Evaluation of a Formalized Encryption Library for Safety-Critical Embedded Systems

Thorsten Schulz, Frank Golatowski, Dirk Timmermann
Institute of Applied Microelectronics and CE, University of Rostock,
{Thorsten.Schulz,Frank.Golatowski,Dirk.Timmermann}@uni-rostock.de

***Abstract*—Complex safety-critical devices require dependable communication. Dependability includes confidentiality and integrity as much as safety. Encrypting gateways with demilitarized zones, Multiple Independent Levels of Security architectures and the infamous Air Gap are diverse integration patterns for safety-critical infrastructure. Though resource restricted embedded safety devices still lack simple, certifiable, and efficient cryptography implementations. Following the recommended formal methods approach for safety-critical devices, we have implemented proven cryptography algorithms in the qualified model based language Scade as the *Safety Leveraged Implementation of Data Encryption* (`SLIDE`) library. Optimization for the synchronous dataflow language is discussed in the paper. The implementation for public-key based encryption and authentication is evaluated for real-world performance. The feasibility is shown by execution time benchmarks on an industrial safety microcontroller platform running a train control safety application.**

I. Introduction

Security is an essential building part of dependable computing. While past critical systems focused on the other important aspects of *Reliability, Availability, Maintainability and Safety* (commonly referred as *RAMS*). Integrity and confidentiality were usually covered with physical barriers (see "The Air Gap" [1]) to prevent malicious attacks to critical infrastructure. But the connected infrastructure is not only limited to process data messaging. With the incline of technical complexity contained in current industrial automation systems, reliability and availability are challenged by difficult failure scenarios. Maintenance more often needs assistance by telemetry data collection systems. Therefore systems become increasingly interconnected and offer interfaces for control, update and diagnostics beyond system borders. Input from these interfaces must never change the system into an unsafe state. This requires the device's safe certified software to assure that any message from an input communication channel must come from an authentic peer with provable integrity. As such, we propose an implementation of the cryptography algorithms to be an integral part of the qualified safe software.

In contrast, current security is often integrated into systems by including common libraries such as *openSSL* or implementations of *IPsec*. Researchers in [2] also found outdated implementations in proprietary products. This is due to the fact that a lot of applications designers are no cryptography experts and rely on these libraries to provide common functionality. To cater for high levels of interoperability security libraries like openSSL provide many algorithms and features also to address legacy devices. Infrequently updated versions contain algorithms with weak or broken security. Due to their massive complexity these libraries are unable to be thoroughly verified by formal methods. For example memory leaks, exploited by the recent *Heartbleed Bug*, must be formally inhibited in the development process for safety qualified systems.

Other approaches use encrypting gateways for outbound communication, as proposed in the early 2000s (see [3]) and before. This tends to provide acceptable protection together with the *Demilitarized Zone* (DMZ) pattern. But other attacks like *Stuxnet* have shown that an attacker may manipulate SCADA systems beyond these gateways or by compromised service computers. While these scenarios require much more attention than "just" integrated communication encryption, it shows that separating security and safety measures leaves open gaps for attacks.

This paper presents the results from implementation of a proven cryptography algorithm for the domain of safety-critical devices. By integrating the cryptography algorithms as part of the overall control model software, mandatory exhaustive software tests can be executed to the full depth. As a result, certification of safety-critical software using secure communication interface decreases in complexity. In the next sections, the application background of safety-critical control systems is illustrated and important basic requirements for an implementation are enumerated. The chapter III contains details about the implementation. Since using *formal methods* requires special attention in coding, an approach to optimizing the implementation is discussed in an extra section. Chapter V contains results collected from execution of the implementation on a physical system. The results are compared and discussed with respect to their application related real-time properties. The discussion finalizes with an outlook of further implementation work. The presentation of a holistic security concept including a wire protocol, certificates, random number generation, key storage and exchange are greatly beyond the scope of this paper and were not considered.

