





Article

Performance Analysis of Secure Elements for IoT

Mario Nosedá ^{1,†} , Lea Zimmerli ^{1,†} , Tobias Schläpfer ^{2,†}  and Andreas Rüst ^{1,*,†} 

¹ Institute of Embedded Systems, Zurich University of Applied Sciences, 8401 Winterthur, Switzerland; mario.nosedá@zhaw.ch (M.N.); lea.zimmerli@zhaw.ch (L.Z.)

² NatWest Group, NatWest Services (Switzerland) Ltd., 8045 Zuerich, Switzerland; tobias.schlaepfer@iost-company.ch

* Correspondence: andreas.ruest@zhaw.ch; Tel.: +41-(0)-58-934-77-01

† These authors contributed equally to this work.

Abstract: New protocol stacks provide wireless IPv6 connectivity down to low power embedded IoT devices. From a security point of view, this leads to high exposure of such IoT devices. Consequently, even though they are highly resource-constrained, these IoT devices need to fulfil similar security requirements as conventional computers. The challenge is to leverage well-known cybersecurity techniques for such devices without dramatically increasing power consumption (and therefore reducing battery lifetime) or the cost regarding memory sizes and required processor performance. Various semiconductor vendors have introduced dedicated hardware devices, so-called secure elements that address these cryptographic challenges. Secure elements provide tamper-resistant memory and hardware-accelerated cryptographic computation support. Moreover, they can be used for mutual authentication with peers, ensuring data integrity and confidentiality, and various other security-related use cases. Nevertheless, publicly available performance figures on energy consumption and execution times are scarce. This paper introduces the concept of secure elements and provides a measurement setup for selected individual cryptographic primitives and a Datagram Transport Layer Security (DTLS) handshake over secure Constrained Application Protocol (CoAPs) in a realistic use case. Consequently, the paper presents quantitative results for the performance of five secure elements. Based on these results, we discuss the characteristics of the individual secure elements and supply developers with the information needed to select a suitable secure element for a specific application.

Keywords: cybersecurity; dtls; secure elements; embedded systems; resource-constrained devices; coaps; IoT



Citation: Nosedá, M.; Zimmerli, L.; Schläpfer, T.; Rüst, A. Performance Analysis of Secure Elements for IoT. *IoT* **2022**, *3*, 1–28. <https://doi.org/10.3390/iot3010001>

Academic Editor: Hyun-Ho Choi

Received: 21 October 2021

Accepted: 7 December 2021

Published: 21 December 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

In a typical, straightforward IoT application, a sensor node is sending data to an application server. Nowadays, in many cases, such a node connects to a proprietary, local network with a gateway providing connectivity to the internet by protocol translation. Due to this non-transparent access from the internet, there is little need to implement performance-hungry security measures on the resource-constrained sensor node itself. In contrast, the gateway with its lower constraints on resources can, to a certain extent, isolate the sensor node through security measures, such as firewalls and standard cryptographic algorithms.

Interestingly, IoT networks like Thread [1] introduce native Internet Protocol version 6 (IPv6) connectivity down to the low-power sensor nodes. While this transparency offers many new opportunities to access these resource-constrained devices, in turn, it now requires them to provide a similar level of security as conventional computers. However, cryptographic algorithms demand complex calculations with long execution times when performed on a microcontroller unit (MCU) with little processing power. These algorithms require a high amount of energy and thus reduce battery lifetime. Moreover, conventional security algorithms (e.g., Rivest-Shamir-Adleman RSA) exacerbate this effect as they were

not designed to run on resource-constrained devices. Thus, various semiconductor vendors have developed dedicated hardware for MCUs to support the execution of cryptographic algorithms. These so-called secure elements improve the overall security of an IoT device and reduce the energy consumed by cryptographic computations. This paper introduces the concept of secure elements and shows their opportunities as well as their challenges. Specifically, we describe five secure elements in detail with appropriate energy and execution time measurements. The provided data supports the selection of a suitable secure element for individual IoT applications.

This paper is structured accordingly: Section 2 highlights the motivation behind this project. Section 3 contains the technical background (secure elements, Thread protocol, CoAP) needed to understand the rest of the paper. Section 4 lists the five evaluated secure elements and all the required soft- and hardware used for the evaluation. The measurements and subsequent discussion are located in Sections 5 and 6, respectively. Section 7 describes the key findings gathered during this project, Section 8 draws attention to related work and similar studies, and Section 9 draws appropriate conclusions.

2. Motivation

Although secure elements have been available for a few years, there are still few to no quantitative comparisons between chips from different manufacturers. Arguably, this is primarily because manufacturers are very reluctant to provide information about the hard- and software of their secure elements. Until recently, one usually had to sign a non-disclosure agreement (NDA) before receiving the documentation and vendor-specific software development kit (SDK) required to use the secure element. Although more and more manufacturers are now moving away from these prohibitive practices, there is still a lack of quantitative comparisons that help embedded security engineers decide on which secure element best fits the task at hand. Embedded security is already lagging behind the current emergence of IoT, which can be seen, among other things, in the lack of large and active developer communities. Understandably, a lack of information does not contribute positively to this situation.

Therefore, our primary motivation for this project is to make such information publicly available and contribute to the security of IoT systems.

3. Technical Background

3.1. Secure Elements

A secure element is an integrated circuit (IC) for executing cryptographic algorithms (mostly) in hardware. Such algorithms typically include the advanced encryption standard (AES), elliptic curve cryptography (ECC), the elliptic curve digital signature algorithm (ECDSA), secure hashing algorithms (SHA), message authentication codes (MAC), and more depending on the feature set of the secure element. They are designed to execute these computationally intensive algorithms fast and with minimal energy. Furthermore, they need to be connected to an MCU over a serial interface and function as a cryptographic coprocessor.

In contrast, more and more MCUs with built-in security peripherals also implement such algorithms in hardware. They have an advantage concerning execution time and energy consumption compared to secure elements as the security peripherals are connected to the internal data bus and not to a comparatively slow external interface. For such MCUs, it is common practice to encrypt sensitive key material with a master key before storing it in the flash memory. However, even though the sensitive data is encrypted, such MCUs usually provide little to no tamper detection as well as no tamper-resistant memory.

Conversely, most secure elements provide tamper-resistant memory, multiple tamper detection mechanisms like active shielded circuits, voltage and temperature monitoring, and inputs for user-defined tamper sensors. These features are a clear added benefit over using MCUs with built-in security peripherals. Additionally, they provide sophisticated countermeasures against side-channel attacks and have some level of Common Criteria certification.

A (D)TLS handshake is a likely use case for secure elements, which is usually implemented in two distinct ways. On the one hand, some secure elements support an MCU in the execution of cryptographically complex computations. However, the complete (D)TLS stack must still be present in the MCU to handle the various messages correctly. Conversely, other secure elements can handle the entire (D)TLS session independently. In this case, the MCU does not have to process the (D)TLS messages but only forwards them in both directions. The execution of all security-related functions within the secure element naturally offers higher security. Nevertheless, these secure elements can also support a (D)TLS stack running on the MCU, making them very versatile. Further theoretical foundations concerning secure elements can be found in our previously released papers: chapter II “Secure Elements” in “Security on IoT Devices with Secure Elements” [2] and “Securing the IoT: Introducing an Evaluation Platform for Secure Elements” [3].

3.2. Constrained Application Protocol

The Constrained Application Protocol (CoAP) is specified by RFC7252 [4] and is a document transfer protocol like Hypertext Transfer Protocol (HTTP). However, as the name implies, it is specifically designed for constrained devices. Bit fields and mappings are used to keep the packets as small as possible, and User Datagram Protocol (UDP) is used instead of Transmission Control Protocol (TCP). CoAP is based on a simple client/server model and follows the RESTful paradigm. The secure Constrained Application Protocol (CoAPs) utilizes DTLS and guarantees confidentiality, integrity and authenticity of the CoAP packets.

3.3. Thread Protocol

Thread is a wireless network protocol for embedded IoT devices building on the Institute of Electrical and Electronics Engineers (IEEE) 802.15.4 [5] standard. It provides a mesh network for resource-constrained devices where each device has IPv6 connectivity. The devices within the Thread network have access to the internet via border routers. This new connectivity provides new opportunities such as service discovery for each device or the possibility to establish a real end-to-end encrypted channel between an end device and an application server. Figure 1 shows a setup of such a Thread network and a corresponding application server residing outside of that network.

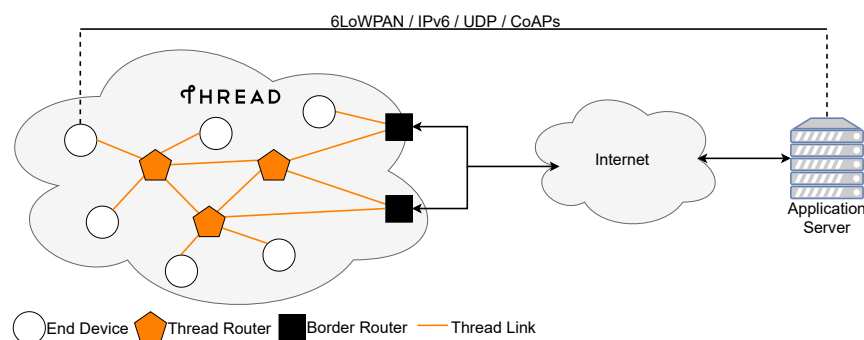


Figure 1. Typical Thread network with access to an application server over the internet.

Figure 2 illustrates the Thread stack which consists of IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN), IP-routing and UDP. These layers, in turn, facilitate the use of application protocols like CoAP.

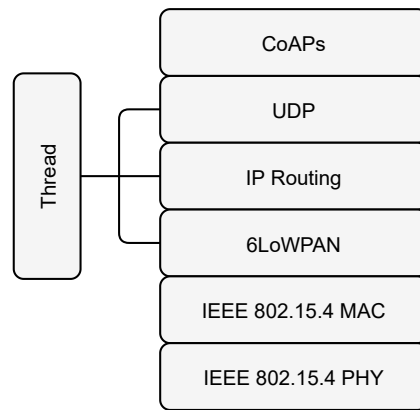


Figure 2. Stack of the Thread protocol.

4. Materials and Methods

4.1. Test Cases

We define two specific test cases for our performance evaluation in terms of execution time and energy consumption. While the first test case focuses on cryptographic primitives, the second one features a practical application. Specifically, a node and a server establish a secure session by executing a DTLS handshake. To establish a baseline *reference* without MCU-external hardware acceleration, we measure both test cases with the cryptographic functions running in software on the MCU using MbedTLS [6]. In contrast, subsequent measurements make use of the evaluated secure elements. The results allow a direct comparison to the reference as well as between devices.

