computing@computingonline.net
www.computingonline.net

Print ISSN 1727-6209
On-line ISSN 2312-5381
International Journal of Computing

INTEGRATION OF HARDWARE SECURITY MODULES INTO A DEEPLY EMBEDDED TLS STACK

Oliver Kehret, Andreas Walz, Axel Sikora

Institute for Reliable Embedded Systems and Communications Electronics,
Offenburg University of Applied Sciences, Badstraße 24, D77652 Offenburg, Germany ,
okehret@stud.hs-offenburg.de, andreas.walz@hs-offenburg.de, axel.sikora@hs-offenburg.de

Abstract: The Transport Layer Security (TLS) protocol is a well-established standard for securing communication over insecure communication links, offering layer-4 VPN functionality. In the classical Internet TLS is widely used. With the advances of the Internet of Things (IoT) there is an increasing need to secure communication on resource-constrained embedded devices. On these devices, computation of complex cryptographic algorithms is difficult. Additionally, sensor nodes are physically exposed to attackers. Cryptographic acceleration and secure hardware security modules (HSMs) are possible solutions to these challenges. The usage of specialized cryptographic modules for TLS is not a new phenomenon. However, there are still few hardware security modules suitable for the use on microcontrollers in sensor networks. We therefore present an overview of HSM and TLS solutions along with sample implementations and share some recommendations how to combine both.

Keywords: hardware security module, HSM, Transport Layer Security, Embedded Systems, cryptography, hardware acceleration, Internet of Things.

1. INTRODUCTION

In the Internet, data communication has to be secured to ensure safe functionality, confidentiality, integrity, authentication, non-repudiation, and availability. These requirements shall be observed not only in legacy Internet applications, but also in the machine-to-machine-(M2M)-type communication of the rapidly growing Internet of Things (IoT). A well-established and very widely used standard to secure communication over reliable channels is the Transport Layer Security [1] (TLS) protocol, initially invented by NetScape [2] Inc. as Secure Sockets Layer (SSL). With Datagram Transport Layer Security [3] (DTLS) a solution based on unreliable channels is available, too. (D)TLS combines strong, standardized cryptographic algorithms to achieve data confidentiality, integrity, and authenticity [4] of the communication partners involved.

However, (D)TLS only provides means to secure the data transport, leaving out devices and data at rest. In the classical internet a server was placed in a data center with rigid access controls. Its counterpart, the client PC or notebook, was located at home or in the company. The location of the devices made physical attacks difficult. In a world with ubiquitous communication and computing nodes many of such devices may be physically exposed to attackers [5], opening new attack vectors. Communicating entities might not only be attacked at the communication, but also be hijacked at communication ends.

This new development makes it particularly important to secure cryptographic material. According to Kerckhoffs' principle [6], the only thing that has to be and should be kept secret is the cryptographic material.

Thus, hardware security modules (HSM) with secure memory are a major ingredient to establish an anchor of trust. For an attacker, cryptographic keys stored in a HSM may be much more difficult to extract. But HSMs can do more, most of them providing e.g. a high-quality True Random Number Generator (TRNG). Good random numbers [7] are a basic requirement for the secure execution of cryptographic algorithms. Furthermore, most HSMs are capable of efficiently and securely executing different cryptographic operations on-chip, avoiding the need to reveal cryptographic keys to off-chip entities.

This paper provides an introduction to the usage of hardware security modules as cryptographic providers for implementations of TLS in embedded

systems. The paper is structured as follows: Chapter 2 includes an overview of available security module types. The TLS protocol along with cryptographic algorithms and implementations is outlined in Chapter 3. Chapter 4 describes the integration of HSMs into an existing TLS stack. In Chapter 5 related work is reviewed. Chapter 6 closes with a conclusion.

2. HARDWARE SECURITY MODULES

The term Hardware Security Module (HSM) is neither standardized nor normed. In the following we will give an overview of what can be considered a HSM and of the corresponding particularities.

2.1 SMART CARD

Smart Cards are widely used security devices for personal identification and authentication. Electronic tickets [8], credit cards, passports [9], and others rely heavily on the use of smart cards. According to Mayes [10] a smart card is a device that

- is able to participate in automated electronic transaction,
- adds security and is not easily forged or copied,
- stores data securely, and
- runs security algorithms.

