embeddedworld2019
Exhibition&Conference
... it's a smarter world

Security on IoT Devices with Secure Elements

Tobias Schläpfer, Andreas Rüst
Zurich University of Applied Science (ZHAW)
Institute of Embedded Systems (InES)
Winterthur, Switzerland
tobias.schlaepfer@zhaw.ch
andreas.ruest@zhaw.ch

***Abstract*—The emergence of new low power IoT networks in which leaf nodes have native IPv6 connectivity and the grown awareness for data protection of IoT devices require leaf nodes to provide a higher level of security, similar to the level of a standard computer system. Especially in terms of energy consumption and device cost, the intensive cryptographic operations of well-known computer security algorithms are a big challenge for resource constrained devices. To face these challenges, semiconductor vendors have recently introduced new dedicated hardware, so called secure elements. These devices provide hardware accelerated support for cryptographic operations and tamper proof memory for the secure storage of cryptographically sensitive material. Moreover, they employ specific techniques against so-called side channel attacks. The paper describes and specifies different classes of secure elements and discusses their opportunities and challenges. Furthermore, the paper provides multiple detailed examples how secure elements can be used for different applications. Finally, this paper briefly presents general measurement results from a performed evaluation with four selected secure elements from different vendors. A more complete report about the performed evaluation will be presented in a following paper. The purpose of this paper is to introduce the concept of secure elements and provide a generic overview of their features, serving as starting point to work with secure elements.**

Keywords—IoT security, secure elements, hardware cryptography, authentication, tamper proof memory, side channel protection, resource constrained devices, Thread network,

I. INTRODUCTION

Newly introduced IoT networks such as Thread [1], which is based on the 802.15.4 standard [2], provide native Internet Protocol version 6 (IPv6) connectivity down to the resource constrained device. This paper refers to resource constrained devices as leaf nodes and understands them as a battery powered electronic device which is controlled by a microcontroller unit (MCU). The native IPv6 connectivity offers many opportunities such as service discovery, end-to-end security and transparent routing down to the leaf node. Due to the transparent routing, there is also no need for a complex and energy consuming gateway. However, due to the missing gateway and straightforward accessibility, the leaf node is exposed to attacks from the outside world. Furthermore, the leaf node can be directly connected to the IT infrastructure and therefore may serve as a simple entry point for attackers. As a result, the required security level for a leaf node is much higher in such IoT networks. The challenge is to leverage the elaborate, well-known computer security, which otherwise would be provided by a gateway, to the resource constrained device without dramatically increasing the power consumption nor the cost with regard to memory sizes and required processor performance.

To face these challenges, semiconductor vendors have recently introduced dedicated hardware devices, so called secure elements. Secure elements execute cryptographic operations in hardware, which allows for fast and energy efficient execution of cryptographic algorithms. Furthermore, secure elements provide tamper proof memory for secure storage of cryptographic material. These features allow the leaf node to provide a high security level and to be an authentic and secure member of an IT infrastructure.

This paper is structured accordingly. First, we introduce the concept of secure elements by providing a categorization for secure elements and a detailed description how secure elements are used. In chapter three and four, we show opportunities as well as challenges of secure elements. Subsequently, we discuss energy and execution time measurements with four different secure elements we performed. The paper closes with appropriate conclusions.

II. SECURE ELEMENTS

Secure elements emerged from the field of security controllers used in smart / banking card applications. These security controllers offer communication interfaces compliant to the ISO7618 [3] standard and provide an operating system (OS) for their specific application. The application on the security controllers are typically written by external security experts. As awareness for security issues in the IoT industry

rises, semiconductor vendors strive to offer an easy to use, out-of-the-box solution with already integrated firmware specifically for IoT applications. Furthermore, these new secure elements offer serial communication interfaces to communicate with a main MCU on the IoT device. This paper focuses on these new secure elements for IoT applications.