4.1.1. Cryptographic Primitives

This test case assesses the performance of the individual secure elements concerning the following cryptographic primitives:

- Generate a random number (32 bytes)
- Generate an ECC key pair (secp256r1 [7])
- Calculate the SHA-256 hash of the random number
- Sign the hash with ECDSA (using key pair from before)
- Verify the signature (using key pair and hash from before)

secp256r1 is one of the most common elliptic curves and thus has been selected for this performance evaluation. Furthermore, curves with larger bit-fields were not evaluated as they are rarely seen in IoT.

Moreover, the generation of the random number with the reference implementation requires the following remarks:

MbedTLS contains a block-cipher counter-mode based deterministic random bit generator (CTR_DRBG) specified in NIST SP800-90A [8]. This cryptographically-secure pseudorandom number generator (CSPRNG) requires a strong external entropy source for periodical reseeding. Thus, the reference implementation does not purely rely on software, as in this case, the true random number generator (TRNG) in the MCU itself is used as an entropy source. We will discuss this further in Section 6.1.

4.1.2. DTLS Handshake

In this test case, a DTLS handshake with a server demonstrates the impact of the secure elements as external crypto accelerators on execution time and energy consumption. We have chosen TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 as the cipher suite for the DTLS handshake as this suite is available in all the evaluated secure elements. Therefore, this choice facilitates a fair comparison even though there are arguably better cipher suites from a security standpoint. Lastly, CoAPs is used as the application protocol for getting some dummy data from the server.

4.2. Test Setup

In order to keep the complexity of the network as low as possible, the Thread network only consists of a border router and a secure node. Furthermore, the CoAPs server is running directly on the border router itself to exclude as many unknowns as possible, which could arbitrarily affect the handshake in any way. Figure 3 illustrates this test setup.

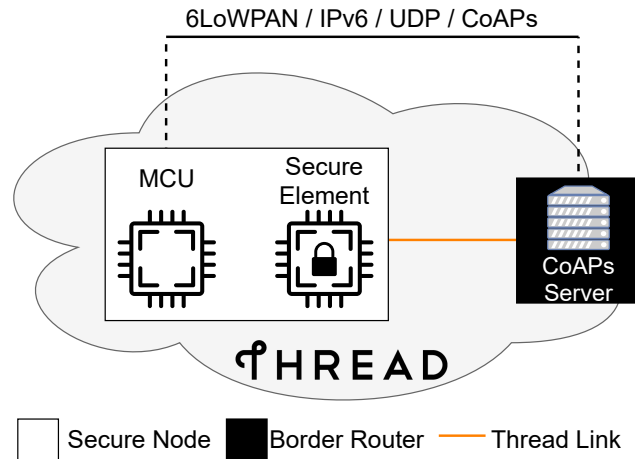


Figure 3. Test setup with reduced complexity to ensure reproducibility of results.

Figure 4 depicts the secure node, which consists of a Nordic nRF52840 development kit (DK) [9] and the secure element shield, which is a self-designed PCB with the form factor of an Arduino shield on which all evaluated secure elements are present. Additionally, a Raspberry Pi 3 Model B+ [10] and an nRF52840 dongle [11] (used as a radio co-processor for IEEE 802.15.4) serve as the required border router and CoAPs server.

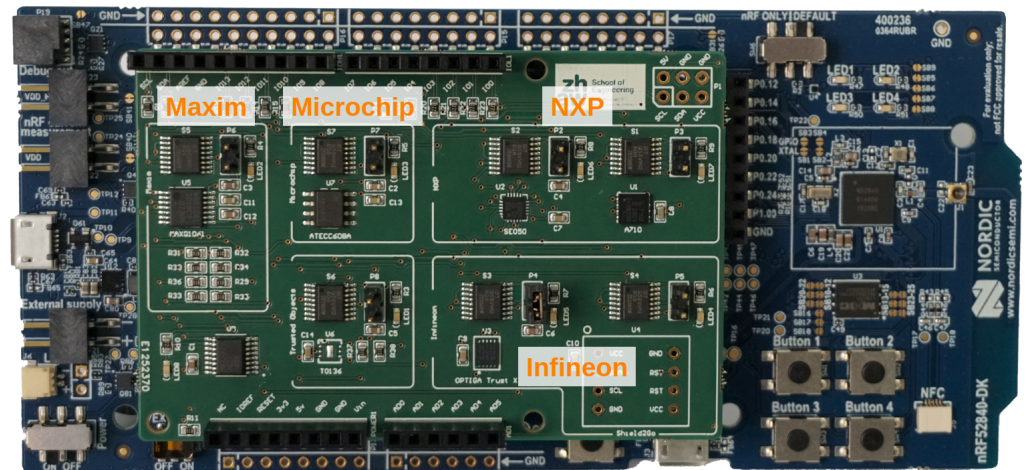


Figure 4. Secure node consisting of nRF52840 DK and secure element shield.

4.3. Evaluated Secure Elements

This paper evaluates the following secure elements:

- Infineon OPTIGA™ Trust M [12,13]
- Infineon OPTIGA™ Trust X [14,15]
- Maxim MAXQ1061 [16]
- Microchip ATECC608B [17,18]
- NXP SE050 [19,20]

These five secure elements have been selected as they list similar use cases and target applications in their data sheets. Furthermore, they all support the specified algorithms

and the corresponding elliptic curve required for performing the cryptographic primitives, which allows the direct comparison of the results.

4.3.1. Supported Cryptographic Primitives

Although all these secure elements provide cryptographic primitives like symmetric encryption with AES, various ECC algorithms (ECDH, ECDHE, ECDSA, ...), secure hashing algorithms (e.g., SHA-256) and message authentication (e.g., HMAC-SHA-256), essential differences exist. These include AES modes, hash and block cipher sizes, and supported elliptic curves. Thus, it is essential to check if the selected secure element provides the required functionality prior to including it in a new design.

Furthermore, the OPTIGA™ Trust X contains a complete (D)TLS stack and can handle the entire handshake on its own by creating and parsing the messages without the help of the MCU. Thus, the MCU does not have to process the (D)TLS messages and only transmits them between the secure element and the (D)TLS peer. Conversely, the other secure elements support the MCU by accelerating the computationally expensive cryptographic algorithms while still requiring a (D)TLS stack to be present in the MCU for creating and parsing the actual (D)TLS packets (i.e., ClientHello, ServerHello, ...).

4.3.2. Memory Structure

The secure elements differ not only in the amount of memory available but also in their structure. While the ATECC608B, OPTIGA™ Trust M, and OPTIGA™ Trust X feature pre-partitioned slots with explicit restrictions on what type of data can be stored within them, the MAXQ1061 and the SE050 allow the users to define the type and size of the storage objects individually.

4.3.3. Available Interfaces

All evaluated secure elements feature an I2C interface. The OPTIGA™ Trust M, OPTIGA™ Trust X, and the ATECC608B allow an I2C clock speed of up to 1 MHz. Moreover, the latter also allows the use of a significantly slower Single-Wire Interface (SWI). The SE050 does not contain any other serial interface than I2C, but it features the fastest I2C clock speed at 3.4 MHz. In contrast, the MAXQ1061 has the slowest maximum I2C clock speed at 400 kHz. However, it is the only secure element containing an SPI peripheral, and with a maximum clock speed of 4 MHz, it provides the fastest serial interface of them all.

We strongly recommend encrypting the communication between the host MCU and the secure element. Even though the symmetric keys used for such secure channels reside somewhere in the memory of the MCU and are at risk of being extracted by an attacker, it significantly increases the level of security. This increase is because the attacker requires drastically more time and effort as merely probing the serial interface is not enough anymore. Therefore, it mostly mitigates the risk of easy eavesdropping but also more dangerous attacks like tampering with the random numbers, which would severely impede the reliability of certain cryptographic algorithms.

4.3.4. Energy Consumption and Sleep Modes

Another critical aspect of secure elements is their energy consumption during computation, idling, and sleeping. Their active supply current, sleep current, wake-up delay, as well as other characteristics, may differ widely (up to orders of magnitude).

4.3.5. Provisioning

All manufacturers provide a provisioning service that can inject unique credentials into the secure elements at secure manufacturing facilities. However, the ATECC608B is the only secure element that must be provisioned and locked before any of the functionalities (e.g., getting random numbers from the TRNG) can be used. On the one hand, this slightly complicates the development stage, but it also increases the overall security as the chip cannot be used without being locked first.

4.3.6. Compliance

Compliance with known certifications is another important aspect of secure elements, as this might be a hard requirement depending on the target application. The Common Criteria (CC) [21] provide a well-known standard within the field of Information Security. The hardware and firmware of the SE050 and the hardware of the OPTIGA™ Trust M and OPTIGA™ Trust X are CC EAL6+ certified. Although compliant development and design practices for CC EAL4+ have been used on the MAXQ1061, which would suggest compliance, Maxim has not performed any actual evaluation and thus cannot prove the compliance by document (W. Pflästerer, personal communication, 6 June 2021). The ATECC608B is the only evaluated secure element that does not state any compliance to CC certifications. Moreover, as true randomness is fundamental in cryptography, the compliance of the TRNGs with further certification is another key characteristic. While the MAXQ1061 does not state a compliance with any certification, the TRNGs of the other secure elements are either compliant with NIST SP 800-90A/B/C [8,22,23] or AIS-31 [24].

4.3.7. Resources for Development

Lastly, the secure elements also differ in how much resources are available. The OPTIGA™ Trust M and OPTIGA™ Trust X provide a lot of publicly available documentation, examples, and ready-to-use driver implementations (e.g., Nordic nRF5 SDK, Zephyr, ...). In contrast, the full data sheet and SDK of the MAXQ1061 are still under NDA at the time of writing this paper, with the remaining secure elements landing somewhere between these two.

4.4. Selected Communication and Sleep Modes

In order to make the measurements uniform, we chose an I2C clock speed of 400 kHz, as this is the fastest possible communication that all secure elements support. The evaluated secure elements require the MCU to poll their busy state over I2C. Moreover, the MCU is not put into sleep mode during the secure elements' execution of the cryptographic primitives.