II. Application Background

A train control system (TCS) module will serve as an example for a complex safety-critical control application. Malfunction of the TCS must not cause uncontrolled movement beyond safe limits which could endanger passengers of a train by, e.g., starting to move while the doors are open. Availability requirements on the other hand set very strict maintenance or down times. Preventative maintenance by evaluation of abnormal real-time sensor data could be a suitable solution. The data is sent via public carriers to the central maintenance system and corrective parameters can be sent back to the control device to circumvent a current issue. This is impossible

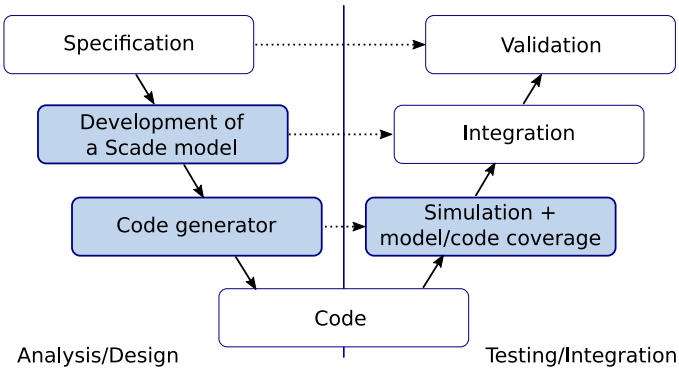

Figure 1. Implementation life-cycle phases in SCADE related to the recommended life-cycle in [4]. SCADE Suite provides phases of the shaded boxes. Redrawn work from [5] "Figure 8.7 SCADE as a design environment".

without strong reliable security that is leveraged to the safety standards.

Systems that execute safety-critical functions have to comply to the norm *IEC 61508* by law of liability. *IEC 61508* is the generic standard, which is then refined for the specific domain, such as aerospace, industrial processes, machine safety, automotive, and railway. For example, the (European) railway domain requires the norm series *CENELEC 5012x* to be applied. Here *CENELEC 50126* defines the overall systems engineering process and how failures are classified and handled.

Within these norms the *Safety Integrity Level* (SIL) is declared in the range of 1 to 4, with four being the level of most confidence. It is derived from the statistical rate of systematic failures and random failures. Confidence in safety integrity can also be reached by effective combination of special architectures, processes, tools and techniques. To give an orientation, German national law typically requires high-speed trains to provide a SIL 4 control systems, whereas freight, commuter and regional trains with a maximum speed of 160 km/h or less suffice with SIL 2 control systems.

Specifically, the norm *CENELEC 50128* (*"software for train control systems"*) defines goals, conformity rules, and the software safety integrity level (SSIL). It also proposes the software life-cycle (see Fig. 1) including its documentation and the development process: tests, verification, validation, assessment, quality assurance, change management and tools and programming languages. Depending on the role and the result of action of a train's device's software, not all on-board devices in a high-speed train must conform to the strongest safety standards. For example a passenger information screen is classified as a non-critical device unless it is part of the emergency evacuation plan.

Safety Leveraged Implementation of Data Encryption (SLIDE) aims to provide an implementation with methods required by higher safety devices with a non-zero SIL. There are several ways and methods to comply with the related norms. Known security libraries are generally available in the C language. But the C language is only allowed with restrictions to be used in critical applications (see [5] Chpt. 7). Without going into this detail, the recent prominent *Heartbleed Bug* is a result of unrestrained use of the C-language. As an alternative, the CENELEC 50128 norm *"Highly Recommends (HR)"* the use of formal methods ([4] Table A.17). [5] introduces several tools from this domain. We chose *ANSYS SCADE* as a proven qualified toolchain for CENELEC 50128-SIL 4 (and other *IEC 61508*) applications for the exemplary implementation. Figure 1 shows which parts are covered by SCADE Suite. The commercial tool-suite contains additional applications to also cater for specification and validation phase of the software life-cycle, but this not in the scope of this paper.

SCADE Suite is a commercial modelling tool for its own language *Scade* that has evolved from the language *LUSTRE*. As Boulanger describes in [5]: "It facilitates modeling based on the concepts of functional block diagrams and dataflow diagrams". An excerpt is shown in Figure 3. *Scade* is a synchronous reactive programming language, that is executed in cycles. On each cycle all inputs stay constant and each output has to compute a defined value. Outputs derive their value from dataflow from inputs through operators. Feedback of data can only be achieved with special memorizing operators.