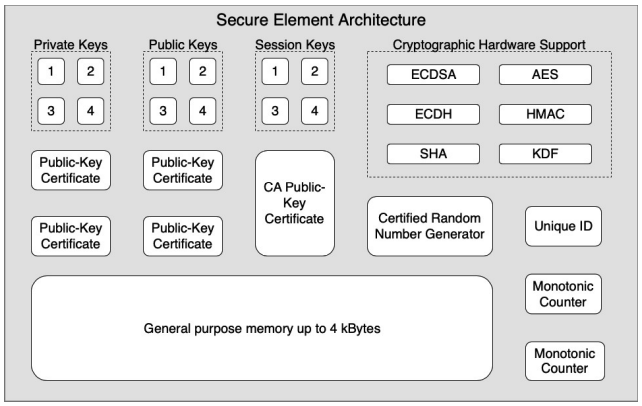


Fig. 1 Secure element architecture

Fig. 1 shows a generic secure element architecture. Secure elements support the MCU by executing standard cryptographic algorithms, e.g. Advance Encryption Standard (AES) [4] or Elliptic Curve Diffie-Hellman (ECDH) [5] operations, in hardware. Typically, all secure elements support elliptic curve cryptography (ECC) [6] on the NIST p-256 curve. The foundation of strong cryptography is a reliable entropy source. To provide such a reliable entropy source, secure elements are equipped with a certified true random number generator (TRNG). Furthermore, secure elements provide tamper proof memory to store cryptographic material, such as private keys, long living session keys or public-key certificates which serve as root of trust.

Additionally, secure elements provide up to 4 kBytes of general purpose memory to securely store e.g. application data. Furthermore, secure elements provide monotonic counters. These counters can only be increased and never be reset. They may be used in multiple ways, e.g. to restrict the use of certain keys to only a certain number of times or to prevent replay attacks. Secure elements also have a unique device number, which may be used to authenticate the secure element to the MCU, allowing to detect unauthorized substitution of the secure element.

Communication between the secure element and its host MCU is typically realized over an I^2C interface with max. data rates up to 1 Mbit/s. Table 1 in the appendix, summarizes common secure element features. However, there are secure elements that additionally to the I^2C offer a single-wire interface (SWI) or a serial peripheral interface (SPI).

Secure elements may be used in applications for various purposes such as authentication, secure storage and for cryptographic operations support. Applications include protection of intellectual property, device authentication, secure end-to-end channel establishment and many more.

A. Categories of secure elements

Yet not all secure elements have the same range of features nor do they target the same application use cases. Essentially, they can be divided into two categories.

First, there is the category of *supporting* secure elements. These secure elements are intended to support the software execution of cryptographic operations by a library such as mbedTLS [7], running on a host MCU. *Supporting* secure elements provide secure storage for sensitive material and hardware acceleration for a variety of cryptographic operations. This increases the leaf nodes security by physically isolating private keys and other secrets as well as by significantly reducing the energy consumption through the hardware accelerated execution. If a cryptographic software is needed on the MCU, e.g. because special cryptographic algorithms like the JPAKE [8] cipher suite are used, *supporting* secure elements are an appropriate choice.

The second category are *standalone* secure elements. These secure elements are capable to autonomously handle a security layer. To achieve this, they provide a complete set of cryptographic operations with additional control features to establish and maintain a secure and authentic connection without software support on the host MCU. Therefore, they have to be able to handle messages for a (Datagram) Transport Layer Security (D)TLS [9] session. Yet, *standalone* secure elements are also able to support a host MCU that uses a cryptographic software, like the *supporting* secure elements.

For both categories of secure elements, one can say that, if all the cryptographic operations are executed on a secure element, the overall security of an IoT device can be enhanced, because all the sensitive material for the cryptographic operations always remains within the tamper proof memory of the secure element.

B. Opportunities of secure elements

A typical IoT device is equipped with an MCU that has constrained processing power. Therefore, elaborate cryptographic operations, e.g. sign and verify operations of the elliptic curve digital signature algorithm (ECDSA) [10], have long execution times. This results in a high energy consumption. Since IoT devices are mostly battery powered, the high energy consumption therefore significantly reduces the battery lifetime by providing hardware acceleration for cryptographic operations. Secure elements enable resource constrained devices to execute elaborate cryptographic algorithms fast and energy efficient. Allowing even a battery-powered device to use well-known computer security algorithms.

Another issue with MCUs is that their memory is accessible, which means that it can be read or even manipulated with a debugger. This exposes sensitive material and makes stored data not reliable. Secure elements on the other hand provide secure and authentic data storage. Secure, due to the tamper

proof memory which protects the stored data against physical attacks, such as imaging or fault injection [11] and authentic because memory access on the secure element requires authentication by its host MCU before data can be read or altered. Additionally, there is also the issue that MCUs mostly offer only a limited amount of memory size. Due to this, MCUs may not be able to store multiple x509 certificates [12], since these certificates can be well up to 1 kBytes or even bigger. Secure elements provide storage capacity for multiple x509 certificates and further capabilities regarding certificates, such as the creation of certificate signing requests (CSR) [13]. Another key feature is the protection of private keys. It is impossible to extract any private key from a secure element, neither by software nor physically. The private key may be either generated inside the secure element itself or inserted in a secure environment at a manufacturer site.

