In Search for a Simple Secure Protocol for Safety-Critical High-Assurance Applications

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*—Security and cryptography protocols are seen by many as black-magic, largely due to their complex mathematical algorithms and entangled state-machines. This complexity has also led to numerous vulnerabilities in past years. Recent developments have simplified conformance requirements, and also introduced formal proofs to mainstream security protocols. In this work-in-progress publication we discuss, how this evolution has greatly improved the situation for critical systems, and how the architecture of MILS systems can raise the confidence for high-assurance systems.**

***Keywords*—security, formal modeling, safety critical systems, CPS**

I. Security in Critical Systems

Critical systems are required to be reliable, available, maintainable and safe according to accepted and governing standards, e.g., EN 50126 in the railway domain. These attributes 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 (product). Physical access barriers, e.g., locked doors, 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 electronic and networked 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. As a consequence, testing requires more effort for a much smaller number of operative products. Modifying a critical system's software requires re-certification – even if it is "just" a security update.

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). For example, the system design could split the application into a safe control component and an independent transmission component with security functions, such as remote authorization, authentication and encryption. The safety function within the control application would be independent of corruptions within the transmission component, if it can continue operation in degraded mode without transmission data. The data flow between the components is guarded by the MILS separation kernel, allowing only predefined data flows between the two domains. Depending on attack vectors, system and application design, the security relevant transmission component could then also be of lower confidence level and classified with a low Software Safety Integrity Level (SSIL), being less susceptible to re-certification requirements.

A MILS system is composed of components. For the system to perform a critical safety function, it needs evaluation and certification to standards required by governmental authorities. As mentioned before, evaluation for security of a composed system of apriori certified components, requires special methodologies. Furgel et.al. [1] present the methodology for "Non-Interfering Composed Evaluation" within Common Criteria. The key requirement for non-interference is that the execution of one component does not undermine another component's security policy. For the general case, this demands that all internal states of a component are well defined and well known at any time, as well as all implicit and explicit interfaces between components are clearly defined and accurately described. For a component to demonstrate the adequate evidence for evaluation, this either requires formal methods / proofs or exhaustive testing, including robustness testing.

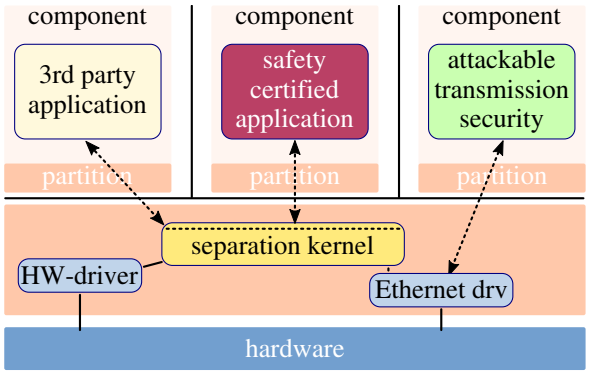

Fig. 1. Architecture of a simple MILS system with a separation kernel, a safety application and additional drivers. The security protocol implementation (green) must be instantiated in a particular environment depending on the level of assured confidence in the correct behaviour and non-interference.

This publication summarizes the findings selecting state of the art methods and algorithms leading to a certifiable cryptographic component for a high-assurance MILS system. The targeted encryption protocol should, on the one hand, help

secure arbitrary data transmission between systems over an open internet protocol (IP) network link, especially in terms of authenticity of the remote system / application, as well as integrity of the message sent. On the other hand, it may also be used as a download channel in the context of Secure Update mechanisms of selected system components. Current protocols and related algorithms will be discussed in the following section.

II. Evaluation of Data Encryption Protocols

Network traffic encryption for security can be applied on different layers of the common ISO-OSI (Open Systems Interconnection model). The tunneled traffic of security protocols typically range from data link layer (2) up to the application layer (7).[1] 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 network layer encryption is the most versatile choice – most prominently used as Virtual Private Networks (L3-VPN). The following 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 the Noise protocol.[2] However, algorithms used within L3-VPN can always, and, in terms of TLS, are also used to tunnel specific higher level application protocols. For example, TLS is used to turn the Hypertext Transport Protocol (HTTP) into its secured version HTTPS. Consequently, a Secure Update functionality based on standardized file transfer protocols would always require a crypto-library that conforms to all mandatory requirements of the given standard.

In cryptography algorithms and security protocols, it is generally advisable to stick to proven solutions ([2]). As mentioned at the beginning, even a small weakness can 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. 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. A common source 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 [3].

Some of those weaknesses of TLS were addressed in the latest version 1.3 [4], which is in the late draft process. Major changes listed in the draft:

- Legacy encryption algorithms were removed.
- Only Authenticated Encryption with Associated Data (AEAD) algorithms are supported.
- Static RSA and Diffie-Hellman (DH) suites were removed in favor of ephemeral variants.

[1] In contrast, the tunnel protocol itself typically runs on top of TCP or UDP in the application layer.

[2] Like TLS, IPsec is of greater complexity (see evaluation by Ferguson and Schneier) and thus not further considered in this comparison.