One major prerequisite that plays into the evaluation of the proposed implementation, is the execution time. TCS typically have real-time application cycles of 50..200 ms. For example the *openETCS* model, a research project for an open-source safety supervision system [6], is targeted to run on a 100 ms cycle. The evaluation reference setup for a TCS with encrypted in- and outputs includes the brake-distance supervision module from *openETCS*. With a powerful safety microcontroller hardware, as used in the following tests, up to 27 ms are required to execute the TCS module within the test bench. The "spare" time can be used for encryption-jobs. The preliminary test results for feasibility on this TCS application conclude the measurements before the discussion section.

III. Implementation

An open and standardized encryption library, that is formally verified for use in high safety-critical applications, is not known. Though related modelling approaches are proposed in [7]. The implementation gap is even underlined in [8] with differentiating between non-encrypted safety-critical and non-safety messages in vehicular ad-hoc networks.

To prove the applicability of authenticated encryption in hard real-time dependable systems the approach was to implement a proven library using a qualified tool-chain from the safety domain. Due to the reasoning explained in the previous chapter II and existing experience, *ANSYS SCADE Suite R16* was chosen.

As a publicly available, sophisticated but simple open source cryptography library, the *Network and Cryptography library* (NaCl, pronounced "salt") by *D. J. Bernstein* [9] was chosen. It represents state-of-the-art crypto-algorithms, that are actively used in a variety of applications. There are also a number of implementations in many programming languages available, of which *μNaCl*, for 8 bit-microcontrollers [10] and a native Java implementation have been used in a preceding project for public-key-signature based Bluetooth low-energy beacons, see [11].

SLIDE is a Scade implementation of NaCl with the algorithmic scope of *TweetNaCl* [12]. It contains all high-level

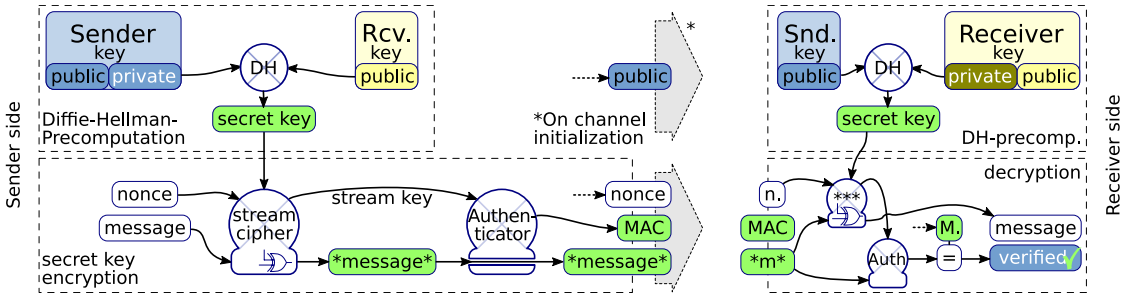

Figure 2. Overview of public key based message exchange. Transmission of the senders public key is only done on channel initiation. Each following message below contains the encrypted data, the MAC and the nonce (a "number-used-once", unique per message and per key).

crypto-primitives (functions) of NaCl necessary to establish secure data transfer based on public-key-cryptography schemes. Public-key cryptography (PK) divides into Authenticated Encryption (see Figure 2) and Signatures[1]. PK authenticated encryption builds upon secret key generation from public-private key-pairs with the Diffie-Hellman-Function (DH) "X25519" on the "Curve25519" (a Montgomery curve) and secret key authenticated encryption primitives. The input to the DH-function are the private key of the sender and the public key of the receiver and on the receiver side vice versa (top half Figure 2). The derived *secret key* can be used for multiple messages, as long as each pass is encrypted with a unique number, a "number-used-once" or "*nonce*". The nonce can be consecutive, randomness is not required.

Secret key authenticated encryption, as drawn in Figure 2, consists of a stream-cipher and an Authenticator. The stream-cipher (here "XSalsa20") generates a stream key that is XOR'ed with the data to encrypt the message. The Message Authentication Code (MAC) produced by the Authenticator (here "Poly1305" algorithm) is a cryptographically secure checksum calculated from the encoded message data and initial part of the stream key, to assure the message was not modified along the transport. Decryption on the receiver side results in almost the same algorithm, since XOR'ing with the same secret key will produce the original message. The MAC computed from the received encrypted message must equal the received MAC to verify the authentication.

The DH-function is a mathematically complex trap-door-function and uses most of the execution time. It is only required on channel initiation or key-renewal. The secret key algorithms run on every message iteration and are highly dependent on the message length. The two aspects of the library will be analyzed in the results section. From the analysis the most critical bottle necks, that were introduced through a naive Scade implementation, were selected for optimization and the findings are detailed in the following section.