Smart cards typically tend to be bound to a specific human. The owner carries the smart card along, typically. One might think of an insurance agent unlocking his PC with a smart card. In most of the application scenarios smart cards are lacking a power supply. Usually the power comes from the reader device. An Automated Teller Machine (ATM) is a typical example of a smart card reader. Due to the large number of cards deployed, smart cards have to be low cost. Smart cards are typically owned by the issuer, which in the previous example would be the bank.

2.2 TRUSTED PLATFORM MODULE

Trusted Platform Modules (TPMs) and the smart cards share many features. The goal of both is to have a secure computing environment and both have to be low cost. The TPM standard was originally developed by the Trusted Computing Group (TCG). Later on, the TPM was standardized by both the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). The TCG introduces the term trusted computing (TC). Trust in that context means "that a device behaves in a particular manner for a specific purpose" [11].

In the following we will highlight some capabilities that are unique to the TPM specification. For a more detailed overview the interested reader can refer to [12].

According to the specification TPMs are shipped in a pre-programmed state following the customer's requests [11]. During initial startup an entity has to take "ownership" of the TPM. Taking ownership is the process of establishing a shared secret between the TPM and the user. Once ownership is taken and the TPM is activated all features are available.

TPM specification introduces to secure storage functions: Binding and Sealing. Binding allows to store data encrypted by the TPM on external storage and Sealing locks data to a specific platform using a set of integrity metrics. In other words bound data may be decrypted having the appropriate key, whereas sealed data may only be decrypted having the right key and the right platform.

The TPM specification, however, does not dictate the use of any specific hardware nor that the TPM has to be an Integrated Circuit. Communication interfaces and bus architectures are determined by the manufacturer.

A dedicated RSA-2048 [13] engine is a requirement. However, 2048 bit keys are only needed for some restricted operations. For most of the operations the key length is arbitrary.

A high-quality True Random Number Generator is required to seed a Pseudo Number Generator. Random numbers are used to generate RSA keys and random nonce.

It should be noted that symmetric algorithms are foreseen in the specification for internal use only.

2.3 HSM FOR THE INTERNET OF THINGS

An emerging class of new HSMs is designed for the Internet of Things. This upcoming class is today often based on existing smart card modules. However, some differences can be observed.

This new kind of HSMs features a common industrial bus interface for connection to the host MCU. Power is provided over the host MCU's power supply.

Its main purpose is to provide cryptographic algorithms and secure storage of cryptographic keys. The algorithms and the strength used may vary according to the HSM's cost. For state-of-the-art implementations elliptic curve cryptographic acceleration, due to its computational efficiency, is typical.

A setup using one HSM for one MCU is expected. However, if required also multiple HSMs on one MCU should be possible, depending on the actual implementation. An application with high traffic or a need to store more keys can likely benefit from multiple HSMs.

EVITA [14] modules are a special kind of HSMs developed mainly for the use in car gateways. Especially the full EVITA module specification has a fitting feature set for a HSM, too.

3. TRANSPORT LAYER SECURITY

The Transport Layer Security (TLS) protocol is a mature standard securing communication over reliable, connection-oriented channels. For communication over unreliable channels the Datagram Transport Layer Security (DTLS) protocol was established. To the best of the authors' knowledge the current standard (D)TLS1.2 can be considered secure if implemented and configured carefully. (D)TLS preserves the confidentiality, authenticity, and integrity of the messages sent. The protocols are following a strict client and server model. The use of a public key infrastructure [15] spares the need to share a secret in advance.

3.1 THE INGREDIENTS OF THE TLS PROTOCOL

(D)TLS is divided in five sub-protocols (cf. Fig. 1). The Record Protocol is responsible for fragmentation of the messages. Furthermore, it provides confidentiality by means of bulk data encryption and message integrity by means of message authentication codes [16] (MAC).

The Change Cipher Spec Protocol is used to switch to encrypted communication and the Alert Protocol is used to convey error messages. The Application Data Protocol exchanges raw application data between the application and the transport layer.

The Handshake Protocol is used to establish a connection between two communication partners. Both partners have to prove their authenticity and to agree upon a key. Connection establishing is the computationally most expensive period of a TLS connection. During a handshake several complex asymmetric cryptographic operations are applied. The establishment starts with the exchange of Hello messages between the client and the server, followed by negotiations of the protocol version and the algorithms used.

