to accepted and governing standards, in Europe EN 50126. The properties are a result of qualified processes and guided methods, requiring specially trained engineers, operators and maintainers to minimize application risks. The processes specifically require that access is limited to that qualified and authorized group to assure the integrity of the processes and the system. Physical access barriers, e.g., locked cabinets, typically have a constant ratio between cost of securing and effort to bypass, largely due to the required physical attendance of the intruder with the specific knowledge to that barrier.

The introduction of networked or remote access to critical systems as Cyber-Physical Systems (CPS), in principle, has not changed this paradigm, but removed the latter physical appearance of an intruder. This has introduced negative scaling effects making even well secured systems with only a small security vulnerability cheap for large-scale attacks. The current mitigation trend in IT systems is to automate and improve testing methods, and to shorten time to update, i.e., patch vulnerabilities. In contrast, critical systems have stricter update policies and typically run on non-standardized hardware. Consequently, testing

[1] ETCS, European Train Control System. OpenETCS was a research project fostering an open reference implementation.

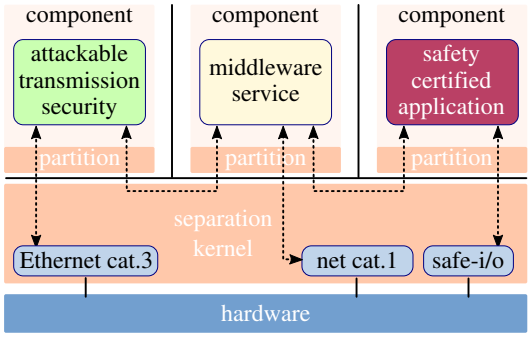

Fig. 1. Architecture of a base MILS system with a separation kernel, a safety application and additional services. Components like the security protocol (green) and the middleware must be instantiated separated to assure non-interference and correct behaviour of the safety application.

requires more effort for a much smaller number of operative products. Modifying a critical system's software requires re-certification – even for security patches.

Current research is developing methodologies to reduce the fore-said re-certification effort through dependable partitioning of a system, applying the Multiple Independent Levels of Security (MILS) architecture (Fig. 1).

The demonstrator's system design splits the system into a safe control component, a communication middleware and an independent security component with remote authorization, authentication and encryption. The safety function within the control application is unaffected of corruptions within the transmission component, if it can continue operation in degraded mode without that transmission data, e.g., safely issue an emergency command. The data flow between the components is guarded by the MILS separation kernel, see Fig. 1, allowing only predefined data flows between the partitions, i.e., application domains. This approach is also covered by IEC 62443 as conduits connecting zones [9]. Depending on attack vectors, system and application design, the security relevant transmission components could then also be of lower software confidence level and classified with a low Software Safety Integrity Level (SSIL), being less susceptible to re-certification requirements.

The separation kernel (SK) of a MILS system connects and controls the components within its partitions. There are different kinds and implementations of SK. For our demonstrator's OBU we use Sysgo's PikeOS. A different approach following the same MILS architecture is implemented in Thales' TAS platform [7] building on a Linux KVM-hypervisor setup for security. The kernel security development process follows different strategies to prove high assurance assumptions claiming interference-free execution of the composed system. Some rely on rigorous testing, using different testing techniques such as systematic and robustness testing (fuzz-testing), others rely on formal methods and a mixture of these techniques [12].

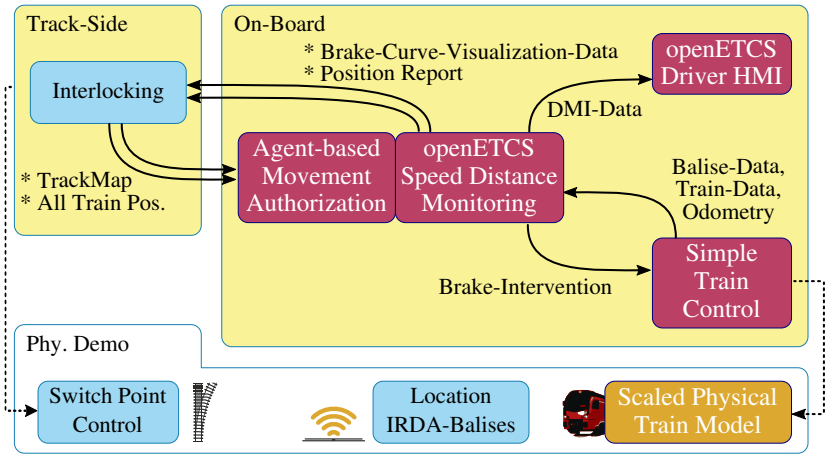

Fig. 2. Demonstrator concept view. On the left, the trackside control device has a GUI and is connected to the demonstrator model. On the right are the On-Board components, which are instantiated for each train.