IV. Optimization

The goal of the initial `SLIDE` implementation is a proof of concept without finalized optimization. Mislead optimization can lead to unnecessary obfuscation which makes verification harder. On the other hand, there are challenges and common pitfalls that can render the result unusually inefficient. Most performance penalties come from using patterns that are foreign to the implementation language.

The synchronous dataflow paradigm of Scade does not support free formed instruction loops known from the C-language. In dataflow an operator is defined to be applied to all elements of an array in parallel (`map`) or in an iteration (`fold`). Most `for`-loops in the encryption library have a constant length by design either due to the algorithm or to avoid side-channel timing attacks. But some encapsulated loops represent awkward constellations that are hard to clarify into linear dataflow structures. The goal was to keep the `SLIDE` implementation as close to the *TweetNaCl* library as reasonable to also be able to verify intermediate results. This section contains an example of resolving the field multiplication `M`, as part of the scalar multiplication of DH, into a high-performance *Scade* implementation.

The `M`-operator uses most processing time of the *Curve25519* algorithms. The C-function is shown in Listing 1 as follows:

```
void M(gf o, gf a, gf b) {
  i64 i, j, t[31];
  for(i=0; i<31; i++) t[i] = 0;
  for(i=0; i<16; i++)
    for(j=0; j<16; j++)
      t[i+j] += a[i] * b[j];
  for(i=0; i<15; i++)
    t[i] += 38 * t[i+16];
  for(i=0; i<16; i++)
    o[i] = t[i];
  car25519(o);
  car25519(o);
}
```

Listing 1. Excerpt from tweetnacl14.c by Daniel J. Bernstein [12]

A naive, direct implementation of the imperative algorithm to a dataflow model would lead to many, apparently non-sequential array accesses (see `t[i+j] += a[i] * b[j]`). Each of those will be subsequently guarded by range checking code. Additionally, writing to an array non-sequentially, also generated a full array copy per assignment. As a result the complete cryptographic primitive took about 4 seconds on the TI Hercules microcontroller to generate a *Curve25519* key-pair. That is twelve times the original C-*TweetNaCl*-implementation.

The revised operator implementation (see Figure 3) linearizes the dataflow to generate the output sequentially. This

[1]The Ed25519-Signatures based on scalar multiplication and SHA2-512 hashing are implemented, but not evaluated in the concluding results section.