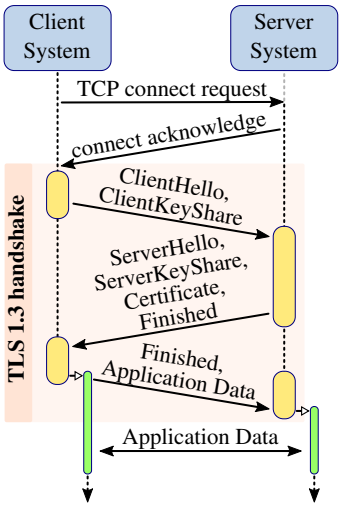

Fig. 2. The basic TLS 1.3 handshake message passing process. There are further more variants related to PSK, 0-RTT and client-authentication.

- Key-exchange suites provide Perfect Forward Secrecy.
- The HMAC-based Extract-and-Expand Key Derivation Function (HKDF) is used for key-derivation.
- The handshake state machine has been significantly restructured to be more consistent and to remove superfluous messages.
- Elliptic curve algorithms are in the base specification, and signature algorithms Ed25519 and Ed448 were added. Point format negotiation was removed in favor of a single point format for each curve.
- Session resumption with and without server-side state.
- Reduction to a single Pre-Shared-Key (PSK) method.

As a result, current TLS Compliance Requirements (see section 9 of the draft [4]) can be considered smaller than previous versions. The subsection 9.1 of the draft lists the mandatory-to-implement cipher suites:

- AEAD:
  - mdt: AES128-GCM-SHA256,
  - opt: AES256-GCM-SHA384 (RFC 5116),
  - opt: ChaCha20-Poly1305 (RFC 7539).
- Diffie-Hellman key exchange:
  - mdt: Elliptic curve secp256r1 (NIST P-256),
  - opt: Elliptic curve X25519 (RFC 7748).
- Digital signatures:
  - RSA PKCS1 SHA256,
  - RSA PSS RSAe SHA256,
  - ECDSA secp256r1 SHA256.

Which cipher suite is used for the application data is negotiated in the handshake process. The TLS handshake messages (Fig. 2) have optional and mandatory extensions, e.g., the "KeyShare", the certificate. 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 [3].

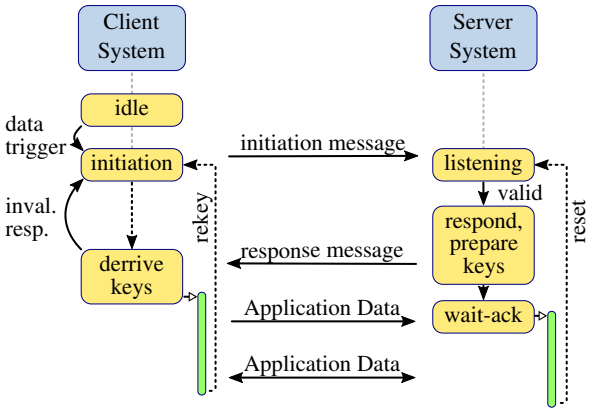

Fig. 3. The WireGuard state-machine has few states. The initiator starts rekeying after 120 seconds. Otherwise, the responder discards its transport keys after a maximum of 160 seconds.

The WireGuard protocol [5] on the other hand uses a much more simplified approach. Only one cipher suite is used and certificates are not part of the protocol. The control-flow in Fig. 3 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. The core of WireGuard is based on the secure protocol framework Noise [6], which has proven use in secure messaging networks such as WhatsApp and Signal. Like in TLS, two messages are sent in the handshake: the initiation and the response. A third message type can be sent by the peer instead of a response with low computational effort, if the replying peer is under heavy load (e.g., in an adversary attack [7]) and unable to serve the costly key calculation. The fourth message type transports the application data. All four message types have a fixed length and a fixed structure, so a message parser is immune against length and buffer-overrun vulnerabilities.

In fact, the cipher suite and the key exchange algorithm are also instantiated in TLS 1.3: the AEAD ChaCha20-Poly1305 (RFC 7539) and the elliptic curve X25519 (RFC 7748). Further implementation requirements are the hash function Blake2s (RFC 7693) and, like in TLS, the Keyed-Hashing for Message Authentication (HMAC) and Extract-and-Expand Key Derivation Function (HKDF) constructs according to RFC 2104 and RFC 5869 respectively.

Very recently, the authors of WireGuard have published their results of the formal security verification of the protocol in [7]. The verification efforts are based on the tool Tamarin and assert the security properties of the modeled protocol according to key agreement, key secrecy, session uniqueness and identity hiding. Due to numerous static data structures and avoidance of dynamic memory allocations, the main C implementation of WireGuard claims low risk of unsafe behaviour to be considered as a Linux kernel driver module. Moreover, the implementation uses verified implementations of the X25519 algorithm published in [8] and [9], applying formal methods of F* and Coq.

III. Summary

After identifying the properties of current state of the art cryptographic protocols in the previous chapter, we need to weight them against the requirements of safety-critical systems from the introduction.