An additional security measure that is applied within the demonstrator design is a cryptography data tunnel. This component can ensure the CIA properties of confidentiality, integrity and authenticity of external data access via untrusted networks (i.e., IEC 62443 cat. 3). Most state-of-the-art encryption protocols provide these properties. However, they strongly differ in algorithmic and integration complexity. Some technology choices are less desirable for carrying process data of safety-critical systems. We will discuss our choice in chapter 5.

In the next chapter, we will introduce the system structure of the demonstrator and its applications. The safety-critical application has a longer life cycle compared to a network-connected middleware or encryption tunnel. These need to be able receive regular security patches, which is supported by the MILS architecture. Therefore, we also separate the middleware described in chapter 4.

3 Demonstrator Overview

The demonstrator show in Fig. 2 has three parts: The trackside control application, termed "interlocking". Then, for each train, the on-board components and last, but not least, the physical demonstrator with model-trains. Larger parts of the on-board application logic (see dark-red boxes in Fig. 2) are implemented using the programming language Scade for safety-critical systems. Scade is an extension of Lustre, a synchronous dataflow programming language for reactive systems. It is graphically defined in the ANSYS SCADE Suite modeller. A code generator creates C code for compilation into executable binaries. The advantages of Scade are proofs towards causality errors, graphical verification,

immanent bounds checking and well defined behaviour. There exist other viable environments for critical implementations. However, the project decision for Scade was based on the existing OpenETCS models.

The demonstrator use-case is a simplified distributed Communications-based Train Control (CBTC) application: It consists of an interlocking, movement authorization, movement supervision (OpenETCS), a driver-machine interface (OpenETCS) and a train control with movement simulation or emulation on the model railway. While the implemented approach is applying state-of-art technology with real-world performance requirements, it is nevertheless, a technology showcase with an uncommon use-case – the *distributed* calculation of the train's movement authorization. Nonetheless, the goal is to integrate practical components in a MILS system for evaluation and education.

The dataflow is found in Fig. 2. The interlocking GUI application controls the physical switches of the demonstrator model and communicates the current track segments to all train instances. It also receives and broadcasts all train positions. The "On-Board" devices are instantiated for each train, for both, simulated or connected to a physical model train. The devices are a train control instance, an ETCS Driver HMI (DMI) and the ETCS On-Board-Unit (OBU). All these on-board applications are implemented as a Scade model. However, for simplification of the demonstrator, the OBU application uses a subset of the OpenETCS OBU with only the core module for Speed and Distance Monitoring (SDM, [15]). The SDM reads a movement authorization, which is the reserved track up to the next stopping point, and a track segment list with speed limits, ascent-profile and special prohibitions. With this data, the SDM calculates safe speed limits that need to be obeyed at upcoming locations. This data is sent back to the interlocking GUI. The GUI captured Fig. 3 visualizes the ETCS data for the demonstrator. If the derived deceleration curves are crossed, they trigger brake commands, which are sent as intervention commands to the train control. In a real ETCS application the movement authorization and the track information is received from track data elements (balises), as well as the radio block centre (RBC) via a mobile communication channel. In our demonstrator, each train receives only the track map with the current switch settings and the locations of all trains (Fig. 2). The on-board application generates the necessary data: the safe movement authorization and the track atlas, for a safe movement of the train on the demonstrator.

The following sections focus on the integration of the SDM + movement authorization application together with the networking components within a MILS system.