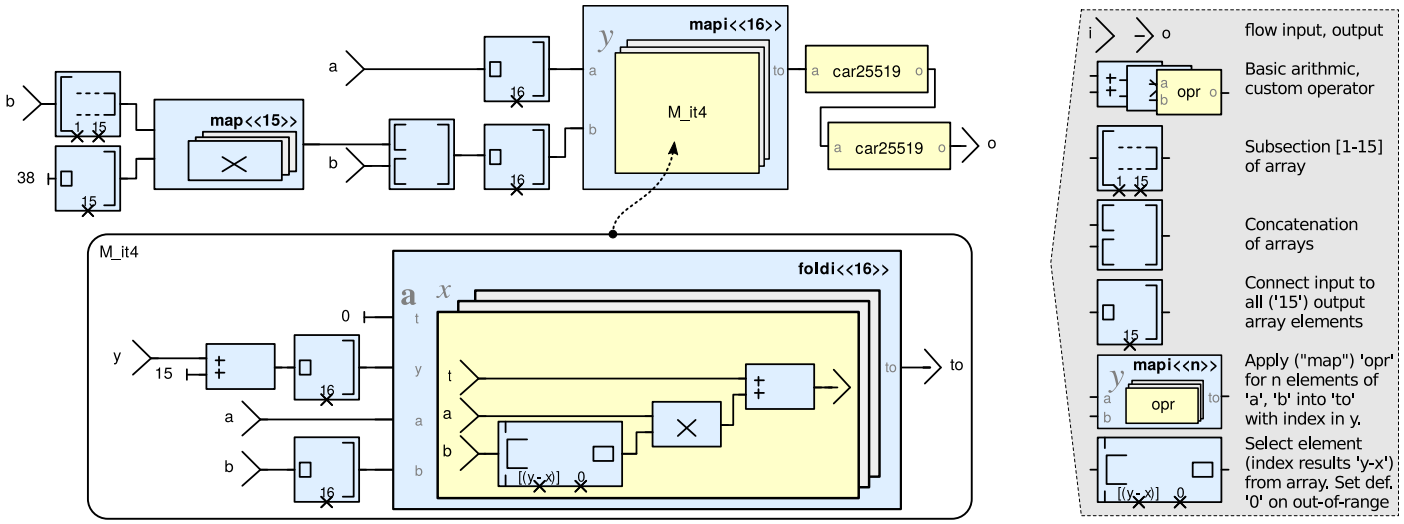

Figure 3. Optimized *Scade* implementation of operator `M` leading to satisfactory performance. This is an assembled screen-shot from the ANSYS SCADE modeller of the `M`-operator and its iteration operators. `foldi` extends `mapi` that it accumulates the first input-output flow (here 't'->'to') along all array elements.

is achieved by transposing the original algorithm. Then the inner iteration only accumulates into a single `t`-field with the outer loop mapping this operation sequentially over the length of the array `t`. This corrected design led to a speed up by factor 15 down to an execution time of 270 ms as shown in the results section.

Another major impact to execution times are compiler and *Scade*-code generator options. The compiler optimization options heavily depend on specific application domain's safety requirements, compiler maturity and the compiler verification results and are out of scope of this paper.

ANSYS SCADE's C-code generator (KCG) has the option to *expand* model operators. Initially each non-expanded operator will generate its own C-function. Expanding a *Scade* operator is comparable to in-lining a C-function. As this is not part of the model domain but of code generation, there is no final penalty on creating sub-operators to keep the model simple for reuse, traceability and certification. After the model design phase, sophisticated tools such as stack- and timing-analyzers by *Absint* (qualified) or *callgrind* from the *valgrind* tools (open-source) can highlight bottle necks, where operator expansion is worth evaluating. Too much expansion leads to untraceable C-code, increases code size and stack usage and may even become uncompilable due to complexity. In fact, the *Texas Instruments ARM code generation toolchain* was unable to parse a fully expanded `SLIDE` library. As a result, there is only one code generation version of the `SLIDE` library in the results section that employs only selected expansion after analysis.

V. Evaluation

Commercial-Off-The-Shelf (COTS) Rugged PC systems with good processing power have gained wide popularity in the critical systems domain. But also embedded devices with limited resources System-on-Chip (SoC) microcontrollers have gained sophisticated computational capabilities. To verify the feasibility of data encryption using a certifiable formal methods implementation, the tests were executed on the safety microcontroller platform *Hercules* by *Texas Instruments* (TI). Typical applications are distributed automotive, railway and industrial control with network-based data bus.

A. Measurements

The *TI Hercules* platform implements an *ARM Cortex R* core with max. 330 MHz, ECC protected 512 kByte SoC-RAM and 4 MB Flash ECC-ROM. The high safety certification of up to *IEC 61508* SIL 3 is based on the *LockStep-CPU* concept: A hardware comparator checks the outputs of two cloned CPU cores on every clock cycle. The specific test hardware is the *TI RM57Lx LaunchPad Development Kit*. The core is configured to run at 300 MHz. This board has only SoC-Memory, so cache will not affect Worst Case Execution Time.

To keep the effort for the proof-of-concept reasonable and the code base clear, the minimalist *TweetNaCl*-implementation (see [12]) was used as reference. Since it is a pure C implementation, it is platform independent and can be used as a reference for the measurements. For comparison with a production-ready library the high-performance implementation *libsodium* (version 1.0.10 [13]) was also chosen. All implementations use similar interfaces with only minor specific adaptations to types and external data structures to equally fit the test bench. The specific encryption parameters were passed individually and without any sort of protocol. Actual network transmission was not used for the tests in the results section.

The tests were compiled using a current *Texas Instruments ARM compiler toolchain* (TI-CGT 15.12.3.LTS). The TI-CGT is advertised to be certifiable but it was not investigated to what extend and under which conditions. TI-CGT brings an extensive set of tunable options and generation switches. Required settings for certification are very application specific and were not taken into account for the performance tests. A single configuration setup was chosen favoring speed.