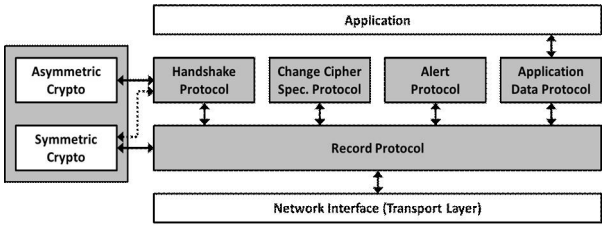


Fig. 1 – The TLS protocol with its sub-protocols in the ISO OSI model.

Therefore, (D)TLS introduces the concept of cipher suites. A cipher suite defines algorithms to preserve message integrity, authenticity, and application data encryption. The exchange of certificates and shared secrets is the last step in the handshake protocol. The algorithms involved are depending on the type of public key infrastructure used. For certificate verification the RSA algorithm or the digital signature algorithm (DSA) may be used. For key exchange the Diffie-Hellman algorithm [17] with static (DH) or ephemeral keys is one option. Public key algorithms are available in a version based on elliptic curves [18] called elliptic curve digital signature algorithm (ECDSA), elliptic curve Diffie-Hellman [19] (ECDH) or ephemeral elliptic curve Diffie-Hellman (ECDHE).

3.2 CRYPTOGRAPHY FOR (D)TLS

The security of the TLS protocol is based on strong, well-established cryptographic algorithms. The implementation of cryptographic algorithms can suffer from many flaws. Therefore it is strongly recommended to use a cryptographic library written by experts rather than develop a custom one.

TLS defines algorithms in cipher suites. A cipher suite consists of:

- a hash function (PRF, MACs and signatures),
- a key agreement function (for forward secure key exchange),
- asymmetric ciphers (for key exchange),
- symmetric ciphers (for bulk data encryption).

For the use on embedded systems cipher suites using elliptic curve algorithms for key agreement (ECDH) and signatures (ECDSA) are recommended [20].

3.3 LEGAL REQUIREMENTS

In Germany and the United States governmental authorities enact laws for the application of TLS in specific fields.

The German Federal Office for Information Security (BSI) obligates the use of BrainpoolP256r1 elliptic curve [21] and higher for ECDH and ECDSA in smart meter gateways. For authentication the algorithms have to be executed in a HSM. BSI doesn't allow the use of RSA or DH schemes for that use case.

National Security Agency (NSA) published new requirements for protection up to top secret [22]. NSA relies on NIST-P384 curves for ECDH and ECDSA. RSA and DH should be used with a modulus of at least 3072 bit.

3.4 TLS PROTOCOL AND CRYPTOGRAPHIC IMPLEMENTATIONS

This section gives an overview of the TLS implementations available on the market as well as cryptographic providers which can be used. In the following the term TLS is used for TLS and DTLS unless stated otherwise.

Pure cryptographic implementations

LibTomCrypt[1] is an open source cryptographic library written in C. It is a mature solution supporting many widely and some less often used algorithms.

MicroECC[2] is a library focusing on very efficient implementations of elliptic curve cryptography. Currently it supports ECDH and ECDSA on four elliptic curves: SECP128R1, SECP192R1, SECP256R1, and SECP384R1 [23]. MicroECC is mainly written in C but comes with optimized assembler routines for many popular microcontrollers.

The STM32 Cryptographic Library[3] is a cryptographic library designed by STMicroelectronics for the STM32 series of microcontrollers. On suitable hardware the library makes use of cryptographic hardware acceleration. However, this is a feature compatible with STM32 microcontrollers exclusively. The sources of the STM32 Cryptographic Library can't be reviewed as it is available in compiled form only.

Atmel ATECC508A[24] is a new HSM based on Atmels CryptoAuthentication series. It mainly offers ECDSA and ECDH based on the NIST-P256 [25] elliptic curve. For secure usage of the algorithms it also comes with a TRNG and secure key storage for up to 16 keys.

VaultIC460 [26] is a FIPS-140-2 [27][28] certified [29] HSM manufactured by Inside Secure. Based on an AT90SO128 microcontroller design it offers various symmetric and asymmetric algorithms. RSA and ECDSA may be freely parameterized by the developer. The VaultIC460 is available with several communication protocols including SPI and USB. Communication to the host MCU might also be secured using Secure Channel Protocol 02 (SCP02) or SCP03 [30]. SCP are a set of protocols specified by GlobalPlatform[4] and designed for the use on Smart Cards. More details can be found in section 4.2.