IoT devices are typically physically accessible for an attacker, which exposes them to hardware attacks such as manipulation, brute force or side channel attacks. With a brute force attack, an adversary systematically tests all possible keys hoping to find the correct one. A side channel attack is an attack where an attacker tries to gain sensitive information by observing and analyzing properties of an IoT device, e.g. power consumption, execution time or electromagnetic leakage. Secure elements provide specific protection measures, such as secure boot, security monitoring or constant-time-implementation, to prevent hardware attacks.

C. Challenges of secure elements

Although, secure elements have many positive features, there are also challenges in using secure elements. The communication between the MCU and the secure element over I^2C poses a potential security threat. Since an attacker may probe this interface, he may be able to read content and to execute replay or fault injection attacks. However, there are multiple ways to secure the I^2C interface. One way is by encrypting or obfuscating the content on the I^2C lines. Therefore, a symmetric secret or asymmetric keys need to be inserted into the secure element and the MCU. However, this method has a major drawback, since the secret or key to decrypt messages is stored on the accessible memory of the MCU. Furthermore, not all secure elements offer the feature to obfuscate the I^2C communication. The second method is to protect the interface by actively shielding the I^2C lines. To achieve this, the lines must be placed between two reference layers (VCC or GND) of the PCB. This may result in higher design effort and cost but provides an effective protection against probing. However, most important is that the developer is aware of the problem and knows exactly which content may be transferred in plain text across the I^2C interface and implements according protection measures. The optimal solution is to never exchange sensitive material between the MCU and the secure element.

Another issue with secure elements is more an administrative one. Currently, if one would like to get detailed information about a secure element a non-disclosure agreement (NDA) between the customer and the secure element vendor has to be in place. Furthermore, the development effort and expertise required to integrate the individual secure element is extremely high. In this study, the integration of all four secure elements required thorough support from the field application engineers of the respective vendors for a successful integration into an existing application. These points pose major obstacles to widely adopt and deploy secure elements in IoT devices. Therefore, the authors of this paper see a strong need for an open information exchange and harmonization of the interfaces.

III. USAGE OF SECURE ELEMENTS

Secure elements contribute to applications in many different ways e.g. secure boot, secure messaging or (D)TLS. These three selected examples are presented in detail in the following section.

A. Example 1: Secure boot with secure elements

Secure boot is defined as a boot sequence in which the MCU, secure element pair is authenticated. This sequence is designed to detect and prevent manipulations either on the MCU or the secure element, e.g. to prevent the unauthorized execution of software on the MCU or unauthorized usage of cryptographic material stored on the secure element. Fig. 2 illustrates how a secure boot process may be executed.

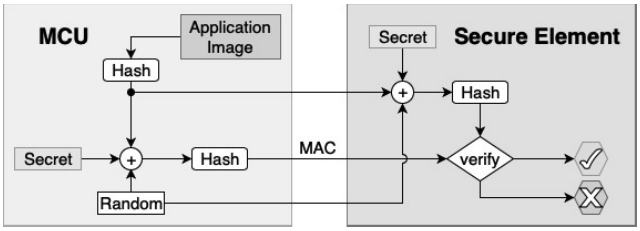


Fig. 2 Secure boot process

This process is executed during the boot process of an MCU. The secure boot process is initiated with the MCU hashing the Application Image (AI). The resulting digest is sent to the secure element. Furthermore, the AI digest combined with a symmetric secret and a random number, are hashed creating the message authentication code (MAC). The MAC as well as the random number are sent to the secure element. The secure element reproduces the MAC using the AI digest and random number received from the MCU as well as the symmetric secret stored on the secure element. This reproduced MAC is then compared with the MAC received from the MCU. If the two MACs coincide, the MCU, secure element pair is mutually authenticated. If the verification fails e.g. due to unauthorized exchange of the MCU, the secure element no longer allows the MCU to use stored keys and denies access to stored data. On the other hand, if the secure element has been exchanged, the MCU must not execute the application image due the secure boot failure.

To shorten the secure boot process, the AI digest may be stored on the secure element after the first successful boot process. This has also the advantage that an attacker is not able to get the AI digest by probing the I^2C lines and allows for the detection of unauthorized changes in the application image. The

random number is used to prevent replay attacks and may be generated either on the MCU or the secure element. The symmetric secret stored on both the MCU and the secure element obfuscates the process for an attacker and makes it impossible to guess the MAC even if an attacker is in possession of the AI digest and the random number. The secret is unique to each leaf node device and should be inserted at the manufacturer site, in a secure environment, e.g. during production testing. Additionally, even if an attacker finds the secret on the accessible memory of the MCU, only a single leaf node device is compromised. This secure boot could also be executed with asymmetric keys, by signing the AI digest on the MCU and verifying the signature with a protected public key on the secure element. However, the cryptographic algorithms needed for the asymmetric version may exceed the processor capacity of the MCU bootloader. [14]

B. Example 2: Secure messaging with secure elements

One of the key features of a secure element is the protection of private keys. Key pairs (private / public key) may be generated inside the secure element. The stored key pair can be used to en- / decrypt messages for the MCU.