The encryption model was linked and flashed to the *TI Hercules* evaluation board directly without operating system. The

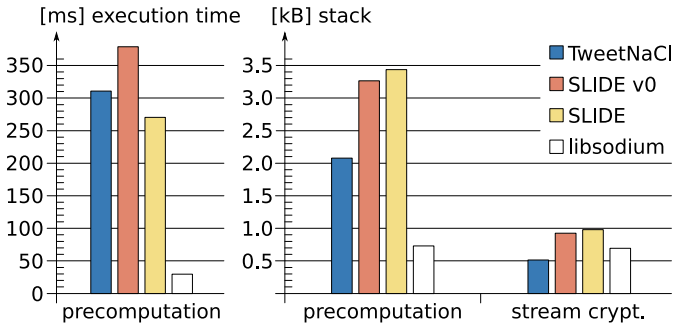

Figure 4. Left: measured execution time of the key precomputation (left) of the different implementations. Right: the stack memory requirement for the precomputation algorithm and the authenticated encryption algorithm (not including the external message buffers).

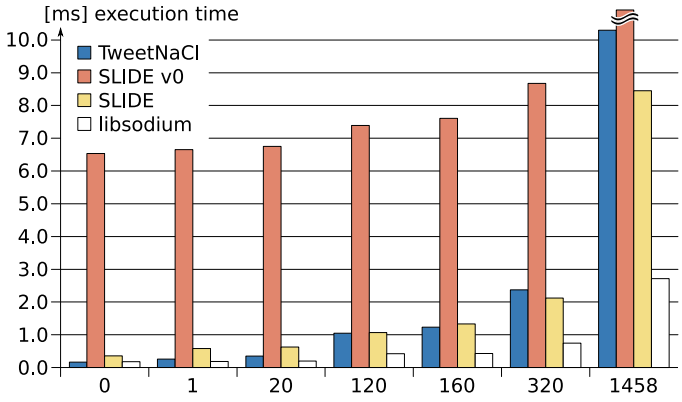

Figure 5. The cyclic authenticated encryption run time dependent on the payload length. The quantization due to block-based (64 byte) handling is noticeable. The *SLIDE* implementation improved over *SLIDE v0* especially due to optimization of the Message-Authentication operator.

results were first recorded, and printed to the embedded serial debug channel after actual model processing. The run time was measured using samples from the real-time counter unit at a sub-microsecond resolution. Since no interrupts were enabled, no caches involved, the measurements were reproducible constant through consecutive executions within the resolution of the real-time counter.

The tests compare the stack usage and the run time of different primitives that were run consecutively. First shown in Figure 4 is the precomputation, which is used to derive the secret encryption key from the public-private key-pair (see top left in conceptual overview Figure 2). The run time of the authenticated encryption depends on the message length. The tests were executed for message payload lengths of [0, 1, 20, 120, 160, 320, 1458] bytes (see Figure 5). The keys and nonces were always derived from the same fixed data source to provide verifiable results. Stack-usage estimation was done via "memory coloring". More sophisticated stack-analysers are available but this was no current focus.

All graphs line-up the four implementations that have been introduced earlier in the paper. *TweetNaCl* is the original one-file-of-C-code version extended with wrapper functions for the test-case. "*SLIDE v0*" is an intermediate version of SLIDE. The *libsodium* was included mostly unmodified, up to minor unneeded parts that were not compilable on the bare-metal platform. Figures 4 and 5 show the improvement from `SLIDE v0` to `SLIDE` in terms of execution time. Not shown for readability is the very first attempt of SLIDE with a precomputation time of 4 seconds. The model-based optimization as explained in section IV could well identify and diminish performance problems of early versions.

To round up the initial exemplary application background (see II) of a TCS module, the execution times of the encryption algorithms are compared to the execution of a computationally complex control application. The *European Train Control System* (ETCS) contains the on-board component *Speed and Distance Monitoring* to supervise the safe braking distance of a moving train, under consideration of track gradients, multiple sections of different speeds with a complex characterization of the train's braking system. The module is known to consume a larger part of the execution time of the ETCS-kernel. Even though in the real-world this is an internal module, for a demonstration system (see [14]) the component was interfaced directly with a display system (HMI), a physical model and a data source. The required inputs are packets of size 32, 4 and 912 Bytes. The outputs are 12, 1400 and 308 Bytes long. The results are shown in Figure 6. The execution time was 27 ms for the Speed and Distance Monitoring module, 13 ms for adaptation to the simulation environment and 19 ms for decryption of the inputs and encryption of the outputs. The execution time of 60 ms fits well within a 100 ms cycle period.