4.5. Software

4.5.1. Secure Node Firmware

Zephyr [25] is an open-source real-time operating system for resource-constrained devices (mainly microcontrollers) with a focus on connectivity. It integrates various protocol stacks, and supports a wide range of architectures and processor families, allowing you to compile the same code for a wide range of MCUs. This abstraction from the hardware and communication interfaces makes this RTOS an excellent choice for IoT projects. The Zephyr project is part of the Linux Foundation, and this is visible in its structure. Among other things, it has a device driver model with clear parallels to conventional Linux. This model allows us to create a so-called external module for every secure element, which contributes to an excellent encapsulation of SDKs provided by the chip vendors. Furthermore, it includes an open-source implementation of the Thread protocol called OpenThread [26], which uses MbedTLS as a cryptographic software library. Although the OPTIGA™ Trust X can handle the complete DTLS handshake on its own, in our setup, the DTLS handshake is handled by MbedTLS. Thus, all secure elements are used as external cryptoprocessors for all security and performance-critical computations, which results in a better comparison between secure elements.

MbedTLS (version 2.7.0 or later) allows alternative cryptographic implementations [27], so-called hooks, by packing the definitions of various functions into include guards as displayed by Listing 1.

Listing 1. MbedTLS ECDSA verify function definition with include guards.

```

1  #if !defined(MBEDTLS_ECDSA_VERIFY_ALT)
2
3  /*
4   * Verify ECDSA signature of hashed message
5   */
6  int mbedtls_ecdsa_verify( mbedtls_ecp_group *grp, const unsigned char *buf,
7                          size_t blen, const mbedtls_ecp_point *Q,
8                          const mbedtls_mpi *r, const mbedtls_mpi *s)
9  {
10     ...
11 }
12 #endif /* !MBEDTLS_ECDSA_VERIFY_ALT */

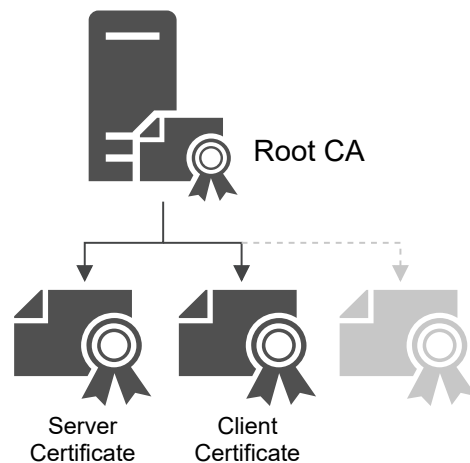
```

By defining the MBEDTLS_ECDSA_VERIFY_ALT option in the MbedTLS configuration, only the function declaration within the corresponding header file remains, and the user can now supply a new definition of the function, which makes use of the secure element. The following configuration options have been defined to hook the desired computations with the evaluated secure elements:

- MBEDTLS_ENTROPY_HARDWARE_ALT
- MBEDTLS_ECDSA_SIGN_ALT
- MBEDTLS_ECDSA_VERIFY_ALT
- MBEDTLS_ECDH_COMPUTE_SHARED_ALT
- MBEDTLS_ECDH_GEN_PUBLIC_ALT

4.5.2. Certificate Authority (CA)

Figure 5 illustrates the certificate structure used for this project with all the required certificates for the DTLS handshake test case. Generally, we strongly advise making use of an intermediate CA. However, for the measurements in this project, only a root CA without any intermediate CAs has been created in order to keep it as simple as possible.

**Figure 5.** Certificate structure used in this project.

4.5.3. CoAPs Server

The CoAPs server running on the OpenThread border router is based on Eclipse Californium™ [28]. This CoAP framework is written in Java and compliant with RFC7252 [4]. By using the build automation tool Maven [29], Californium™ can be easily obtained and used in a project. The Californium (Cf) core implements the basic CoAP functionality, and the additional Scandium (Sc) submodule complements this with the necessary security

capabilities (like DTLS 1.2 and key storage) to set up the desired CoAPs server. The server is provided with a PKCS#12 [30] archive file in which all required certificates and the server's private key are stored.

4.6. Measurement Setup

Figure 6 shows how a Keysight N6705B power analyzer is used to determine execution times and power consumption. It supplies the nRF52840 MCU with 3.3 V which in turn supplies the connected secure element. It is used to measure the supply current of the MCU and the secure element. Furthermore, it measures the voltage of the supply and an auxiliary GPIO of the MCU (left side of the MCU directly connected to a voltage measurement channel of the power analyzer), which signals when the individual cryptographic primitives and the DTLS handshake are currently being executed. This setup does not just allow the acquisition of the execution time, but also the energy consumption of the MCU, secure element and both of them combined.

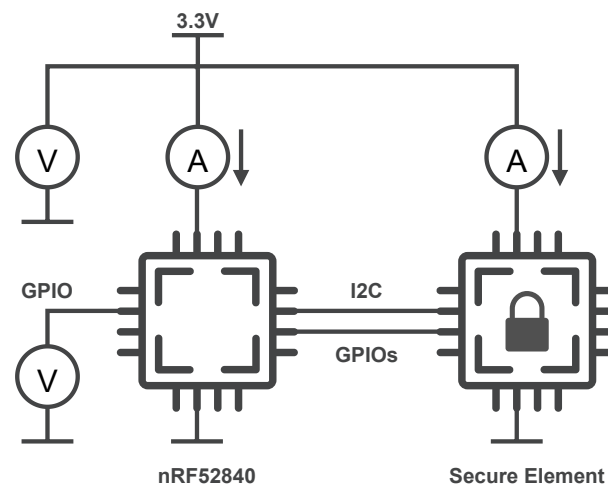


Figure 6. Conceptual measurement setup of the node for acquiring the execution time and energy consumption during the stated test cases.

In order to allow the separation of the MCU supply from the rest of the development kit, it needs to be set up as listed in Appendix A.1. Additionally, Appendix A.2 provides a more detailed explanation of the measurement setup.

To assess the variance, we acquire a large number of measurements by remotely controlling the power analyzer through a Python package. The same package processes the individual measurements and performs statistical analysis.

5. Results

5.1. Cryptographic Primitives

Figures 7–11 and the corresponding Tables 1–5 show the execution time and energy consumption of the node (secure element and MCU) during the cryptographic primitives in percentages relative to the reference implementation using only MbedTLS on the MCU. E.g., when generating an ECC key pair (secp256r1), the SE050 takes 26% of the time required by the reference implementation. I.e., substituting the software-based implementation in MbedTLS with the hardware-based alternative in the SE050 decreases the execution time of the key pair generation by 74%. The percentages have been calculated by dividing the median time and energy consumption of the node (secure element and MCU) by the reference (MCU by itself). The tables in Appendix C.1 contain the median values of the absolute and relative measurements used for creating these figures. Notably, the time and energy required for getting the 32 bytes of random data in the reference implementation originated from triggering the reseeding process of MbedTLS' CTR_DRBG, which accessed the TRNG in the MCU itself.

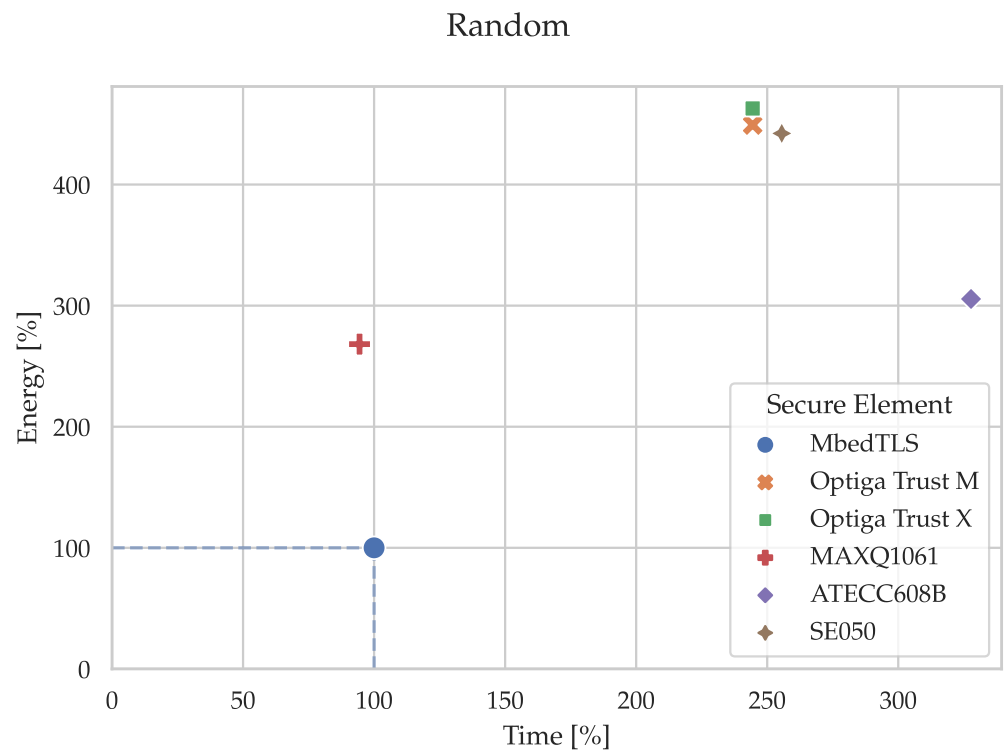


Figure 7. Time and energy measured for getting 32 bytes of random data relative to the reference implementation. (median of values, rounded to the nearest percent, n = 100).

Table 1. Table of values for Figure 7.