Pure TLS implementations

S2N[5] is a relatively new TLS implementation (2015) developed by Amazon Web Services - Labs. The goal of the project was to implement a small and better maintainable alternative to OpenSSL. It provides a full implementation up to TLS1.2 with some weak ciphers disabled by default. However, it makes use of the OpenSSL cryptographic library[6] for cryptographic algorithms.

emb::TLS [31] is a TLS stack developed for deeply embedded devices by the institute of the authors. The TLS stack is capable of all protocol versions from SSL3.0 up to TLS1.2.Written in pure ANSI C, it provides high portability. Thanks to the generic cryptographic interface (GCI) used in emb::TLS, the cryptographic provider may be exchanged seamlessly. The GCI removes all dependencies between the TLS application and the underlying cryptographic library. A reference version uses the LibTomCrypt library for cryptographic operations.

Bundled implementations

mbedTLS[7], formerly known as PolarSSL, is a TLS implementation developed by ARM. mbedTLS is delivered bundled with its own cryptographic library. It is mainly optimized for deployment on microcontroller with ARM-based CPUs.

OpenSSL[8] is the most widely known TLS implementation. A lot of forks exist like LibreSSL[9] developed for FreeBSD[10] and BoringSSL[11] developed for internal usage at Google Inc.. OpenSSL is both a TLS implementation and a cryptographic library. For deployment on resource-constrained embedded systems it is rather huge and makes extensive use of heap allocation for cryptographic computations.

GnuTLS[12] is a TLS implementation developed by the Free Software Foundation. Like OpenSSL GnuTLS is mainly developed for usage in a desktop environment. Licensed under the GNU Lesser General Public License, version 2.1 (LGPLv2.1) [32] it is possible to use GnuTLS in a lot of GPL licensed projects where other license types may not be compatible. It supports TPMs as well as PKCS #11 [33] Smart Cards natively. TPM can be used for RSA key generation and signing. For comfortable access to PKCS #11 Smart Cards GnutTLS provides

[1] https://github.com/libtom/libtomcrypt
[2] https://github.com/kmackay/micro-ecc
[3] http://www.st.com/web/en/catalog/tools/PF259409
[4] https://www.globalplatform.org
[5] https://github.com/awslabs/s2n
[6] https://www.openssl.org/docs/manmaster/crypto/crypto.html
[7] https://github.com/ARMmbed/mbedtls
[8] https://www.openssl.org/
[9] http://www.libressl.org/
[10] https://www.freebsd.org/de/
[11] https://boringssl.googlesource.com/boringssl/
[12] http://www.gnutls.org/

an application interface, mainly to store and load sensitive information.

TinyDTLS[13] is a DTLS implementation especially designed for resource-constrained embedded systems. It is delivered with its own cryptographic implementation. TinyDTLS lacks TLS support.

WolfSSL[14] is a mature TLS library developed by wolfSSL Inc. It is targeted on embedded devices, IoT devices as well as desktop environments. It is about 20 times smaller than OpenSSL. An OpenSSL compatibility layer ensures good interoperability with server and desktop systems.

TLS implementations with hardware support

Atmel Hardware-TLS[15] is a TLS solution that allows usage of the Atmel ATECC508A security module as a crypto provider. It can be used based on OpenSSL[16] or WolfSSL[17]. The implementation basically outsources some cryptographic operations to the hardware security module. The OpenSSL development is done out of the regular development tree. This can be a problem if the vendor decides to cancel support for these specialized TLS implementations.

4. INTEGRATION OF HARDWARE SECURITY MODULES

The integration of hardware security modules is a challenge as compared to a pure software solution many different measures have to be performed.

For the sample application we used a STM32F4 discovery board [34] with the STMCubeF4 Hardware Abstraction Layer (F4HAL) [35]. Most of the software development is done using the libraries provided freely by the manufacturers to ensure comparability. Both HSMs, the VaultIC460 and the ATECC508A, were tested independently. Both HSMs are programmed using their respective APIs[18]. The TLS implementation used is emb::TLS developed at the authors' institute.