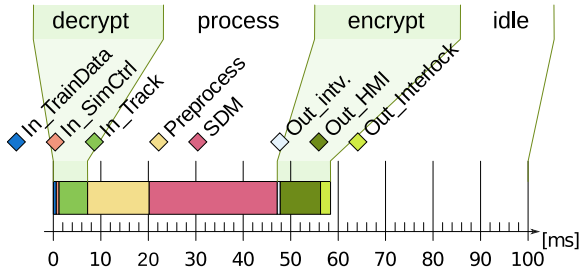

Figure 6. Experimental execution time within the period of the openETCS Speed and Distance Monitoring with encrypted in- and outputs in a test setup.

B. Discussion

The advantages of software based solutions are architecture independent implementations and easier maintainability throughout the system life-cycle. The testing can start earlier and there is less risk of interfacing a proprietary hardware solution. A disadvantage over hardware solutions is the demand for processing power.

It must also be noted that the precomputation of the secret symmetric encryption key (the *Curve25519*-scalar multiplication) takes notably longer, here 44 times more than the per-cycle authenticated data en-/decryption. Though this functionality is only required on channel establishment or on key-renewal. On the investigated *TI Hercules* board the `SLIDE` execution time of 270 ms is larger than typical train control application periods of 100 ms. A functional extension was added successfully to statically break the scalar multiplication across multiple execution cycles, but the results are not yet reflected in the preceding section.

Very interesting research in [15] investigates the safety properties, i.e. hazard rate of undetected message corruption, of current (symmetric) cryptography algorithms AES and DES. The algorithm *XSalsa20* used in *NaCl* (and as such in SLIDE) has a comparable strength to AES. Since our approach uses the *Poly1305*-MAC in each message additionally to underlying OSI layer codes of Ethernet like CRC-32 or even CRC of the UDP packet, it should surpass reliability rates. The SLIDE implementation does not yet provide any such safety related evaluation neither does it provide any safety protocol related fields. Though the conclusion is, that a cryptographically secure message transport may also be a safe transport.

VI. Conclusion and Outlook

The results demonstrate that software based hard-realtime implementation of data encryption within the higher safety domain is a viable option for industrial applications. Under consideration of cycle periods of 50..100 ms, this can also be performed by resource restricted safety microcontroller platforms. Excerpts from the implementation process have shown that not respecting divergent coding paradigms of a safety programming language results in inefficient code.

What was disregarded for explicit proof, was whether the *SCADE-KCG* generated code provides the same resilience to side channel attacks as the original implementation. The qualification of KCG guarantees, that the generated code behaves identical to the model. So, obeying to the same algorithms and precautions (e.g. no branches on secret data) is expected to propagate to the code.

The property of the *NaCl* algorithms to be completely deterministic is essential for verification and validation. For further implementation, quality randomness for key generation and secure storage of secret keys need to be added. By definition this requires external hardware support, e.g. trusted platform modules, cryptography chips (e.g. *Atmel CryptoCompanion*) or potentially microcontroller internal peripherals. The examined *TI Hercules* did not provide any *Hardware Security Module*, but a computationally comparable *Infineon AURIX safety platform* is equipped with such.

Concerning encrypted *communication*, the examined test cases are fairly limited. The next step is to implement a communication protocol to pass multiple messages. The "minimaLT" concept in [16] proposes a simple and efficient approach, but it still lacks acceptance. The DTLS protocol on the other hand has been recently advanced with the RFC 7905, adding the authenticator *Poly1305* (as used in SLIDE) and the stream cipher *ChaCha20* which is a recent successor of *Salsa20*. Focus will subsequently shift towards implementation of ChaCha20 and the DTLS protocol to make an integrated SLIDE device talk to COTS DTLS implementations.

Another recently added aspect is to use the signature and verification algorithms of the SLIDE library for Secure Boot/Update purposes in Multiple Independent Levels of Security (MILS) high-assurance systems. It can provide additional algorithms if a hardware implementation is missing specific support or is found vulnerable during the life-cycle. As discussed earlier, further research must assure that security properties of the abstract software library hold for refinement on specific hardware.