Secure Element	Time [%]	Energy [%]
MbedTLS	100	100
Optiga Trust M	244	449
Optiga Trust X	244	463
MAXQ1061	94	268
ATECC608B	328	306
SE050	256	442

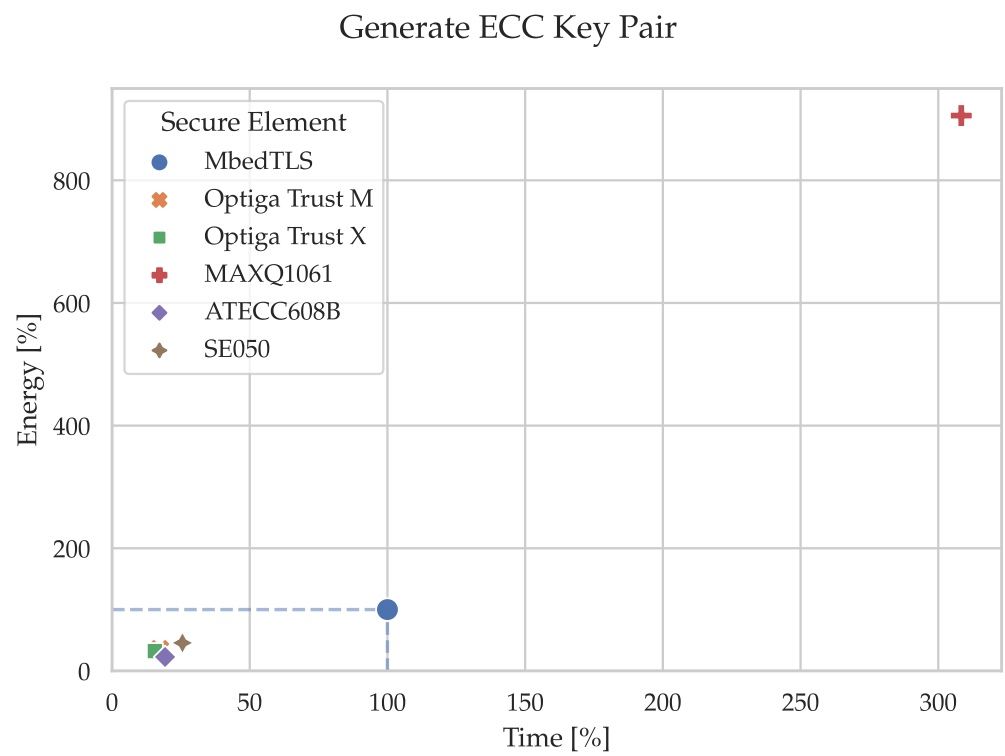


Figure 8. Time and energy measured for generating an ECC key pair relative to the reference implementation. (median of values, rounded to the nearest percent, $n = 100$).

Table 2. Table of values for Figure 8.

Secure Element	Time [%]	Energy [%]
MbedTLS	100	100
Optiga Trust M	17	35
Optiga Trust X	16	32
MAXQ1061	308	906
ATECC608B	19	23
SE050	26	46

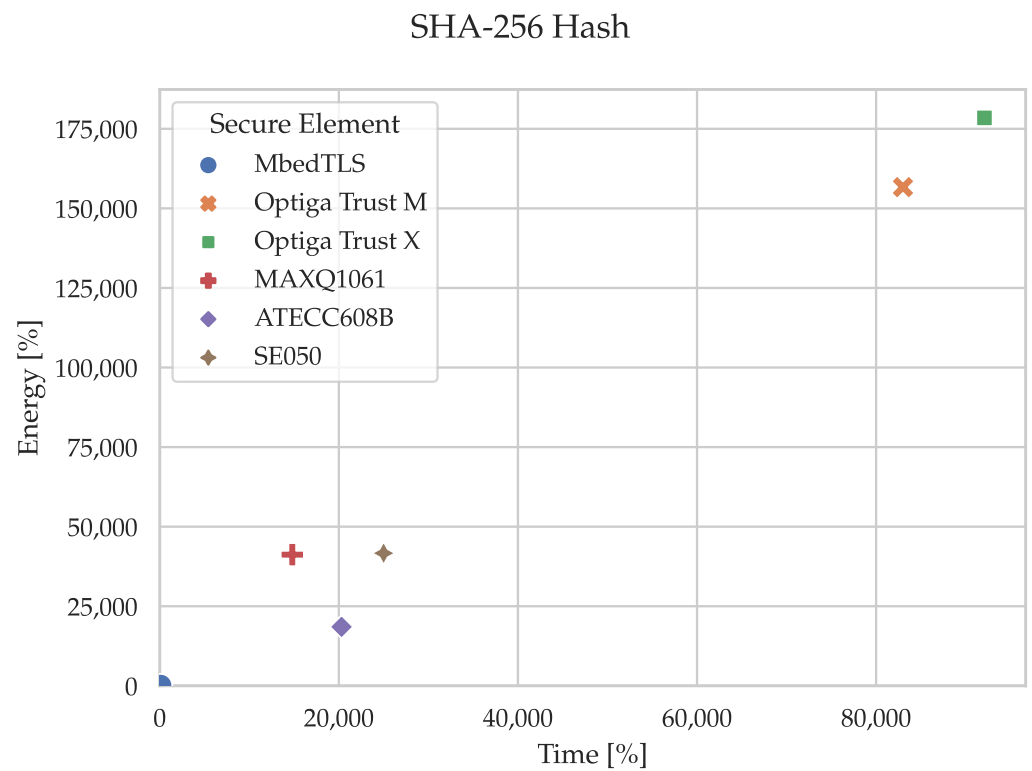


Figure 9. Time and energy measured for calculating the SHA-256 for 32 bytes of data relative to the reference implementation. (median of values, rounded to the nearest percent, $n = 100$).

Table 3. Table of values for Figure 9.

Secure Element	Time [%]	Energy [%]
MbedTLS	100	100
Optiga Trust M	83,000	156,670
Optiga Trust X	92,100	178,431
MAXQ1061	14,800	41,206
ATECC608B	20,300	18,530
SE050	25,000	41,663

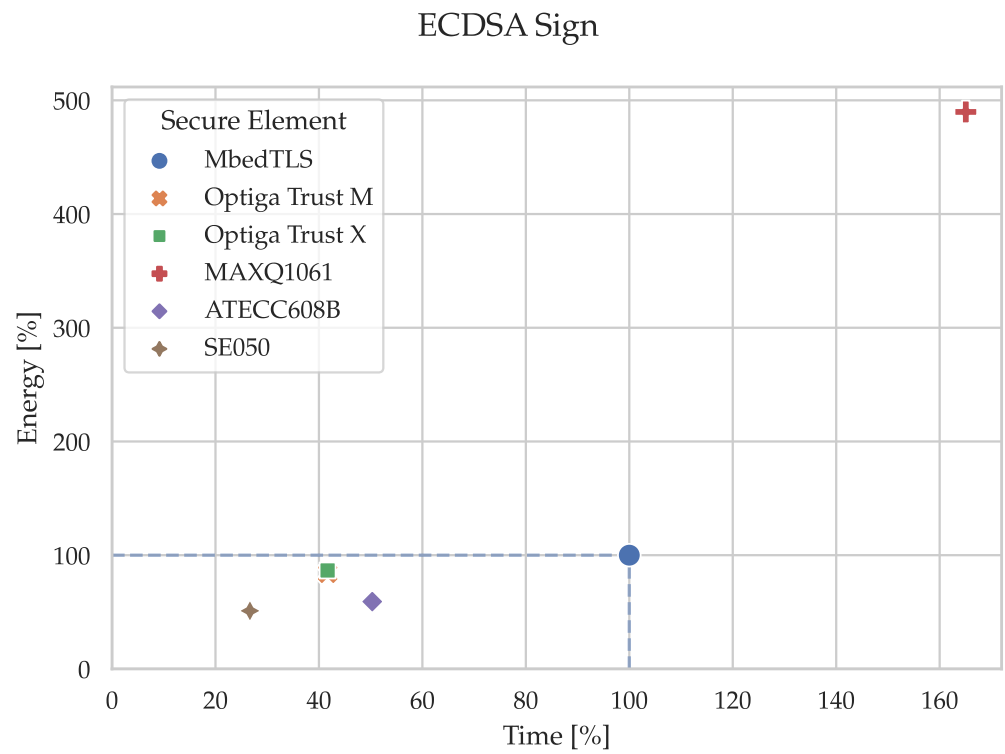


Figure 10. Time and energy measured for calculating an ECDSA signature (secp256r1) relative to the reference implementation. (median of values, rounded to the nearest percent, $n = 100$).

Table 4. Table of values for Figure 10.

Secure Element	Time [%]	Energy [%]
MbedTLS	100	100
Optiga Trust M	42	84
Optiga Trust X	42	87
MAXQ1061	165	490
ATECC608B	50	59
SE050	27	51

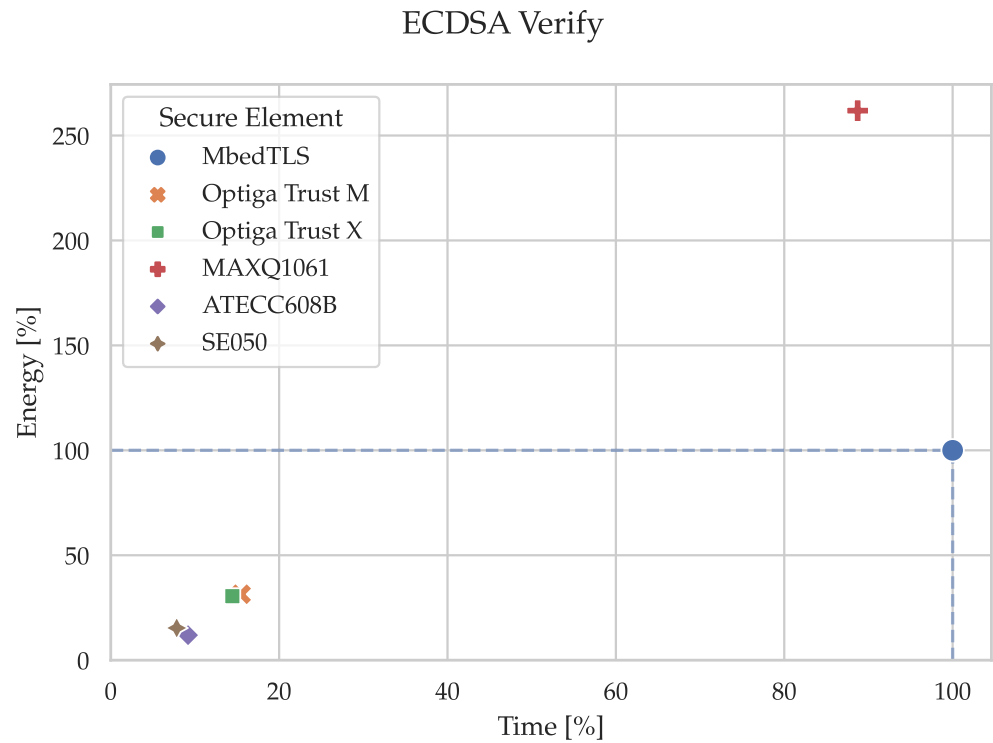


Figure 11. Time and energy measured for verifying an ECDSA signature (secp256r1) relative to the reference implementation. (median of values, rounded to the nearest percent, $n = 100$).

Table 5. Table of values for Figure 11.