4.1 PHYSICAL CONNECTION

Both HSMs are connected to the microcontroller unit (MCU) using standard communication interfaces. The VaultIC460 is used with a Standard Peripheral Interface Bus (SPI) operated at 2 MHz. This guarantees a high speed connection, however, at the price of eight[19] wires to be connected. Additional wires are used for selection of the bus protocol and further HSM specific adjustments. The ATECC508 is connected via I^2C clocked at 40 kHz. The I^2C bus offers significantly lower data rates but only needs three wires to work. In addition the ATECC508A is only capable of executing asymmetric cryptographic operations. As a consequence there is no bulk data encryption expected and only small amounts of data have to be transmitted via this link.

4.2 LOGICAL CONNECTION

From an abstract point of view the implementation is compounded of three main parts: The TLS application, the generic cryptographic interface (GCI) (cf. Fig. 2) and the cryptographic provider. The cryptographic provider computes the crypto graphic function needed by the application. To prevent a situation in which the application is totally dependent on one specific provider the calls are abstracted thanks to the GCI. Using the GCI, crypto providers may be changed seamlessly.

The connection between the application, running on the MCU, and the provider, on the HSM, is a weak link. Data may be exchanged in plaintext over this channel. To prevent unauthorized change of components, HSM and MCU should provide means for mutual authentication. There are some solutions to that problem on the market. VaultIC460 can be used with the Secure Channel Protocol (SCP) version '02 or '03 [30]. However, HSMs capable of the more recent SCP11 [36] are still missing. SCP03 uses symmetric algorithms like DES or AES to encrypt the communication. For authentication a password along with username may be used. ATECC508A, missing symmetric algorithms, is only capable of authentication. Due to this, data has to be sent unencrypted over the channel between MCU and HSM.

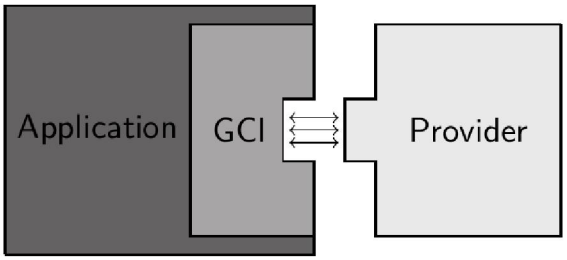


Fig. 2 – The Generic Cryptographic Interface is linking the application and the cryptographic provider. The GCI is an abstraction layer and makes the application independent from the specific provider.

[13] https://projects.eclipse.org/projects/iot.tinydtls
[14] https://wolfssl.com/wolfSSL/Home.html
[15] http://www.atmel.com/Images/Atmel-45176-Hardening-Transport-Layer-Security-for-IoT_Flyer.pdf
[16] https://github.com/AtmelCSO/cryptoauth-openssl-engine
[17] http://www.atmel.com/tools/Atmel-HW-TLS.aspx
[18] http://www.atmel.com/tools/CryptoAuthLib.aspx

[19] SPI/I^2C may need fewer wires, however the HSMs need the additional wires to work

4.3 KEY MANAGEMENT

Key management is potentially the most significant difference of HSMs compared to a pure software TLS solution. Storing keys on a MCU is rather trivial. Most MCUs are equipped with sufficient storage space. That is, as long as some free storage is available one can easily create or import a new key. In a pure software solution key management is done in software.

Using a HSM each module type is shipped with a specific key management solution. VaultIC460 offers 112 kByte of secured memory for storage of keys, certificates, and arbitrary confidential data. Key import and export is possible depending on the mode selected on the HSM. In normal mode export and import of private and public keys is possible without restrictions. In secure mode private keys may never leave the device. ATECC508A comes with a different approach. It is capable of storing 16 keys. Public keys may be imported, private keys along with their public counter parts can be generated on the HSM only.

The capability to store keys in a secure manner is one of the most important features of a HSM.In general, secure slot memory can be written only a limited number of times because of the used EEPROM. Due to that a system has to ensure that the usage of key slots is balanced and distributed. The used HSMs have no functionality to get a list of free slots. The best solution to check for a free slot is simple trial and error. For this we propose a mechanism that uses the TRNG of the HSM to determine a slot to store. The algorithm basically seeds the software PRNG with a true random number and then determines the slot by creation of pseudo random numbers in the given range. The TRNG is used to seed the C standard library RNG. The algorithm tries up to N times to find a free slot, if no free slot is found after that the supposed slot is freed and written to. The collision probability for this is relatively low having a lot of slots (cf. Fig. 3).