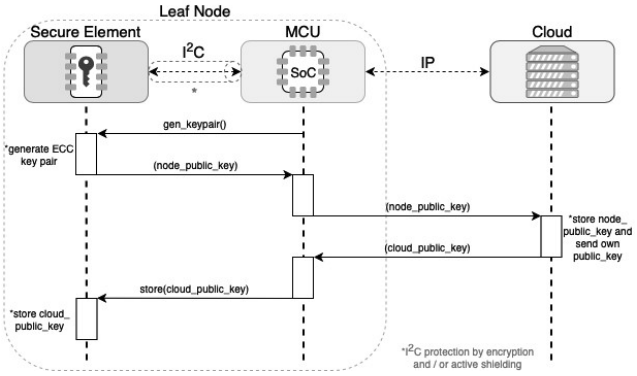


Fig. 3 Generation of an ECC public/private key pair on a secure element followed by an exchange of public keys between the leaf node and a cloud

Fig. 3 shows the key generation on a secure element followed by an exchange of public keys between a leaf node and a cloud server. A key point is, that the private key always remains on the secure element whereas the public key is sent to the cloud server. The cloud stores the public key of the leaf node and in exchange sends its public key to the leaf node, where the MCU forwards the public key to the secure element for storage.

Fig. 4 shows how the leaf node may use the previously exchanged keys, to en- / decrypt messages. As an example, the MCU first collects data from a sensor and generates the message. The message is then handed to the secure element, which encrypts the message using the previously stored public key of the cloud server. The encrypted message is then sent to the cloud. The cloud processes the message and sends an encrypted response to the leaf node. The MCU forwards this response to the secure element which uses its internally generated private key to decrypt it. The response is transferred to the MCU, which acts accordingly.

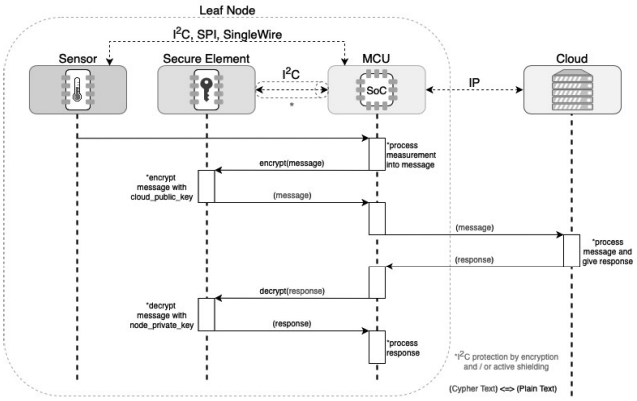


Fig. 4 Encrypted exchange of messages between leaf node and cloud

C. Example 3: DTLS session with secure elements

New IoT networks, such as Thread, provide IPv6 all the way down to the resource constrained device. The new IP connectivity on the leaf node offers the opportunity to establish secure end-to-end channels between a leaf node and a server, shown in Fig. 5.

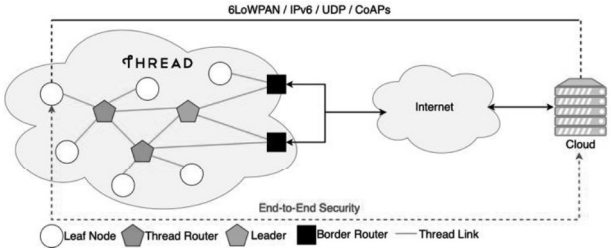


Fig. 5 Thread network with native IPv6 connectivity on the leaf node

A standard way to establish an authentic secure channel is by using the (D)TLS protocol. To establish a (D)TLS session a handshake needs to be executed first. During this process, the leaf node and the server present their respective public-key certificates to authenticate themselves and then use asymmetric keys to derive a shared secret. This shared secret is used as a symmetric key to en- / decrypt messages between the leaf node and the server. To authenticate the server to the leaf node, the leaf node requires the public-key certificate of the server's certificate authority (CA). This previously obtained and stored certificate allows to verify the server certificate presented during the handshake. The CA public-key certificate is the root of trust and therefore must be protected against manipulation. Accordingly, the public-key certificate should be stored in the secured memory of the secure element. Furthermore, the secure element can accelerate the handshake execution and therefore reduce energy consumption.