Secure Element	Time [%]	Energy [%]
MbedTLS	100	100
Optiga Trust M	15	31
Optiga Trust X	14	31
MAXQ1061	89	262
ATECC608B	9	12
SE050	8	15

The relative standard deviation (RSD), also known as the coefficient of variation (CV), describes the amount of dispersion with regard to the mean. In contrast to the standard deviation, this allows us to compare data sets with widely different means, as is the case with the measurements of the cryptographic primitives. Equation (1) expresses how the RSD is calculated using the standard deviation σ and the mean μ .

$$RSD = \frac{\sigma}{\mu} \times 100 \quad (1)$$

The secure elements exhibited very consistent performance as 10.6% (energy consumption of the MAXQ1061 while generating random numbers) was the highest RSD of all the time and energy measurements within the cryptographic primitives. All the remaining measurements even had an $RSD < 5\%$. Even though the values are tightly grouped, they are not normally-distributed throughout. This is mainly because the secure elements are polled at defined intervals via I2C, and thus the end of a primitive is recorded on the next poll instead of the exact time when the primitive finished. Primitives with a very short execution time would require an extremely small polling interval in order to test for normality. For this reason, the figures and tables in this chapter are based on the median instead of the arithmetic mean. However, the remaining statistical values (min, max, mean,

standard deviation, and relative standard deviation) are available in the supplementary data (refer to the supplementary materials statement after Section 9).

5.2. DTLS Handshake

We noticed during the measurement process that handshake packets were sometimes lost and had to be retransmitted either by the client or the server, thus dramatically increasing the execution time. As an example, Figure 12 shows the DTLS handshake using the software-based MbedTLS reference. Since the focus is not on the reliability of OpenThread, we have removed all outliers that differ from the median by 20 percent or more. Figure 13 shows the corresponding adjusted measurements. We applied the same removal for all handshake measurements.

Identical to the benchmark for the cryptographic primitives, the DTLS handshake benchmark was repeated 100 times. Table 6 lists the remaining amount of valid measurements of the DTLS handshake after the removal of outliers caused by OpenThread.

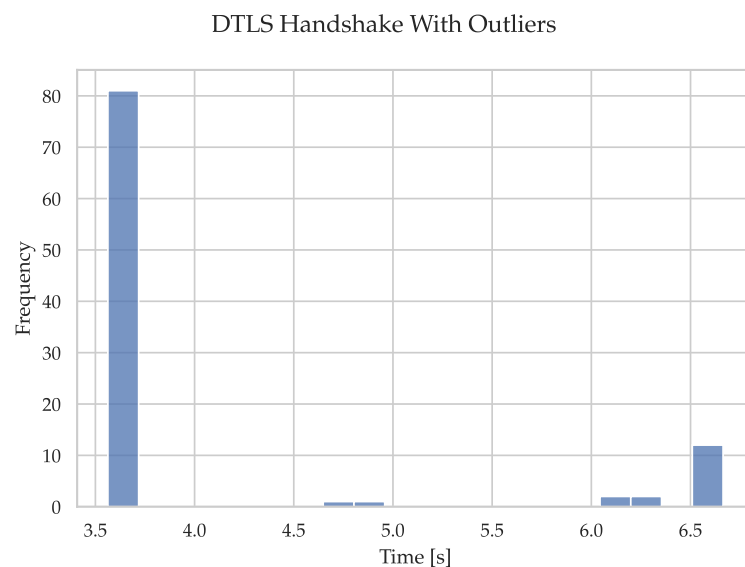


Figure 12. Execution time of the DTLS handshake using MbedTLS including outliers.

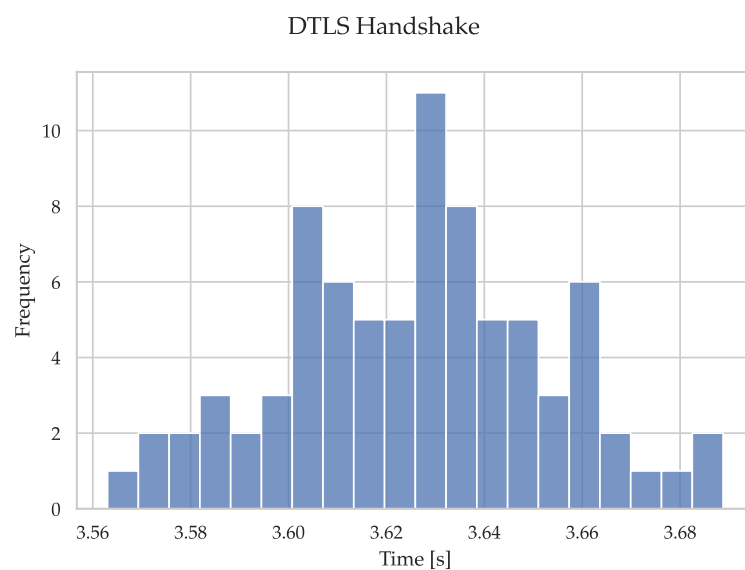
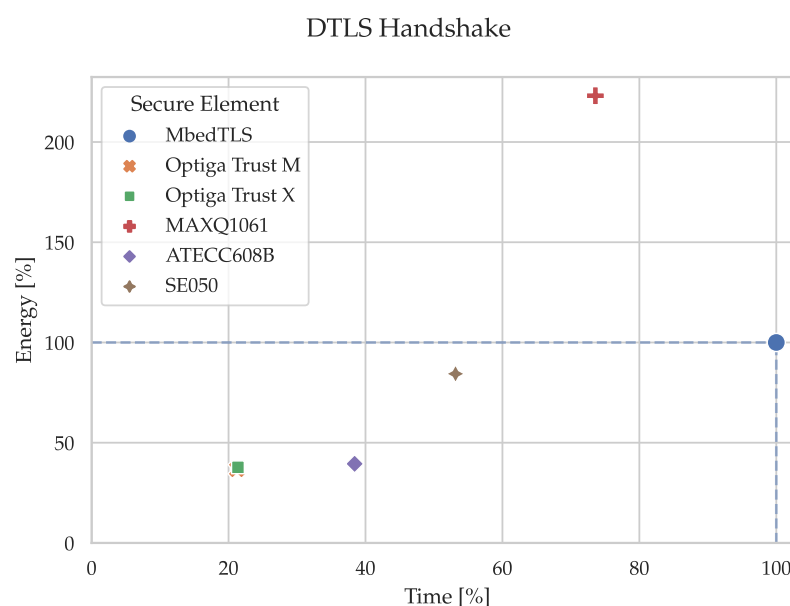


Figure 13. Execution time of the DTLS handshake using MbedTLS without outliers differing more than 20% from the median.

Table 6. Measurement count for the DTLS handshake after removing outliers caused by OpenThread.

Handshake Measurement Count	n
MbedTLS	81
Optiga Trust M	85
Optiga Trust X	82
MAXQ1061	79
ATECC608B	75
SE050	78

Figure 14 and Table 7 show the time and energy consumption of the node (secure element and MCU) for the DTLS handshake in percentages relative to the reference implementation using only MbedTLS on the MCU. E.g., the ATECC608B takes 39% of the energy consumed by the reference implementation when performing a DTLS handshake. I.e., substituting the software-based cryptographic operations in MbedTLS with the hardware-based alternatives in the ATECC608B decreases the energy consumption of the DTLS handshake by 61%. The percentages have been calculated by dividing the median time and energy consumption of the node (secure element and MCU) by the reference (MCU by itself). The corresponding median values of the absolute and relative measurements are listed in the tables in Appendix C.2.

**Figure 14.** Time and energy measured for a DTLS handshake relative to the reference implementation. (median of values, rounded to the nearest percent, n = (see Table 6)).**Table 7.** Table of values for Figure 14.

Operation	Time [%]	Energy [%]
MbedTLS	100	100
Optiga Trust M	21	37
Optiga Trust X	21	38
MAXQ1061	74	223
ATECC608B	38	39
SE050	53	84

As with the cryptographic primitives, we calculated the RSD for all adjusted handshake measurements. The largest RSD resulted from the execution time of the OPTIGA™ Trust M

at 3.4%. Again, this is an indication of how tightly grouped the measurements are, and thus only the median is listed in this section, and all the remaining statistical values are available in the supplementary data (refer to the supplementary materials statement after Section 9).

6. Discussion

Importantly, even though this paper focuses on the performance of secure elements, their use is not limited to improving execution time and energy consumption. When used correctly, they also significantly improve the IoT device's overall security. Furthermore, a potential increase in execution time or energy consumption compared to the pure MbedTLS reference is often justified by the gained security.

6.1. Cryptographic Primitives

The results of the cryptographic primitives show that, as expected, no blanket statements can be made about the use of secure elements in regard to execution time and energy savings. However, looking at the primitives where the secure elements generally exhibited a higher performance as the software-based reference (i.e., generating an ECC key pair and ECDSA sign/verify), the performance increase was particularly evident in the execution time. In order to be able to make further statements about the results, the primitives require to be examined individually.

As mentioned in Section 4.1.1, the CTR_DRBG needs to be seeded with a strong entropy source. This source would be either the TRNG within the MCU in case of the reference implementation or the externally connected secure element. By default, MbedTLS reseeds the CTR_DRBG after every 10,000 calls to `mbedtls_ctr_drbg_random`, which is used to retrieve random data. Furthermore, the prediction resistance can also require the CTR_DRBG to be reseeded if it is enabled. Thus, the internal TRNG or the externally connected secure element will not be polled for random data with every call to `mbedtls_ctr_drbg_random`. Requesting 32 bytes of random data from the CTR_DRBG without reseeding only requires 0.246 ms and 2.76 μ J, which significantly reduces the average time and energy required for getting random data. Of course, one has to decide on the tradeoff between reducing the energy consumption but still reseeding the CTR_DRBG frequently enough in order to keep the prediction resistance high. It is important to note that in contrast to most TRNGs in the secure elements, the TRNG in the nRF52840 is not compliant with any certificate. Thus, the higher time and energy consumption is clearly justified as strong sources of randomness are at the foundation of information security.

Using a secure element for calculating the SHA-256 over 32 bytes requires vastly more time and energy than the reference implementation. Arguably, calculating the hash on a secure element does not improve the overall security of an IoT device as hashing does not involve any cryptographic secret or key. Thus, the data might as well just be hashed internally.

Looking at the three primitives containing asymmetric cryptography, we see that secure elements shorten the execution time considerably in 13 out of 15 measurements while decreasing the energy consumption in 12 out of 15. Depending on your specific application, it is possible to either significantly improve or degrade the performance in terms of execution time and energy consumption. Therefore, a suitable secure element must be carefully evaluated and selected. This necessity reinforces the need for quantitative comparisons, as presented in this paper, so that developers can improve their application's overall security without sacrificing performance.