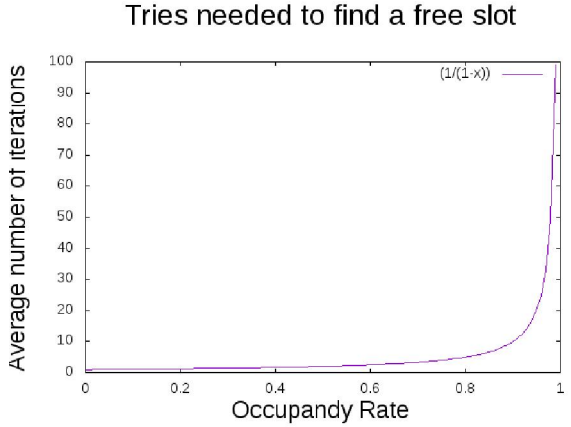


Fig. 3 – The average number of iterations needed to find a free slot as a function of the slot occupancy.

Another solution is to store a table of slots used on the MCU. However, this would require extra persistent memory on the MCU and we don't want to rely on extra hardware. It is expected by the authors that a sudden loss of power is likely. If the loss of power happens before writing information to the storage, information in the table is wrong. No known HSM allows the user to get information about the key stored in a slot.

4.4 SOME CHALLENGES

HSMs, like other hardware, tend to have a fixed, specific instruction set. Depending on the possibility for a good or bad implementation in hardware it may take longer or shorter until HSMs with new algorithms appear on the market. To the authors' knowledge it is more likely to take longer until new HSM featuring new algorithms are released [37].

Private keys might be stored very securely in a HSM. However, keys for authentication or encryption of communication with the HSM have to be stored on the MCU. As a result the keys on the MCU are always the weak point. This situation could be improved by adding some secure storage to the MCU, too.

4.5 LESSONS LEARNED

Hardware-based TRNGs [38], providing much more entropy compared to software solutions are a huge plus, as high-quality random numbers are required for basically every cryptographic algorithm used in TLS. Weak random numbers are often the reason of breaches in cryptographic protocols [39].

Efficiency of cryptographic algorithms computed in a HSM compared to MCU computations for both speed and energy is a more controversial topic. Sophisticated HSM algorithms are able to outperform software implementations [40] on both energy efficiency and computation time. However, using a different MCU, software libraries, or HSM as well as algorithms the situation may change significantly.

5. CONCLUSION

The design of secure sensor networks is a delicate task. The solutions available have to be combined carefully. (D)TLS with strong, cryptographic algorithms is a secure base. HSMs can add more efficient algorithm computation, better random numbers and secure key storage to the system. However, HSMs have to be selected carefully. Updates are in general not possible and the set of cryptographic algorithms is very small compared to software implementations. If using a subsidiary software library even the software

algorithms can benefit from the true random numbers provided by the hardware implementation.

However, actual deployment of a HSM mainly depends on the physical vulnerability of the system. System cost can also be saved by not upgrading to a better MCU and offloading computational expensive tasks to the HSM. This has to be decided as the case arises.

In future research the energy consumption and the computing time of the HSM algorithms have to be measured. Based on this numbers and previous research [41] assumptions about the energy efficiency and speed of the hardware-accelerated emb::TLS can be made. The results have to be verified using a physical test setup.

Furthermore, it is planned to extend the hardware support of emb::TLS in the future. Extern HSMs can be supported by symmetric cryptographic units integrated in the MCU, e.g. the STM32F417xx series, providing AES, HMAC, and a hardware RNG.

With the market launch of appropriate products we want to integrate HSMs ready for governmental use in Germany and the United States.

6. REFERENCES

[1] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2 RFC5246*, http://www.ietf.org/rfc/rfc5246, accessed March 2016.

[2] R. Oppliger, *SSL and TLS: Theory and Practice*, Artech House, 2009.

[3] E. Rescorla and N. Modadugu, *Datagram Transport Layer Security Version 1.2 RFC7507*, available online on http://www.ietf.org/rfc/rfc6347, accessed March 2016.

[4] Legal Information Institute, *U.S. Code § 3542 - Definitions*, https://www.law.cornell.edu/uscode/text/44/3542, accessed March 2016.