4 Lightweight Middleware

A middleware is required to connect the interfaces of separate applications. The most basic communication channels exist on the same hardware as Inter-Process-Communication (IPC) techniques like message queues and shared memories with signalling. However, basic IPC are not regarded as middleware, as they are pro-

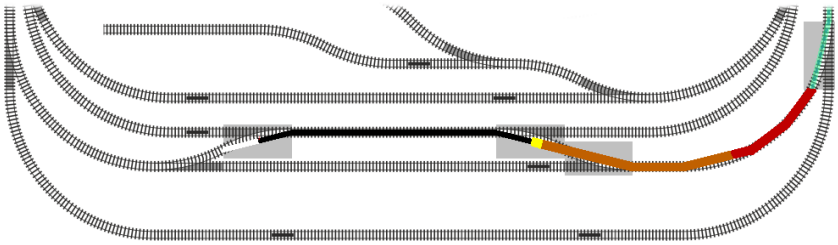

Fig. 3. GUI output of the interlocking application. It also displays the brake intervention locations received from each train. Here, due to the speed limitation imposed by the upcoming switch, ETCS brake-target operation is activated, showing warning (org.) and intervention (red).

vided and governed by the operating system kernel. Strictly defined IPC are also one of the key features of the MILS system architecture, which will be discussed in the following chapter. Middleware typically abstracts application's interfaces in a standardized way, independent of the underlying operating system or hardware architecture to enable interoperability. For example, in the demonstrator setup (Fig. 2), the OpenETCS component Speed and Distance Monitoring passes a uniform data structure to the DMI, which is connected via Ethernet. The first approach copied the output of one into a UDP packet and sourced it to the ScadeDisplay DMI application on the other side. This worked, as long as both ran the same hardware, same OS, same compiler, etc. When we changed one device a from 32 to a 64 bit CPU architecture, the memory addresses of the fields within the data structure changed due to specific memory alignments. This is a typical issue addressed by middleware referred to as "marshalling". We will return to this later in this section.

Middleware nowadays are based on web services, XML, SOAP and other service-oriented architectures. These are often not targeted for real-time applications on embedded, resource-constrained systems. More fitting alternatives for machine-to-machine communications in industrial automation (IIoT) are CoAP, MQTT, DDS and OPC UA. Despite its tremendous success in factory process automation, OPC UA has lately been criticized for its standard's complexity leading to non-interoperable implementations. Beyond these open protocols, there are also many proprietary industrial Ethernet technologies, like ProfiNET, EtherCAT, SERCOS III, TTEthernet, which can provide safe data transmission up to SIL 3, but this is out of scope of this demonstrator.

The railway industry has recently standardized a network protocol tailored towards efficient on-board process communication, the Train Real Time Data Protocol (TRDP, IEC 61375-2-3 [1]). It is intended to replace vendor proprietary solutions based on legacy buses or numerous incompatible custom solutions. It implements pull requests and cyclic push messages, as well as filtering based on a publish-subscribe scheme. The accompanying TCN standards (Train Communication Network) also define discovery, topology and direction based services.

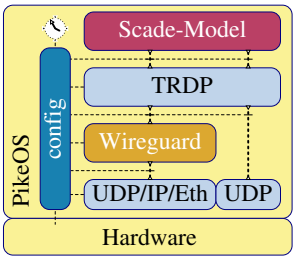

Fig. 4. Overview of the combined technologies in the OBU device of the demonstrator. All SW components are separated into partitions (see Fig. 1). The configuration for each component, the scheduling setup and the regulated information flow is sourced from a coordinated config pool.

The payloads are predefined, immutable binary data structures. This makes it a perfect fit interfacing with the Scade applications, which, in the generated code, provide bare C-structures.

For the demonstrator, we used the open TRDP Light from the TCNopen consortium. It provides a reference implementation of the core functionality of TRDP. The Light implementation also provides the XML configuration functionality ([1] Annex C), for memory, process and message ("telegram") configuration for TRDP. Typically, TRDP functions are linked from a library to the application code. However, following the MILS separation approach we integrate TRDP as a generic component in our demonstrator. It obtains its specializing configuration from the embedded operating system's (here PikeOS) central data provider, see Fig. 4.

For transferring the inputs and outputs of the Scade application component with the TRDP component, we use a small shared-memory area. To ensure synchronicity between the Scade application's interface and TRDP's related telegram definition, we implemented a small "type bridge" tool that converts between the data model descriptions. Coming back to the motivation of this chapter, TRDP Light also provides dataset marshalling based on the provided XML configuration. Different memory alignments and architecture endianness (big endian vs. little endian) are taken care of. The IEC 61375 standard defines 16 basic types, considering character, integer, float and time types with different bit-widths and, for integers, signedness. These data types can be combined to custom dataset structures making up a telegram.