6.2. DTLS Handshake

All secure elements noticeably save time when compared to the reference implementation. This saving is due to the MbedTLS hooks introduced in Section 4.5.1, which allow the ECDSA signature creation and verification to be offloaded to the secure elements. As observed with the cryptographic primitives, ECDSA computations benefit significantly from the hardware acceleration supplied by the secure elements. Furthermore, storing the sensitive key pairs and certificates in the tamper-resistant memory of a secure element, rather than

in the flash of an MCU, can significantly improve protection. Although generating random numbers with the MCU's internal TRNG was faster and more energy-efficient, we still added this as a hook for the DTLS handshake to make use of the secure elements' certified TRNGs. Since this hook is only needed for reseeding the CTR_DRBG, the negative impact on time and energy is negligible and therefore clearly justifiable. Moreover, we did not hook the computation of hashes such as SHA-256 since this neither improves performance nor security. Finally, cipher suites using the elliptic curve Diffie-Hellman ephemeral (ECDHE) key exchange require an ECC key pair to be generated and subsequently used in a key exchange for every handshake. MbedTLS again provides hooks to offload these two computationally-expensive computations to the connected secure element. Unfortunately, it turned out that we could not simply implement these hooks for one of the secure elements and then just duplicate the code with some minor modifications, mainly because their API structures are vastly different. While one requires a session (between the secure element and the host MCU) to be opened, another does not allow the import of arbitrary public keys without sticking to a strict key importation process. The ECDHE key exchange could not be implemented for the MAXQ1061 and the SE050 without changes to MbedTLS itself. This should not be a problem in and of itself but was considered out of scope for this project. Therefore, further significant time and energy savings can still be expected for these two secure elements.

6.3. Performance Comparison

The Microchip ATECC608B was the most energy-efficient secure element in 3 out of 5 tested primitives when compared to the other secure elements. Especially with resource-constrained devices, this can be a huge selling point. Although it is important to note that it was never the fastest one, and a complete DTLS stack needs to be present in the MCU for performing handshakes as it cannot handle the handshake on its own. This requires a substantial amount of memory and might end in selecting an MCU with more memory than otherwise needed.

Interestingly, even though the Infineon OPTIGA™ Trust M is the successor of the OPTIGA™ Trust X, it didn't outperform its predecessor as their results were either tied or only marginally different. Therefore, the selection between these two secure elements should be purely based on their respective feature set.

The Maxim MAXQ1061 requires the most energy out of all secure elements while performing asymmetric cryptography. On the other hand, it is more than twice as fast as the second-ranked secure element when generating random data. Additionally, the MAXQ1061 can be connected using SPI with a clock speed of up to 4 MHz (20 MHz for when using the AES stream encryption engine) which is the fastest connectivity out of all the secure elements. As data transmission is a significant portion of the execution time, this provides a significant improvement of the energy consumption as well. Furthermore, the newly released MAXQ1065 has an extended feature set by adding a physically unclonable function (PUF), extended DTLS support, etc. More importantly, it should be able to perform the tested primitives with significantly less energy given that the complete communication can run over SPI with a maximal clock speed of 20 MHz and an active supply current of 5 mA compared to the 17 mA of the MAXQ1061.

Lastly, the NXP SE050 was clearly the fastest secure element for ECDSA signing and verifying. Additionally, it was never the best but also never the worst regarding energy consumption, which earns it the all-rounder title.

6.4. Feature Set and Development

From a developer's point of view, the OPTIGA™ Trust M and OPTIGA™ Trust X allowed for the most straightforward integration. This is mainly because they provide a lot of publicly available resources like extensive documentation, examples, and ready-to-use driver implementations (e.g., Nordic nRF5 SDK, Zephyr, ...). Furthermore, the same software framework can be used for the OPTIGA™ Trust M as well as the OPTIGA™ Trust X without any changes, which makes switching between them extremely easy. Currently,

version 1 (V1) and version 3 (V3) of the OPTIGA™ Trust M are available, with the V3 having additional features (i.e., AES, HMAC, HKDF, ...). Unfortunately, the OPTIGA™ Trust M V1 and OPTIGA™ Trust X used for this project don't provide a feature for loading externally generated private keys into the secure element, and thus have to be generated internally. In contrast, the OPTIGA™ Trust M V3 is capable of such a protected key update. Although, not being able to import private keys is arguably a desirable restriction from a security standpoint as it forces the private keys to never exist outside of the secure element in the first place. However, during development or in the case of a project like this one, this is rather cumbersome as we cannot just create a key pair and a corresponding client certificate once and load it into the secure elements.

Even though the ATECC608B performed the best regarding energy consumption, it has even stricter setup requirements as it has to be configured, provisioned and then locked before even the least security-critical functions like getting random data can be executed. This constraint results in a relatively rigid development process. However, it increases security by preventing human errors (e.g., locking cannot be forgotten if the device is subjected to a functional test as it simply would not work without it being locked). In contrast, other secure elements can be deployed without locking them.

Even though the MAXQ1061 required significantly more energy for the asymmetric cryptography compared to the ATECC608B, it provides considerably more flexibility during development in regards to not having to lock the provisioned data before being able to execute anything. It features a life-cycle model with 4 states (Delivery, Initialized, Operational, Terminated) and four security conditions (Anyone, Admin, Host, Secure boot), otherwise known as roles. Finally, for any given life-cycle state, an object in the internal memory can be assigned any combination of the five object access rules (Read/Export, Write/Import, Execute, Generate, Delete) and the previously mentioned security conditions. This culminates in the most extensive role-based access model compared to all the other secure elements. Furthermore, 32 kB of user-programmable EEPROM storage is significantly more flexible than other secure elements where the memory is already partitioned in fixed slots.

The SE050's access model might not be as extensive as the MAXQ1061's. However, it features up to 50 kB of user-programmable storage, which should satisfy all secure storage-related needs of any given project. Moreover, the SE050 has the largest feature set of all evaluated secure elements regarding algorithms, supported elliptic curves, and bit lengths. It features another I2C peripheral where it acts as a master in order to read and write sensors securely. NXP provides extensive API documentation and corresponding samples, which speed up the development immensely.

6.5. Which Secure Element Is the Right One for the Job?

The measurements show that the fastest secure element for any given task is not necessarily the most energetically favorable. The same observation also holds in the other direction, where the most energetically favorable secure element is not necessarily the fastest. Furthermore, each secure element achieves the first or second rank in execution time or energy consumption for at least one cryptographic primitive. A look at the general description of the secure elements also shows that their feature set does not allow any conclusions to be drawn about their performance or certification.

These are essential findings and underline our motivation for this project. This is why it is so important for developers to have easy access to information and comparisons, because depending on the project, a different secure element may be the best choice to achieve the optimum in terms of performance and energy consumption.

7. Key Findings

Working with secure elements has shown their great potential and need in the IoT-industry. Secure elements support resource-constrained devices by performing cryptographic algorithms in hardware and providing tamper-resistant storage. Especially in asymmetric cryptography, we were able to improve time and energy efficiency compared to the software-

based reference implementation. Conversely, the generation of random numbers resulted in a lower performance independent of the used secure element. However, it is imperative to note that some of the secure element TRNGs are certified, which is not the case for the internal TRNG of the nRF52840. Thus, their use is quite justified despite lower performance.

Furthermore, we strongly advise against hashing with secure elements because the time and energy consumption exceeds the reference by several orders of magnitude. Especially as there is also no increase in the security level. In contrast, the secure storage of sensitive material, such as private keys, long-lived session keys, and root-of-trust certificates, provides an opportunity to increase the overall security of embedded devices significantly.

The results of the measurements clearly show that no general statements can be made as to which is the best or the worst performing one. Other metrics like feature set, development experience, and cost must also be considered for this decision because they might justify a slightly lower performance in some cases. The specific use case doesn't just dictate which metric is more important but also what is "better" or "worse". e.g., one would intuitively define a larger feature set as the better one. However, a project might require some lesser-known curve or algorithm that just happens to be implemented by a secure element with an otherwise significantly smaller feature set. Fortunately, the measurement data in this paper, together with the respective data sheets, should allow developers to make an informed decision.

It all comes down to the availability of extensive and—most importantly—publicly accessible information for developers to find a suitable secure element for their specific applications. Until recently, this was made much more difficult because NDAs had to be signed for accessing the documentation and SDK of each secure element. Fortunately, more and more manufacturers are moving away from such prohibitive practices, and at the time of publishing this paper, you only had to sign an NDA for one out of five chips. We strongly believe that detailed and publicly available information significantly lowers the threshold for using secure elements.

8. Related Work

As outlined in Section 2, we cannot find any quantitative comparisons of secure elements even with extensive searches. Works that provide performance figures mostly compare a single secure element against other types of hardware acceleration. For example, P. Kietzman et al. [31] compared the performance of the ARM CryptoCell 310 (contained in the nRF52840) to the ATECC608A with various symmetric and asymmetric algorithms. They obtained comparable time and energy consumption results during ECC key generation, ECDSA signing and verifying (secp256r1) to our measurements with the ATECC608B. In another publication, P. Kietzmann et al. [32] provide a guideline on PRNGs in the IoT by comparing various aspects (like statistical properties, execution time, energy consumption, and memory overhead) of software- and hardware-based solutions. Unfortunately, their results using the ATECC508A (predecessor of the ATECC608A) cannot be directly compared to ours as these two secure elements do not contain the same RNG. According to Microchip's application note AN2589 [33], "the ATECC608A includes an enhanced high-quality cryptographic random number generator [...]" when compared to its predecessor.

B. Pearson et al. [34] acquired a faster ECDSA signature generation (−4.2%) using the ATECC608B and secp256r1 and ECDSA signature verification (−19%). However, it is unclear if they start the time measurement before calling the first ATECC608B API function and stop the time after the last API call returns, or if they measured the time between the first and last I2C packet. This point would, of course, explain the observed differences. Moreover, the listed MCUs (Texas Instruments CC3220SF, Espressif ESP8266, Espressif ESP32) have maximum clock speeds of 80, 160, and 160 MHz, respectively. This is consistently higher than the nRF52840, which could lead to faster execution of the API calls and thus result in shorter execution times.