[5] M. Abomhara et al., Cyber Security and the Internet of Things: Vulnerabilities, Threats, Intruders and Attacks, *Journal of Cyber Security*, (4) 1 (2015), pp. 65-88.

[6] A. Kerckhoffs, La cryptographie militaire, *Journal des sciences militaires*, 1883.

[7] D. Eastlake et al., *Randomness Requirements for Security RFC4086*, http://www.ietf.org/rfc/rfc4086, accessed March 2016.

[8] W. H. Tan, *Practical Attacks on the MIFARE Classic*, Imperial College London, http://www.doc.ic.ac.uk/~mgv98/MIFARE_files/report.pdf, accessed March 2016.

[9] Bundesdruckerei, *ePassport Pocket Guide 2013*, https://www.bundesdruckerei.de/sites/default/files/documents/2013/08/pocketguide_epass_en.pdf, accessed March 2016.

[10] K. Mayes, *An Introduction to Smart Cards*, in: Smart Cards, Tokens, Security and Applications, Springer US, 2008, pp. 155-172.

[11] TCG, *TCG Specification Architecture Overview*, http://www.trustedcomputinggroup.org/files/resource_files/AC652DE1-1D09-3519-ADA026A0C05CFAC2/TCG_1_4_Architecture_Overview.pdf, 2007, accessed March 2016.

[12] A. Tomlinson, *Introduction to the TPM*, in: Smart Cards, Tokens, Security and Applications, Springer US, 2008, pp. 155-172.

[13] R.L. Rivest et. al., *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, https://pdfs.semanticscholar.org/21b2/34ff1ec4b42fb84f5f27f4de1a2cd05d7f2b.pdf, 1978, accessed March 2016.

[14] M. Wolf, T. Gendrullis, Design, Implementation, and Evaluation of a Vehicular Hardware Security Module, in Proceeding of the 14th International Conference on Information Security and Cryptology ICISC'11, Springer-Verlag Berlin, Heidelberg, 2011, pp. 302-318.

[15] M. Cooper et al., *Internet X.509 Public Key Infrastructure: Certification Path Building RFC4158*, http://www.ietf.org/rfc/rfc4158, accessed March 2016.

[16] H. Krawczyk et al., *HMAC: Keyed-Hashing for Message Authentication RFC2104*, http://www.ietf.org/rfc/rfc2104, accessed March 2016.

[17] E. Rescorla, *Diffie-Hellman Key Agreement Method RFC2631*, available online on http://www.ietf.org/rfc/rfc2631, accessed March 2016.

[18] S. Blake-Wilson et al., *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) RFC4492*, http://www.ietf.org/rfc/rfc4492, accessed March 2016.

[19] D. McGrew et al., *Fundamental Elliptic Curve Cryptography Algorithms RFC6090*, http://www.ietf.org/rfc/rfc6090, accessed March 2016.

[20] N. Gura et al., *Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs*, https://www.iacr.org/archive/ches2004/31560117/31560117.pdf, accessed March 2016.

[21] BSI, TR-03116-3, *Kryptographische Vorgaben für Projekte der Bundesregierung*, 2015.

[22] NSA, *NSA Suite B Cryptography*, 2015, https://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml#guides.

[23] *Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters*, http://www.secg.org/SEC2-Ver-1.0.pdf, accessed March 2016.

[24] Atmel Inc., *ATECC508A Summary Datasheet*, http://www.atmel.com/images/atmel-8923s-cryptoauth-atecc508a-datasheet-summary.pdf, accessed March 2016.

[25] National Institute of Standards and Technology, *Recommended Elliptic Curves for federal Government use*, http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf, accessed March 2016.

[26] Inside Secure, *VaultIC460 Summary Datasheet*, http://www.insidesecure.com/content/download/1381/8640/version/2/file/SummaryVIC460_6606CS.pdf , accessed March 2016.

[27] Inside Secure, *FIPS PUB 140-2 Non-proprietary Security Policy*, http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1762.pdf, accessed March 2016.

[28] National Institute of Standards and Technology, *FIPS PUB 140-2 Security Requirements for cryptographic modules*, http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf, accessed March 2016.

[29] National Institute of Standards and Technology, *Validated FIPS 140-1 and FIPS 140-2 Cryptographic Modules*, http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm, accessed March 2016.