However, we could not use the stock marshalling function and had to implement a refined version. This is due to the OpenETCS applications being implemented in a former version of Scade (6.4) that is limited to a single integer flow type, e.g., int32 *or* int64. Our modified marshalling function can be configured to take these type mappings into account, i.e., inflate an incoming int8 to the Scade-defined type and vice-versa. This approach maybe also required for other specialized programming environments.

To transport safety-critical SIL-2 process payloads, the Safe Data Transmission extension ([1] Annex B) has to be used. The application-to-application safety channel needs implementation of the safety code (checksum) calculation in the same safety context as the application, hence in Scade. This is, however, still work in progress. Security assurance is achieved by having TRDP as a separate component as discussed. The MILS approach simplifies security patching in case of a discovered weakness in the network protocol.

5 Secure Data Encryption Tunnel

Network traffic encryption for security can be applied on different layers of the common ISO-OSI (Open Systems Interconnection model). The tunnelled traffic of security protocols ranges from data link layer (2) up to the application layer (7). When the data link layer, such as Ethernet, can be abstracted, as well as application specific higher schemes are out of scope, OSI layer 3 layer encryption (L3-VPN) is the most versatile choice to be used as a component. The section will thus look at the choice of the Transport Layer Security protocol (TLS) as used in OpenVPN, as well as the WireGuard protocol, a VPN implementation using one specific configuration of the Noise protocol. Since IPsec is of even greater complexity than TLS, according to evaluation by Schneier and Ferguson [5], we do not consider it here. However, partner projects dealing with Distributed MILS (D-MILS) approaches have well analysed this technology for deterministic networking, see their results in [8]. Algorithms used within L3-VPNs are also known from tunnel specific higher-level application protocols. For example, TLS is used to turn the Hypertext Transport Protocol (HTTP) into its secured version HTTPS and the Noise protocol is used by the widespread WhatsApp messaging service.

TLS itself is standardized and can tunnel payload and higher level protocols in many ways. For this reason, we will only refer to TLS in general in the following paragraphs. In comparison, "Noise" rather is a protocol framework that describes a protocol and requires a specific application implementation to exist. Therefore, we will refer to WireGuard implementing exactly one crypto algorithm and one protocol scheme.

For cryptography algorithms and security protocols, it is generally advisable to stick to proven solutions ([16]). Even small weaknesses turn the whole implementation vulnerable. For this reason, TLS has been widely adopted. However, a matured solution like TLS that has received continuous updates and adoptions to many applications, also grows in complexity and becomes more susceptible to implementation flaws, for example the infamous Heartbleed Bug published in 2014. The implementation of TLS is accompanied by over 20 extensional internet standards (RFC). E.g., the informational RFC 7457 exists alone to list known vulnerabilities and weaknesses to TLS implementations. Common sources of weaknesses in TLS are protocol downgrades to a broken cryptographic algorithm, buffer overruns of message parser, weak implementations of algorithms and dubious interpretation of certificates [17].

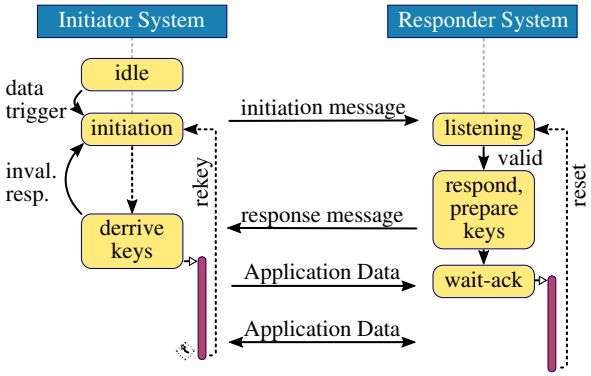

Fig. 5. The WireGuard state-machine has few states. The initiator starts rekeying after 120 s. Otherwise, the responder discards its transport keys after a maximum of 180 s.