Clearly, TLS has a much wider adoption in all kinds of systems compared to WireGuard. TLS has multiple available implementations and is the basis for encryption for many everyday protocols. Nevertheless, its complexity has grown and recent updates to the standard suggest to rely on the latest version 1.3 for critical systems for its reduced feature set. WireGuard comes with a minimal feature set and small code-base, which make it easier to verify security proofs of the protocol and the execution resources of the implementation. In an earlier publication [10] we also showed that it is feasible, to model the cipher suites in the formal language Scade for safety critical domains.

What has not yet been evaluated in experiment, whether cryptographic hardware support affects the choice. The discussed cipher suite ChaCha20-Poly1305 was designed for superior performance in software on modern processors [11] and there is no known COTS hardware support. AES encryption on the other hand, is often supported in hardware. Advanced crypto-peripherals, increasingly included in larger System-on-Chip (SoC) designs, also assist in hashing (SHA, HMAC) and can calculate standard elliptic curve DH, such as NIST-secp256r1. However, such access to specialized hardware peripherals needs elevated rights in a MILS system and requires a sophisticated system design to secure the data flows. Furthermore, HW-crypto-accelerators cannot be patched on discovered flaws. Whether SW or HW-crypto provides better and long-term security, is an active ongoing discussion and beyond the scope of this summary.

In the introduction, different composition architectures of a MILS system were mentioned. Future work needs to evaluate, whether a network security protocol implementation is too complex to be trustworthy and needs to be shifted into an unprivileged partition component, or if it can be run in the privileged separation kernel / hardware abstraction (PSP) space. The latter case may be advantageous for HW-acceleration and improved performance. However, formal proofs or rigorous robustness tests must assure the non-interference with other privileged components, e.g., the kernel and other drivers.

Ongoing work in the certMILS project will further investigate and implement robustness testing techniques, as well as formal implementations of WireGuard. These experiments will evaluate the feasibility of security for safety on multiple hardware platforms, in different safety-critical domains.

Acknowledgment

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] I. Furgel, V. Saftig, T. Wagner, K. Müller, R. Schwarz, and A. S. F. Blomberg, *Non-interfering composed evaluation*, 2016. DOI: 10.5281/zenodo.47979. [Online]. Available: http://dx.doi.org/10.5281/zenodo.47979.

[2] J. Victors, "TLS 1.3 and the future of cryptographic protocols," Synopsys, Tech. Rep., Apr. 14, 2016, https://www.synopsys.com/blogs/software-security/tls-1-3/. [Online]. Available: https://www.synopsys.com/blogs/software-security/tls-1-3/.

[3] A. Walz and A. Sikora, "Exploiting dissent: Towards fuzzing-based differential black box testing of TLS implementations," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2017, ISSN: 1545-5971. DOI: 10.1109/TDSC.2017.2763947.

[4] E. Rescorla, "The transport layer security (TLS) protocol version 1.3, Draft-ietf-tls-tls13-28," IETF, Tech. Rep., Mar. 20, 2018. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-tls-tls13/.

[5] J. A. Donenfeld, "Wireguard: Next generation kernel network tunnel," in *NDSS Symposium*, Feb. 27, 2017. DOI: 4846ada1492f5d92198df154f48c3d54205657bc. [Online]. Available: https://www.wireguard.com/papers/wireguard.pdf.

[6] T. Perrin, "The noise protocol framework," Tech. Rep., Oct. 4, 2017. [Online]. Available: https://noiseprotocol.org/noise.html.

[7] J. A. Donenfeld and K. Milner, "Formal verification of the wireguard protocol," Oxford University, Tech. Rep., Jan. 21, 2018. DOI: d376f649d7f4b68f616e05e5f64d660e9b23d7af. [Online]. Available: https://www.wireguard.com/papers/wireguard-formal-verification.pdf.

[8] J.-K. Zinzindohoué, K. Bhargavan, J. Protzenko, and B. Beurdouche, "Hacl*: A verified modern cryptographic library," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, Dallas, Texas, USA: ACM, 2017, pp. 1789–1806, ISBN: 978-1-4503-4946-8. DOI: 10.1145/3133956.3134043. [Online]. Available: https://eprint.iacr.org/2017/536.

[9] A. Erbsen, J. Philipoom, J. Gross, R. Sloan, and A. Chlipala, "Systematic generation of fast elliptic curve cryptography implementations," MIT, Cambridge, MA, USA, Tech. Rep., 2017. [Online]. Available: https://people.csail.mit.edu/jgross/personal-website/papers/2018-fiat-crypto-pldi-draft.pdf.

[10] T. Schulz, F. Golatowski, and D. Timmermann, "Evaluation of a formalized encryption library for safety-critical embedded systems," in *Industrial Technology (ICIT), 2017 IEEE International Conference on*, IEEE, 2017, pp. 1153–1158, ISBN: 978-1-5090-5320-9. DOI: 10.1109/ICIT.2017.7915525.

[11] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering*, pp. 77–89, 2012, Document ID: , http://ed25519.cr.yp.to/.