For both of the before discussed usage examples, secure boot and secure messaging, it does not matter whether a *supporting* or a *standalone* secure element is used. This changes for a (D)TLS handshake. *Supporting* secure elements only support the cryptographic software, in executing the handshake by executing cryptographic operations and providing trusted

memory storage. On the other hand, *standalone* secure elements are able to autonomously handle the handshake messages with no further support of the MCU.

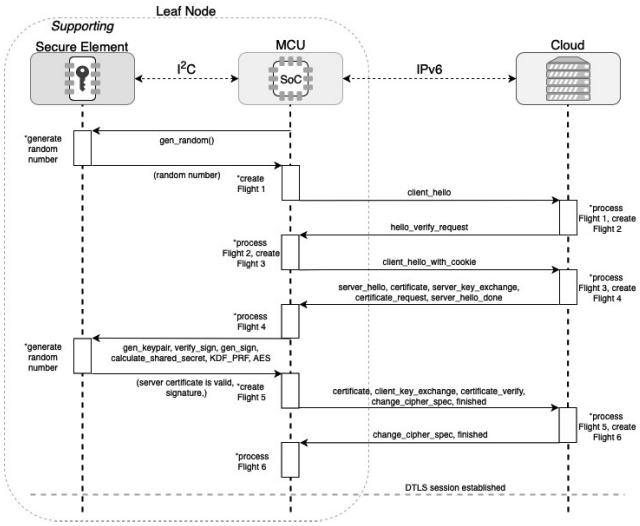


Fig. 6 DTLS handshake using a supporting secure element

Fig. 6 shows a DTLS handshake executed with the support of a *supporting* secure element. The MCU uses the supporting secure element to outsource specific cryptographic operations, e.g. ECDSA sign and verify ECDH calculation of the shared secret or en- / decryption with AES. This allows for faster execution of the handshake and increases the overall security since sensitive material such as the shared secret and the derived keys remain on the secure storage of the secure element. After the handshake is successfully executed, the messaging process is the same as in the secure messaging example. The difference is that with a (D)TLS session in place, the communication is not just encrypted but also authentic. This means, the leaf node knows to whom it is talking, whereas in the key exchange example it has not been able to verify that the cloud server is indeed who it claims to be.

Fig. 7 shows a DTLS handshake with a *standalone* secure element. These secure elements are able to autonomously handle the handshake process. The MCU only sends the messages from the secure element to the cloud and forwards the received messages to the secure element. By executing the DTLS handshake on the secure element it-self, all sensitive data is stored on the secure element alone. In addition, the secure element accelerates the execution of the handshake.

When selecting a secure element for a specific application that involves (D)TLS sessions, some important considerations regarding secure elements have to be made. First, *standalone* secure elements typically only support a single specific cipher suite. Cipher suites are defined in Internet Engineering Task Force (IETF) [15] standards and are used to determine the used algorithms for the key exchange, signature, verification and encryption.

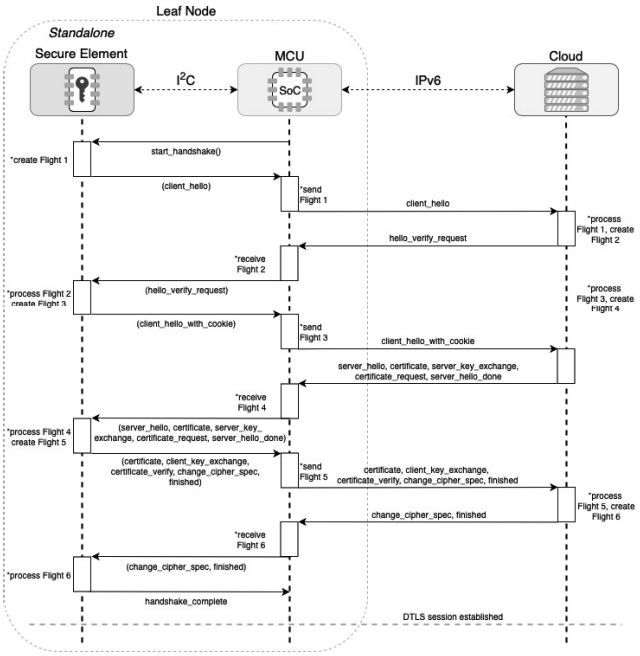


Fig. 7 DTLS handshake using a standalone secure element

For example, the cipher suite TLS_ECDHE_ECDSA_WITH_AES128_CCM_8 [16] defines that ECDH ephemeral key exchange is used as well as ECDSA for signing and verification. Finally, en- / decryption is done using AES128 in the Counter CBC-MAC modus. Another important aspect is the number of parallel sessions that need to be established and maintained. There are secure elements, which support up to four parallel sessions.