Acknowledgment

This work has been partially funded by the German Federal Ministry of Education and Research (BMBF) under reference numbers 01IS13019H and 01IS13019B as part of the European ITEA project "Building as a Service" (BaaS).

Extensive thanks to the authors of the *NaCl Networking and Cryptography library* for their excellent work!

References

[1] E. Byres, "The air gap: Scada's enduring security myth," *Commun. ACM*, vol. 56, no. 8, pp. 29–31, Aug. 2013. [Online]. Available: http://doi.acm.org/10.1145/2492007.2492018

[2] A. Guarino, "Information security standards in critical infrastructure protection," in *ISSE 2015*. Springer, 2015, pp. 263–269.

[3] E. Byres and J. Lowe, "The myths and facts behind cyber security risks for industrial control systems," in *Proceedings of the VDE Kongress*, vol. 116, 2004, pp. 213–218.

[4] C. E. de Normalisation Electrotechnique, "Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems, EN 50128," *EUROPEAN STANDARD*.

[5] J.-L. Boulanger, *CENELEC 50128 and IEC 62279 standards*. John Wiley & Sons, 2015.

[6] D. P. Mahlmann, B. Hekele, A. Mohammed, M. Behrens, D. H. Hungar, D. M. Jastram, S. Karg, U. Steinke, J. Welte, D. C. König, D. C. Stahl, and T. Schulz, "openetcs: Design and implementation of open-proof-concepts for the european train control system etcs," 2016. [Online]. Available: http://github.com/openETCS/Dissemination

[7] A. Motii, A. Lanusse, B. Hamid, and J.-M. Bruel, *Model-Based Real-Time Evaluation of Security Patterns: A SCADA System Case Study*. Cham: Springer International Publishing, 2016, pp. 375–389. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-45480-1_30

[8] C. Büttner and S. A. Huss, *An Efficient Anonymous Authenticated Key Agreement Protocol for Vehicular Ad-Hoc Networks Based on Ring Signatures and the Elliptic Curve Integrated Encryption Scheme*. Cham: Springer International Publishing, 2015, pp. 139–159. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-27668-7_9

[9] D. J. Bernstein, T. Lange, and P. Schwabe, "The security impact of a new cryptographic library," in *Progress in Cryptology – LATINCRYPT 2012*, ser. Lecture Notes in Computer Science, vol. 7533, 2012, pp. 159–176, document ID: 5f6fc69cc5a319aecba43760c56fab04, http://cryptojedi.org/papers/#coolnacl.

[10] M. Hutter and P. Schwabe, "NaCl on 8-bit AVR microcontrollers," in *Progress in Cryptology – AFRICACRYPT 2013*, ser. LNCS, vol. 7918, pp. 156–172, document ID: cd4aad485407c33ece17e509622eb554, http://cryptojedi.org/papers/#avrnacl.

[11] T. Schulz, F. Golatowski, and D. Timmermann, "Secure privacy preserving information beacons for public transportation systems," in *IEEE International Conference on Pervasive Computing and Communication*. IEEE, 2016, pp. 190–195. [Online]. Available: http://dx.doi.org/10.1109/PERCOMW.2016.7457096

[12] D. J. Bernstein, B. van Gastel, W. Janssen, T. Lange, P. Schwabe, and S. Smetsers, "TweetNaCl: A crypto library in 100 tweets," in *Progress in Cryptology – LATINCRYPT 2014*, ser. LNCS, vol. 8895, 2015, pp. 64–83, document ID: c74b5bbf605ba02ad8d9e49f04aca9a2.

[13] F. Denis, D. J. Bernstein, P. Schwabe, and Community, "The sodium crypto library (libsodium)," May 2016, https://libsodium.org.

[14] P. Gorski, M. Özer, T. Schulz, and F. Golatowski, "A modular train control system through the use of certified cots hw/sw and qualified tools," pp. 42–49, September 2016.

[15] M. Franekova, P. Luley, and T. Ondrasina, "Modelling of failures effect of open transmission system for safety critical applications with the intention of safety," *Elektronika ir Elektrotechnika*, vol. 20, no. 1, pp. 19–24, 2014.

[16] W. M. Petullo, X. Zhang, J. A. Solworth, D. J. Bernstein, and T. Lange, "Minimalt: Minimal-latency networking through better security," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13, 2013, pp. 425–438. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516737