Some of those weaknesses of TLS are addressed in the latest version 1.3 [10], which is now a proposed standard, RFC 8446. Major changes listed in the RFC:

- Legacy encryption algorithms were removed.
- Only Authenticated Encryption with Associated Data (AEAD) alg.
- Static RSA and Diffie-Hellman (DH) suites were removed for ephemeral var.
- Key-exchange suites provide Perfect Forward Secrecy.
- The handshake state machine has been significantly restructured to be more consistent and to remove superfluous messages.
- Elliptic curve algorithms move to base specification; signature algorithms Ed25519 and Ed448 were added. Point format negotiation was removed.

As a result, current TLS Compliance Requirements (see section 9 of the RFC [10]) can be considered smaller than previous versions. The subsection 9.1 of the RFC lists one mandatory and two optional to implement cipher suites for AEAD and two Diffie-Hellman key exchanges. Which cipher suite is used for the application data, is negotiated in the handshake process. The TLS handshake messages have optional and mandatory extensions, e.g., the "KeyShare", the certificate, etc. As a result, the handshake messages are of variable length and of varying complexity, which has led to vulnerabilities and implementation mismatches in the past [17].

The WireGuard protocol [2] on the other hand, uses a much more simplified approach. Only one cipher suite is specified and certificates are not part of the protocol. The control-flow in Fig. 5 has an untangled structure. Peer selection and verification is solely based on public keys. A more or less sophisticated Public-Key-Infrastructure (PKI) may be implemented separately, but only if the overall application requires this. Like in TLS 1.3, two message types are sent in the handshake: the initiation and the response. A third message type can be sent by the responder instead of a response msg., with low computational effort, if

the replying responder is unable to serve the costly Diffie-Hellman key calculation. This may be due to exhausting the real-time scheduling budget, when the component is under heavy load (e.g., in an adversary attack), avoiding denial of service complications. The fourth message type transports the application data. All four message types have a fixed length and a fixed structure. Hence, the parser is immune against length and buffer-overrun vulnerabilities.

Recently in 2018, the authors of WireGuard have published their results of the formal security verification of the protocol in [3]. The verification efforts are based on the tool Tamarin, and assert the security properties of the modelled protocol according to key agreement, key secrecy, session uniqueness and identity hiding. Beyond these properties, due to numerous static data structures and avoidance of dynamic memory allocations, the main C implementation of WireGuard claims low risk of unsafe behaviour and was recently accepted as a mainline Linux kernel driver module. The Linux implementation uses formally verified implementations of the X25519 algorithm published in [18] and [4], applying formal methods of F* and Coq.

As an alternative for use in safety-critical systems, we have also approached implementing the WireGuard protocol as a Scade model based on previous work in [13]. While the implementation reached proof-of-feasibility status for the demonstrator within the OBU component, it is still work in progress. We can conclude that the straightforward protocol state-machine and clear algorithmic choices of WireGuard make such an implementation feasible with hard real-time requirements. On the other hand, it must still be analysed whether a special implementation in Scade is necessary or, if the mainline implementation together with the anyway necessary security measures supported by the MILS architecture provide enough assurance. The MILS separation architecture does ensure non-interfering separation in terms of CPU time and memory space, for each component independent of its assurance level.

6 Conclusion

In the previous chapters, we argued our choice of components and the applied system architecture to build a secure demonstrator composed of network-connected devices hosting model-based safety-critical applications.

When used together with TRDP as a middleware, the Scade-generated code does not require much boilerplate code, other than memory initialization and cycle timing. TRDP needs only minor adaptations in terms of a modified Scade-type dataset marshalling function and operating system layer modification. PikeOS is not directly supported in TRDP Light, but via the POSIX adaptation layer. After modification, sourcing the TRDP-XML configuration from the PikeOS rom-image property file system (pfs) unifies overall configuration and removes the need for a full-blown file system. The WireGuard component was also adapted to source the configuration for peer-public-keys and endpoint addresses from the pfs. Building these components on the base of the MILS separation architecture ensures security assurance for different software integrity-

Fig. 6. The physical demonstrator component. Shown are the modified model trains with the Bluetooth-LE (red PCB) back-channel to notify balise detection for odometry. The IRDA balises are seen in the foreground tracks between the white ribbon-cables connected to the IR-PCB.

and assurance levels throughout the whole life cycle accommodating security updates for individual components.