IV. Evaluation of Secure Elements

As part of a project at the Institute of Embedded System a detailed evaluation of four secure elements from four different vendors has been performed. The evaluation features detailed power measurements for two self-designed test cases with all four secure elements. The measurements serve to uncover the properties and familiarize with the handling of the individual secure elements. As a final report of the evaluation a paper has been planned. Unfortunately, one vendor is currently preventing the publication based on the NDA that protects the information about his secure element gained during the evaluation. Therefore, the measurement results of the evaluation are discussed only in general in this paper. As soon as the legal issues will be resolved, a paper will follow, discussing the measurement results in detail.

A. Performed measurements

The evaluation involved four secure elements, two from the category of the *supporting* secure elements and two *standalone* secure elements. The *supporting* secure elements are the ATECC608A [17] from Microchip and the A71CH [18] from NXP. The *standalone* secure elements are the TO136 [19] from Trusted Objects and the OPTIGA™ Trust X [20] from Infineon. To provide a comparable environment for the measurements a

printed circuit board (PCB) with all four secure elements on it has been designed, shown in Fig. 8.

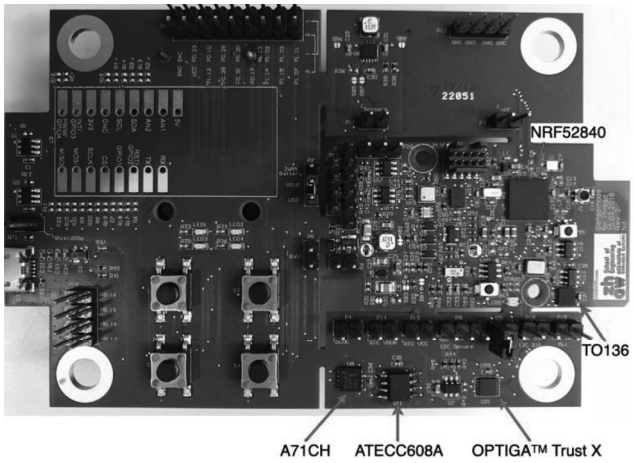


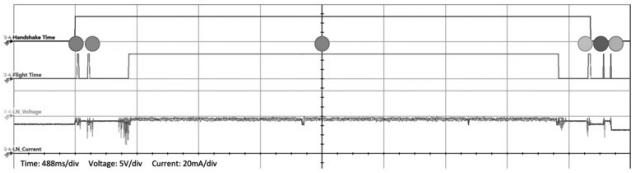
Fig. 8 Leaf node with four different secure elements

As host MCU, a NRF52840 [21] from Nordic Semiconductors is used. For the evaluation, two test cases have been defined. The first one focuses on the execution of five standard cryptographic operations allowing for a comparison between the different secure elements. The following five operations are executed:

- Generation of a random number
- Generation of an elliptic curve key pair
- Calculation of a SHA256 hash
- Calculation of an ECDSA signature
- Verification of the signature

The second test case focuses on a real-world application case. In this test case, the leaf node executes a DTLS handshake with a server and makes a CoAPs [22] GET request to the server. The secure elements support the MCU in the execution of the handshake, as described in the chapter III.C.

As a reference measurement, the test case has been executed with the cryptographic software mbedTLS. Fig. 9 shows the result of this reference measurement. The DTLS handshake executed only in software needs about 4 s and 67'209 nWh. The measurement results of the test case with the support of secure elements show that both execution time and energy consumption can be significantly reduced.



	Operation	Ex. Time [ms]	Avg. Current [mA]	Energy [nWh]
●	Get Flight 1	15.6	15.46	232
●	Handle Flight 2 Get Flight 3	15.3	15.41	228
●	Handle Flight 4 Get Flight 5	3'399	18.2	56'721
●	Handle Flight 6	21.6	15.48	314
	Complete Handshake	4'075	17.99	67'209
●	Send encrypted data	5.5	15.74	84
●	Receive encrypted data	5.8	15.44	87

Fig. 9 Power measurement of a DTLS handshake executed with mbedTLS (no secure element)

B. Measurement results

The measurements results show that there are significant differences between the secure elements. The comparisons between the secure elements in the first test case have revealed that there are secure elements, which execute an ECDSA operation in about 60 ms while others need up to 1 s. Even though, the fast secure elements tend to have higher current consumption, due to the fast execution the overall energy consumption is significantly smaller than with the slow secure elements. The same can be applied for the real-world application test case, where the fast secure elements were able to not only halve the execution time of the handshake but also reduce the required energy by a factor 5 compared to the reference measurement with mbedTLS (using no secure element). As a result, using a secure element the DTLS handshake could be executed in about 2 s and only required about 15'000 nWh. Furthermore, the measurements show that secure elements have reasonably small average currents, in a range from 3 – 10 mA. Also, the sleep currents are low, typically about 50 µA or even 0 µA for one specific secure element.