[30] *GlobalPlatform: Card Technology Secure Channel Protocol '03' Card Specification v2.2 – Amendment D V1.1.1*, http://www.globalplatform.org/specificationscard.asp, accessed March 2016.

[31] A. Yushev et. al, *Securing Embedded Communication with TLS1.2*, 2015.

[32] *Free Software Foundation, GNU Lesser General Public License, version 2.1*, http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html, accessed March 2016.

[33] RSA Laboratories, *PKCS #11 Base Functionality v2.30: Cryptoki – Draft 4*, ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-30/pkcs-11v2-30b-d6.pdf, accessed March 2016.

[34] STMicroelectronics, *STM32F4DISCOVERY*, http://www.st.com/web/en/resource/technical/document/data_brief/DM00037955.pdf, accessed March 2016.

[35] STMicroelectronics, *STM32CubeF4*, http://www.st.com/st-web-ui/static/active/en/resource/technical/document/data_brief/DM00103572.pdf, accessed March 2016.

[36] GlobalPlatform, *Card Secure Channel Protocol '11' Card Specification v2.2 – Amendment F v1.0*, http://www.globalplatform.org/specificationscard.asp, accessed March 2016.

[37] PRNewswire, *Atmel First to Ship Ultra-Secure Crypto Element Enabling Smart, Connected and Secure Systems*, http://www.prnewswire.com/news-releases/atmel-first-to-ship-ultra-secure-crypto-element-enabling-smart-connected-and-secure-systems-300036172.html, accessed March 2016.

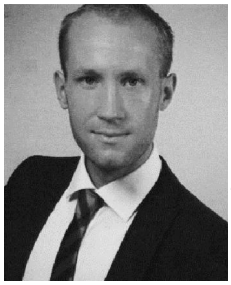
[38] National Institute of Standards and Technology, *Recommendation for the Entropy Sources Used for Random Bit Generation, NIST SP 800-90B*, http://csrc.nist.gov/publications/drafts/800-90/sp800-90b_second_draft.pdf, accessed March 2016.

[39] F. D. Garcia et al., Computer Security, in Proceedings of the 13th European Syposium on Research in Computer Security ESORICS'08:, Málaga, Spain, 2008, Springer Berlin, Heidelberg, Chapter: Dismantling MIFARE Classic, pp. 97-114.

[40] M. Koschuch et al., *Hardware/Software Co-Design of Elliptic Curve Cryptography on an 8051 Microcontroller*, https://www.iacr.org/archive/ches2006/34/34.pdf, accessed March 2016.

[41] N. A. Kofi et al., *Embedded TLS 1.2 Implementation for Smart Metering & Smart Grid Applications*, 2015.

***Oliver Kehret**, holds a B. Eng. Degree from Offenburg University of Applied Sciences. He is currently pursuing the M.S. degree with University of Applied Sciences Offenburg, Germany. His research interest includes security algorithms and protocols for resource constrained devices.*



***Andreas Walz,** holds a diploma in Physics from the University of Freiburg. From his studies and from working as a research assistant in experimental particle physics he has many years of experience in the development of embedded hardware systems as well as efficient software. Currently, he is pursuing his Ph.D. in the field of security in embedded systems with special interest in the TLS protocol family.*

Axel Sikora, *holds a diploma of Electrical Engineering and a diploma of Business Administration, both from Aachen Technical University. He has done a Ph.D. in Electrical Engineering at the Fraunhofer Institute of Microelectronics Circuits and Systems, Duisburg, with a thesis on SOI-Technologies. After positions in the telecommunications and semiconductor industry, he became a professor at the Baden-Wuerttemberg Cooperative State University Loerrach in 1999. In 2011, he joined Offenburg University of Apllied Sciences, where he leads the institute for reliable embedded systems and communications electronics. His major interest is in the field of efficient, energy-aware, safe and secure algorithms and protocols for wired and wireless embedded communication. In 2002 he founded the Steinbeis Transfer Center Embedded Design and Networking for professional protocol and platform developments, which was successfully spun off as STACKFORCE GmbH in 2014. Dr. Sikora is author, co-author, editor and co-editor of several textbooks and numerous papers in the field of embedded design and wireless and wired networking, and head and member of manifold steering and program committees of international scientific conferences.*