R. A. Nofal et al. [35] conducted extensive measurements of the TLS handshake and record layer. Amongst other things, they measured the execution time and energy consump-

tion of ECDSA signing and verifying (secp256r1) on a BCM4343W which is also an ARM Cortex M4 MCU. As cryptographic libraries, they used MbedTLS, as well as the micro-ECC (μ ECC) library [36]. Interestingly, our implementation using MbedTLS on a slower MCU resulted in a 56% faster ECDSA sign and 26% faster ECDSA verify operation. Conversely, their μ ECC implementation is even faster (signing 56% and verifying 85% faster). E. Nascimento [37] and T. Silde [38] concluded independently that μ ECC's side channel protections are not properly documented. Additionally, T. Silde states that further security measures for differential power analysis are required to be on par with MbedTLS.

9. Conclusions & Future Work

Secure elements can support resource-constrained devices in terms of security by providing tamper-resistant memory, a variety of cryptographic algorithms, and the on-chip generation of key pairs with the private key never leaving the secure element. Thus, they allow increasing the overall security of an IoT device. Unfortunately, there are few to no quantitative comparisons between secure elements from different manufacturers as they have been reluctant to publicly release information about their devices' hard- and software. This paper provides detailed measurement results of cryptographic primitives and a real-world application use case executed on five different secure elements to combat this information gap. The measurement results indicate that secure elements can improve the battery life of embedded devices depending on the respective execution time and energy consumption of the cryptographic algorithm. Furthermore, the paper presents detailed information about the evaluated secure elements to supply developers with additional information to select a suitable secure element for a given application.

The benchmark used in this paper tests only a few primitives and a single real-world use case. Since these were chosen strategically, they already provide much information about the performance of the secure elements. Nevertheless, future work should significantly expand the benchmark in order to supply the developers with a more extensive overview of the available secure elements. For example, the already used primitives should be executed for various data lengths where applicable (e.g., random generation and SHA-256), and other curves could be used for the asymmetric primitives (e.g., secp256k1). Moreover, primitives that were not tested in the current benchmark, such as message authentication codes (MACs) or symmetric ciphers, should also be included. Finally, secure elements from other manufacturers should also be added in order to be able to create an even more comprehensive benchmark.

Supplementary Materials: The processed data and the corresponding statistical analysis presented in this study are available in CSV files at <https://www.mdpi.com/article/10.3390/iot3010001/s1>.

Author Contributions: Conceptualization, M.N., L.Z., T.S. and A.R.; software, M.N., L.Z. and T.S.; data acquisition, M.N.; writing—original draft preparation, M.N.; writing—review and editing, L.Z., T.S. and A.R.; supervision and project administration, A.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The processed data presented in this study is available in the supplementary material attached to this article. Contact the authors directly for requesting access to raw measurement data.

Acknowledgments: We thank Infineon Technologies AG, Maxim Integrated, Microchip Technology Inc., and NXP Semiconductors N.V. for their generous support during the reviewing process of this manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks
AES	Advanced Encryption Standard
AG	Aktiengesellschaft (US: Incorporated)
API	Application Programming Interface
CA	Certificate Authority
CSPRNG	Cryptographically Secure Pseudorandom Number Generator
CTR_DRBG	Block-Cipher Counter-Mode Based Deterministic Random Bit Generator
CV	Coefficient of Variation
CoAP(s)	Constrained Application Protocol (Secure)
DK	Development Kit
(D)TLS	(Datagram) Transport Layer Security
EAL	Evaluation Assurance Level
ECC	Elliptic Curve Cryptography
ECDH(E)	Elliptic Curve Diffie-Hellmann (Ephemeral)
ECDSA	Elliptic Curve Digital Signature Algorithm
EEPROM	Electrically Erasable Programmable Read-only Memory
GND	Ground
GPIO	General-purpose input/output
HKDF	HMAC-Based Key Derivation Function
(H)MAC	(Hashed) Message Authentication Code
HTTP	Hypertext Transfer Protocol
HW	Hardware
I2C	Inter-integrated Circuit
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP(v6)	Internet Protocol (version 6)
InES	Institute of Embedded Systems
IoT	Internet of Things
KDF	Key Derivation Function
MCU	Microcontroller Unit
MDPI	Multidisciplinary Digital Publishing Institute
MTU	Maximum Transmission Unit
N.V.	Naamloze Vennootschap (US: Incorporated)
NDA	Non-disclosure Agreement
NIST	National Institute of Standards and Technology
PCB	Printed Circuit Board
PKCS	Public Key Cryptography Standards
PRF	Pseudorandom Function Family
PUF	Physical Unclonable Function
REST	Representational State Transfer
RESTful	API that conforms to REST
RFC	Request for Comments
(T D N)RNG	(True Deterministic Non-deterministic) Random Number Generator
RSA	Rivest-Shamir-Adleman (initials of the authors of the cryptosystem)
RSD	Relative Standard Deviation
RTOS	Real-time Operating System
SDK	Software Development Kit
SECG	Standards for Efficient Cryptography Group
SHA	Secure Hash Algorithms
SPI	Serial Peripheral Interface
SWI	Single-wire Interface
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus

Appendix A. Hardware

Table A1 contains the hardware used for the devices under test as well as the test infrastructure (e.g., OpenThread border router).

Table A1. Hardware used for the test setup.

Hardware	Manufacturer
ATECC608B	Microchip Technology Inc. Chandler, Arizona, United States
MAXQ1061	Maxim Integrated San Jose, California, United States
nRF52840 DK & Dongle	Nordic Semiconductor Trondheim, Trøndelag, Norway
OPTIGA Trust M V1 & X	Infineon Technologies AG Neubiberg, Bavaria, Germany
Raspberry Pi 3 Model B+	Raspberry Pi Foundation Cambridge, England, UK
SE050	NXP Semiconductors N.V. Eindhoven, North Brabant, Netherlands

Appendix A.1. nRF52840 DK

Table A2 contains the configuration and modification required for an nRF52840 DK version 2.0.1 for reproducing the measurements.

Table A2. Configuration and modification of the nRF52840 DK.

Part	Configuration/Modification
SW6	nRF only
SW9	VDD
SW10	VEXT -> nRF
P0.03	100 kΩ pull-down to GND

The host MCU resets the OPTIGA™ Trust M, OPTIGA™ Trust X, and MAXQ1061 using their dedicated RESET pin. On the secure element shield, the RESET pins of the secure elements are connected to the RESET pin of the Arduino pin header. Lastly, this connects to P0.18 of the nRF52840 which is also the RESET pin of the MCU. In order for the MCU to be able to independently control this pin, the reset button (SW5) needs to be disconnected with the modifications listed in Table A3.

Table A3. Modification of the nRF52840 DK for disconnecting the reset button.

Trace-Cut Jumper	Modification
SB42	open
SB44	closed
SB56	open

Appendix A.2. Measurement Setup

Figure A1 illustrates how the Keysight N6705B power analyzer is connected to the secure node. It supplies the nRF52840 MCU as a constant voltage source with 3.3 V over channel 1 while simultaneously monitoring the voltage and current of the MCU with the same channel. Channel 2 is configured as a voltmeter and is connected to the additional GPIO (P0.03) which signals when the individual cryptographic primitives and the DTLS handshake are currently being executed. The rest of the DK (on-board JLink probe, etc.) is supplied over USB (J2) as this would not be present on a custom board. Thus, it is supplied

separately in order to prevent any erroneous energy measurements of the MCU. The DK contains a regulator that provides a 3.3 V rail which is then passed through channel 3 configured as an ammeter and finally used to power the secure element.

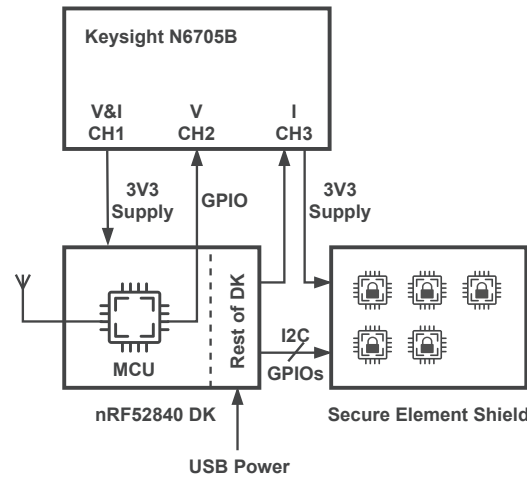


Figure A1. Measurement setup of the secure node for acquiring the execution time and energy consumption during the stated test cases.

Appendix B. Software

Table A4 contains the versions of all the libraries, frameworks, and SDKs used in this project. They were further modified to solve incompatibilities and add missing functionality.

Table A4. Software versions of the used libraries and frameworks.

Software	Version	Year	Manufacturer
Californium	2.5.0	2020	Eclipse Foundation Ottawa, Ontario, Canada
cryptoauthlib	20180329	2018	Microchip Technology Inc. Chandler, Arizona, United States
MAXQ1061 SDK	5.3.0	2020	Maxim Integrated San Jose, California, United States
MbedTLS	2.16.4	2020	ARM Ltd. Cambridge, England, UK
OpenThread	ab89f40	2020	Google LLC Mountain View, California, US
optiga-trust-m	1.30.885	2019	Infineon Technologies AG Neubiberg, Bavaria, Germany
plug-and-trust	03.00.02	2020	NXP Semiconductors N.V. Eindhoven, North Brabant, Netherlands
Zephyr	2.2.0	2020	Linux Foundation San Francisco, California, US

Appendix C. Measurement Values

Appendix C.1. Cryptographic Primitives

Tables A5 and A6 contain the calculated medians of all time and energy measurements while executing the cryptographic primitives. In both tables, the column “MbedTLS” again represents the reference implementation and lists the execution time and energy consumption of the MCU by itself. The other columns represent the time and energy required by the MCU and the corresponding secure element combined and list the absolute values as well as the percentages relative to the reference. E.g., during the ECC key pair generation, the MCU using MbedTLS consumed 12.4 mJ while the ATECC608B combined with the MCU only consumed 2.83 mJ. Thus, the hardware implementation using the ATECC608B required only 22.8% of the energy that the reference implementation consumed for the same task. Furthermore, Table A7 lists the energy consumption of the secure element itself without the MCU during the cryptographic primitives.

Table A5. Absolute execution time (abs.) of the node (MCU) during the cryptographic primitives using MbedTLS as a reference and absolute execution time of the node (MCU and secure element) during the cryptographic primitives using the secure elements with percentages relative (rel.) to reference (median of values, rounded to 3 significant digits, n = 100).