V. KEY FINDINGS

Working with secure elements has shown their great potential for resource constrained IoT devices. With the dedicated hardware acceleration, cryptographic operations can be executed faster and may save the leaf node energy and therefore extend the battery lifetime. The tamper proof memory combined with the access authentication provides a trustworthy storage opportunity where sensitive data can be stored. This allows the leaf node to become an authentic and trusted entity of a network. The performed evaluation has revealed major differences regarding execution time and energy consumption between the four selected secure elements. This variety offers the opportunity to use secure elements for many different

applications. However, the evaluation has also shown that a lot of effort has to be invested to integrate secure elements into an application. This required effort poses a major obstacle to use secure elements on IoT devices. Furthermore, the requirement for an NDA before any detailed information is exchanged between the costumer and the vendor raises the bar to use secure elements. However, first impulses into a more open information exchange have already been observed.

VI. Conclusion

This paper introduces the concept of secure elements and provides a general description of their features on a basic level. Furthermore, the paper defines two categories for secure elements. The presented usage examples illustrate how secure elements can be used in real-world applications. Although, for legal reasons the made evaluation of four selected secure elements is not yet ready to be published, this paper presents first key learnings from the evaluation. This paper may be seen as the introduction to a following paper describing the performed evaluation and the selected secure elements in detail.

VII. Appendix

The following table lists a set of common secure elements features and gives a short description.

Table 1 Set of common secure element features

Feature	***Description***
RNG	Certified random number generator, certifications include AIS-31 [23], Common Criteria (CC) [24] or cryptographic algorithm validation program (CAVP) [25]
ECDH	Elliptic curve Diffie-Hellmann key exchange
ECDSA	Elliptic curve digital signature algorithm
HMAC	Key-Hash Message Authentication Code [26]
Key Gen	ECC key pair generated within the secure element
Elliptic Curve	Curves include NIST p-256, p-384
SHA	Secure hash algorithm [27]
AES	Hardware accelerated advanced encryption standard
(D)TLS	(Datagram) Transport layer security, supported cipher suites are: TLS_ECDH/E_ECDSA_WITH_AES128_CBC_SHA256 [28] TLS_ECDHE_ECDSA_WITH_AES128_CCM_8
Memory	Tamper proof memory, resistant against physical attacks
Cert. handling	Handling and storage of x509 certificates
I^2C	Communication interface with speeds up to 1 Mbit/s
I^2C encrypt.	Possibility to encrypt the I^2C interface either with symmetric or asymmetric secrets
Avg. Current	The average current consumption lies between 3 – 10 mA.
Sleep Current	Sleep currents lie between 0 – 150 µA

References

[1] Thread Group Inc., "About Thread," [Online]. Available: https://www.threadgroup.org/What-is-Thread. [Accessed Dec 2018].

[2] IEEE Standart Association, "Low-Rate Wireless Network (802.15.4)," [Online]. Available: https://standards.ieee.org/standard/802_15_4-2015-Cor1-2018.html. [Accessed Dec 2018].

[3] CardLogix Corporation, "Smard Card Basics," [Online]. Available: http://www.smartcardbasics.com/smart-card-standards.html.

[4] J. Daemen and V. Rijmen, "AES Standard," National Institute of Standards and Technology (NIST), [Online]. Available: https://doi.org/10.6028/NIST.FIPS.197. [Accessed Jan 2019].

[5] E. Rescorla and RTFM, "ECDH," Internet Engineering Task Force (IETF), [Online]. Available: https://tools.ietf.org/html/rfc2631. [Accessed Jan 2019].

[6] D. McGrew, K. Igoe, M. Salter, Cisco_Systems and National_Security_Agency, "ECC," Internet Engineering Task Force (IETF), [Online]. Available: https://tools.ietf.org/html/rfc6090. [Accessed Dec 2018].

[7] Advanced RISC Machines (ARM), "MbedTLS," [Online]. Available: https://tls.mbed.org. [Accessed Dec 2018].