Ongoing work in the certMILS project will guide the discussed MILS system architecture towards an accomplished certification methodology for secure safety-critical products. This will be demonstrated on real-world pilots, such as a power-grid control unit, a platform approach for SIL-4 railway applications and a demonstrator for the Prague subway system. Our educational railway demonstrator (Fig. 6) will also benefit from those pilot projects, applying developed testing techniques, improving integration tooling and fixing bugs that still need special procedures. In the short term, we also like to integrate the TRDP-SDT extension with Scade models, as well as evaluate the performance of our formalized WireGuard implementation and find an answer whether it is beneficial for secure data transport in safety-critical applications. A vital part of this ongoing work is to discuss the current real-time performance of the demonstrator on competitive, i.e., related industrial hardware.

Acknowledgments. This work is part of the certMILS project, funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 731456.

References

1. CENELEC: Electronic railway equipment – train communication network (TCN) – part 2-3: TCN communication profile (IEC 61375-2-3:2016). Tech. rep., IEC (2017)
2. Donenfeld, J.A.: Wireguard: Next generation kernel network tunnel. In: NDSS Symposium (2017), https://www.wireguard.com/papers/wireguard.pdf

3. Donenfeld, J.A., Milner, K.: Formal verification of the wireguard protocol. Tech. rep., Oxford University (2018), https://www.wireguard.com/papers/wireguard-formal-verification.pdf
4. Erbsen, A., Philipoom, J., Gross, J., Sloan, R., Chlipala, A.: Systematic generation of fast elliptic curve cryptography implementations. Tech. rep., MIT, Cambridge, MA, USA (2017)
5. Ferguson, N., Schneier, B.: A cryptographic evaluation of ipsec. Counterpane Internet Security, Inc (2000), http://www.cs.fsu.edu/~yasinsac/Papers/ipsec.pdf
6. Gorski, P., Özer, M., Schulz, T., Golatowski, F.: A modular train control system through the use of certified cots hw/sw and qualified tools. Elektronik **18** (9 2016)
7. Hametner, R., Resch, S.: A Platform Approach for Fusing Safety and Security on a Solid Foundation. In: 4th International Workshop on MILS. Zenodo (6 2018). https://doi.org/10.5281/zenodo.1306081
8. Hirschler, B., Jakovljevic, M.: Secure deterministic l2/l3 ethernet networking for integrated architectures. resreport, SAE Technical Paper (2017)
9. IEC TC65 WG10: IEC TS 62443-2-4 Industrial communication networks - Network and system security - Part 2-4: Requirements for IACS solution suppliers (2015)
10. Rescorla, E.: The transport layer security (TLS) protocol version 1.3. Tech. rep., IETF (2018), https://datatracker.ietf.org/doc/rfc8446/
11. Rushby, J.: A trusted computing base for embedded systems. In: Proceedings of the 7th D/NBS Computer Security Conference (1984)
12. Schulz, T., Griest, C., Golatowski, F., Timmermann, D.: Strategy for security certification of high assurance industrial automation and control systems. In: IEEE 13th International Symposium on Industrial Embedded Systems (SIES) (6 2018). https://doi.org/10.1109/SIES.2018.8442081
13. Schulz, T., Golatowski, F., Timmermann, D.: Evaluation of a formalized encryption library for safety-critical embedded systems. In: IEEE International Conference on Industrial Technology (ICIT) (2017). https://doi.org/10.1109/ICIT.2017.7915525
14. Tverdyshev, S.: Security by design: Introduction to mils. MILS Workshop Embedded World Conference (2017). https://doi.org/10.5281/zenodo.571164
15. UNISIG: SUBSET-026 – System Requirements Specif. SRS 3.3.0, ERA (2012)
16. Victors, J.: TLS 1.3 and the future of cryptographic protocols. Tech. rep., Synopsys (2016), https://www.synopsys.com/blogs/software-security/tls-1-3/
17. Walz, A., Sikora, A.: Exploiting dissent: Towards fuzzing-based differential black box testing of TLS implementations. IEEE Transactions on Dependable and Secure Computing pp. 1–1 (2017). https://doi.org/10.1109/TDSC.2017.2763947
18. Zinzindohoué, J.K., Bhargavan, K., Protzenko, J., Beurdouche, B.: Hacl*: A verified modern cryptographic library. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. CCS '17 (2017). https://doi.org/10.1145/3133956.3134043