[8] F. Hao and N. University, "JPAKE," Internet Engineering Task Force (IETF), [Online]. Available: https://tools.ietf.org/html/rfc8236. [Accessed Dec 2018].

[9] E. Rescorla, RTFM, N. Modadugu and Google, "DTLS," Internet Engineering Task Force (IETF), [Online]. Available: https://tools.ietf.org/html/rfc6347. [Accessed Dec 2018].

[10] T. Pornin, "ECDSA," Internet Engineering Task Force (IETF), [Online]. Available: https://tools.ietf.org/html/rfc6979. [Accessed Jan 2019].

[11] S. Skorobogatov, "Physical Attacks and Tamper Resistance," [Online]. Available: https://pdfs.semanticscholar.org/d043/16ba2a6895febe541d566f2a141168d6d7d5.pdf. [Accessed Jan 2019].

[12] D. Cooper, NIST, S. Santesson, Microsoft, S. Farrell, Trinity_College_Dublin, S. Boeyen, Entrust, R. Housley, Vigil_Security and W. Polk, "x509 Certificates," Internet Engineering Task Force (IETF), [Online]. Available: https://tools.ietf.org/html/rfc5280. [Accessed Dec 2018].

[13] M. Nystrom, B. Kaliski and RSA_Security, "CSR," Internet Engineering Task Force (IETF), [Online]. Available: https://tools.ietf.org/html/rfc2986. [Accessed Dec 2018].

[14] Microchip Technology, "Youtube," [Online]. Available: https://www.youtube.com/watch?v=cbOuo-wL2Ms&t=563s. [Accessed Dec 2018].

[15] Internet Engineering Task Force (IETF), "About IETF," [Online]. Available: https://www.ietf.org/about/. [Accessed Jan 2019].

[16] D. McGrew, Cisco_Systems, D. Bailey, Ruhr-University-Bochum, M. Campagna, R. Dugal and Certicom_Corp., "Cipher CCM," Internet Engineering Task Force (IETF), [Online]. Available: https://tools.ietf.org/html/rfc7251. [Accessed Dec 2018].

[17] Microchip, "ATECC608A," [Online]. Available: https://www.microchip.com/wwwproducts/en/ATECC608A. [Accessed Dec 2018].

[18] NXP Semiconductor, "A71CH," [Online]. Available: https://www.nxp.com/products/identification-and-security/authentication/plug-and-trust-the-fast-easy-way-to-deploy-secure-iot-connections:A71CH. [Accessed Dec 2018].

[19] Trusted Objects, "TO136," [Online]. Available: https://www.trusted-objects.com/webtest/index.php?page=en-TO136-secure-element. [Accessed Dec 2018].

[20] Infineon, "OPTIGA Trust X," [Online]. Available: https://www.infineon.com/cms/de/product/security-smart-card-solutions/optiga-embedded-security-solutions/optiga-trust/optiga-trust-x-sls-32aia/. [Accessed Dec 2018].

[21] Nordic Semiconductor, "nRF52840," [Online]. Available: https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF52840. [Accessed Dec 2018].

[22] Z. Shelby, ARM, K. Hartke, C. Bormann and U. B. TZI, "CoAP," Internet Engineering Task Force (IETF), [Online]. Available: https://www.rfc-editor.org/rfc/rfc7252.txt. [Accessed Dec 2018].

[23] German Federal Office for Information Security, "AIS-31," [Online]. Available: https://www.bsi.bund.de/DE/Themen/ZertifizierungundAnerkennung/Produktzertifizierung/ZertifizierungnachCC/Anwendungshinweiseund Interpretationen/AIS-Liste.html. [Accessed Dec 2018].

[24] Common Criteria for Information Technology Security Evaluation (CC), "CommonCriteria," [Online]. Available: https://www.commoncriteriaportal.org. [Accessed Jan 2019].

[25] National Institute of Standards and Technology (NIST), "CAVP," [Online]. Available: https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program. [Accessed Jan 2019].

[26] H. Krawczyk, IBM, M. Bellare, UCSD and R. Canetti, "HMAC," Internet Engineering Task Force (IETF), [Online]. Available: https://tools.ietf.org/html/rfc2104. [Accessed Dec 2018].

[27] D. Eastlake, Motorola_Labs, T. Hansen and AT&T_Labs, "SHA256," Internet Engineering Task Force (IETF), [Online]. Available: https://tools.ietf.org/html/rfc4634. [Accessed Dec 2018].

[28] E. Rescorla and RTFM, "Cipher CBC," Internet Engineering Task Force (IETF), [Online]. Available: https://www.ietf.org/rfc/rfc5289.txt. [Accessed Dec 2018].