Time [s]	MbedTLS	Optiga Trust M		Optiga Trust X		MAXQ1061		ATECC608B		SE050	
	Abs.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.
Random	5.90×10^{-3}	1.44×10^{-2}	$2.44 \times 10^{+2\%}$	1.44×10^{-2}	$2.44 \times 10^{+2\%}$	5.57×10^{-3}	$9.44 \times 10^{+1\%}$	1.93×10^{-2}	$3.28 \times 10^{+2\%}$	1.51×10^{-2}	$2.56 \times 10^{+2\%}$
Gen. ECC Key Pair	4.41×10^{-1}	7.59×10^{-2}	$1.72 \times 10^{+1\%}$	6.86×10^{-2}	$1.56 \times 10^{+1\%}$	$1.36 \times 10^{+0}$	$3.08 \times 10^{+2\%}$	8.50×10^{-2}	$1.93 \times 10^{+1\%}$	1.13×10^{-1}	$2.56 \times 10^{+1\%}$
SHA-256 Hash	8.19×10^{-5}	6.80×10^{-2}	$8.30 \times 10^{+4\%}$	7.54×10^{-2}	$9.21 \times 10^{+4\%}$	1.21×10^{-2}	$1.48 \times 10^{+4\%}$	1.66×10^{-2}	$2.03 \times 10^{+4\%}$	2.05×10^{-2}	$2.50 \times 10^{+4\%}$
ECDSA Sign	1.85×10^{-1}	7.71×10^{-2}	$4.17 \times 10^{+1\%}$	7.71×10^{-2}	$4.17 \times 10^{+1\%}$	3.05×10^{-1}	$1.65 \times 10^{+2\%}$	9.31×10^{-2}	$5.03 \times 10^{+1\%}$	4.93×10^{-2}	$2.67 \times 10^{+1\%}$
ECDSA Verify	6.09×10^{-1}	9.42×10^{-2}	$1.55 \times 10^{+1\%}$	8.79×10^{-2}	$1.44 \times 10^{+1\%}$	5.40×10^{-1}	$8.87 \times 10^{+1\%}$	5.59×10^{-2}	$9.18 \times 10^{+0\%}$	4.77×10^{-2}	$7.83 \times 10^{+0\%}$

Table A6. Absolute energy consumption (abs.) of the node (MCU) during the cryptographic primitives using MbedTLS as a reference and absolute energy consumption of the node (MCU and secure element) during the cryptographic primitives using the secure elements with percentages relative (rel.) to reference (median of values, rounded to 3 significant digits, n = 100).

Energy [J]	MbedTLS	Optiga Trust M		Optiga Trust X		MAXQ1061		ATECC608B		SE050	
	Abs.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.
Random	1.66×10^{-4}	7.44×10^{-4}	$4.49 \times 10^{+2\%}$	7.67×10^{-4}	$4.63 \times 10^{+2\%}$	4.44×10^{-4}	$2.68 \times 10^{+2\%}$	5.06×10^{-4}	$3.06 \times 10^{+2\%}$	7.33×10^{-4}	$4.42 \times 10^{+2\%}$
Gen. ECC Key Pair	1.24×10^{-2}	4.29×10^{-3}	$3.45 \times 10^{+1\%}$	4.01×10^{-3}	$3.23 \times 10^{+1\%}$	1.13×10^{-1}	$9.06 \times 10^{+2\%}$	2.83×10^{-3}	$2.28 \times 10^{+1\%}$	5.69×10^{-3}	$4.57 \times 10^{+1\%}$
SHA-256 Hash	2.34×10^{-6}	3.66×10^{-3}	$1.57 \times 10^{+5\%}$	4.17×10^{-3}	$1.78 \times 10^{+5\%}$	9.62×10^{-4}	$4.12 \times 10^{+4\%}$	4.33×10^{-4}	$1.85 \times 10^{+4\%}$	9.73×10^{-4}	$4.17 \times 10^{+4\%}$
ECDSA Sign	5.20×10^{-3}	4.37×10^{-3}	$8.41 \times 10^{+1\%}$	4.50×10^{-3}	$8.66 \times 10^{+1\%}$	2.55×10^{-2}	$4.90 \times 10^{+2\%}$	3.08×10^{-3}	$5.92 \times 10^{+1\%}$	2.65×10^{-3}	$5.11 \times 10^{+1\%}$
ECDSA Verify	1.72×10^{-2}	5.39×10^{-3}	$3.14 \times 10^{+1\%}$	5.23×10^{-3}	$3.05 \times 10^{+1\%}$	4.49×10^{-2}	$2.62 \times 10^{+2\%}$	2.05×10^{-3}	$1.19 \times 10^{+1\%}$	2.63×10^{-3}	$1.53 \times 10^{+1\%}$

Table A7. Energy consumption of the secure elements without the MCU during the cryptographic primitives (median of values, rounded to 3 significant digits, n = 100).

Energy [J]	Optiga Trust M	Optiga Trust X	MAXQ1061	ATECC608B	SE050
Random	4.22×10^{-4}	4.45×10^{-4}	3.17×10^{-4}	8.95×10^{-5}	3.26×10^{-4}
Gen. ECC Key Pair	2.67×10^{-3}	2.54×10^{-3}	8.08×10^{-2}	1.03×10^{-3}	2.64×10^{-3}
SHA-256 Hash	2.07×10^{-3}	2.42×10^{-3}	6.84×10^{-4}	6.31×10^{-5}	4.21×10^{-4}
ECDSA Sign	2.71×10^{-3}	2.84×10^{-3}	1.90×10^{-2}	1.09×10^{-3}	1.32×10^{-3}
ECDSA Verify	3.36×10^{-3}	3.34×10^{-3}	3.36×10^{-2}	8.35×10^{-4}	1.35×10^{-3}

Appendix C.2. DTLS Handshake

Tables A8 and A9 list the time and energy required for executing the DTLS handshake. They are constructed in the same way as Tables A5 and A6, with the percentages again being relative to the reference (column “MbedTLS”).

Table A8. Absolute execution time (abs.) of the node (MCU) during the DTLS handshake using MbedTLS (reference) and absolute execution time of the node (MCU and secure element) during the handshake using the secure elements with percentages relative (rel.) to the reference (median of values, rounded to 3 significant digits, n = (see table 6)).

Time [s]	MbedTLS	Optiga Trust M		Optiga Trust X		MAXQ1061		ATECC608B		SE050	
	Abs.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.
DTLS Handshake	$3.63 \times 10^{+0}$	7.71×10^{-1}	$2.12 \times 10^{+1}\%$	7.77×10^{-1}	$2.14 \times 10^{+1}\%$	$2.67 \times 10^{+0}$	$7.36 \times 10^{+1}\%$	$1.40 \times 10^{+0}$	$3.84 \times 10^{+1}\%$	$1.93 \times 10^{+0}$	$5.31 \times 10^{+1}\%$

Table A9. Absolute energy consumption (abs.) of the node (MCU) during the DTLS handshake using MbedTLS (reference) and absolute energy consumption of the node (MCU and secure element) during the handshake using the secure elements with percentages relative (rel.) to the reference (median of values, rounded to 3 significant digits, n = (see table 6)).

Energy [J]	MbedTLS	Optiga Trust M		Optiga Trust X		MAXQ1061		ATECC608B		SE050	
	Abs.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.
DTLS Handshake	9.85×10^{-2}	3.62×10^{-2}	$3.67 \times 10^{+1}\%$	3.72×10^{-2}	$3.77 \times 10^{+1}\%$	2.20×10^{-1}	$2.23 \times 10^{+2}\%$	3.89×10^{-2}	$3.95 \times 10^{+1}\%$	8.31×10^{-2}	$8.44 \times 10^{+1}\%$

References

1. Thread Group. What is Thread. Available online: <https://www.threadgroup.org/What-is-Thread/Overview> (accessed on 28 April 2021).
2. Schläpfer, T.; Rüst, A. Security on IoT Devices with Secure Elements. In Proceedings of the Embedded World Conference, Nuremberg, Germany, 26–28 February 2019. Available online: <https://doi.org/10.21256/zhaw-3350> (accessed on 6 May 2021).
3. Zimmerli, L.; Schläpfer, T.; Rüst, A. Securing the IoT: Introducing an Evaluation Platform for Secure Elements. In Proceedings of the Konferenz Internet of Things—vom Sensor bis zur Cloud, Stuttgart, Germany, 20 November 2019. Available online: [doi:10.21256/zhaw-18794](https://doi.org/10.21256/zhaw-18794) (accessed on 6 May 2021). [CrossRef]
4. Shelby, Z.; Hartke, K.; Bormann, C. RFC7252: The Constrained Application Protocol (CoAP). *IETF*. June 2014. Available online: <https://www.rfc-editor.org/rfc/rfc7252.html> (accessed on 6 May 2021).
5. Montenegro, G.; Hui, J.; Culler, D.; Kushalnagar, N. RFC4944: Transmission of IPv6 Packets over IEEE 802.15.4 Networks. *IETF*. September 2007. Available online: <https://rfc-editor.org/rfc/rfc4944.html> (accessed on 28 April 2021).
6. MbedTLS. Available online: <https://tls.mbed.org/> (accessed on 23 August 2021).
7. Certicom Research. SEC 2: Recommended Elliptic Curve Domain Parameters. *SECG*. January 2010. Available online: <https://www.secg.org/sec2-v2.pdf> (accessed on 23 August 2021).
8. Barker, E.; Kelsey, J. NIST Special Publication 800-90A Revision 1: Recommendation for Random Number Generation Using Deterministic Random Bit Generators. *NIST*. June 2015. Available online: <https://csrc.nist.gov/publications/detail/sp/800-90a/rev-1/final> (accessed on 11 June 2021).
9. nRF52840 DK. Available online: <https://www.nordicsemi.com/Products/Development-hardware/nrf52840-dk> (accessed on 31 August 2021).
10. Raspberry Pi 3 Model B+. Available online: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/> (accessed on 31 August 2021).
11. nRF52840 Dongle. Available online: <https://www.nordicsemi.com/Products/Development-hardware/nRF52840-Dongle> (accessed on 1 August 2021).
12. Infineon OPTIGA Trust M. Available online: <https://www.infineon.com/cms/en/product/security-smart-card-solutions/optiga-embedded-security-solutions/optiga-trust/optiga-trust-m-sls32aia/> (accessed on 28 September 2021).
13. Optiga-trust-m. Available online: <https://github.com/Infineon/optiga-trust-m> (accessed on 28 September 2021).
14. Infineon OPTIGA Trust X. Available online: <https://www.infineon.com/cms/en/product/security-smart-card-solutions/optiga-embedded-security-solutions/optiga-trust/optiga-trust-x-sls-32aia/> (accessed on 28 September 2021).
15. Optiga-trust-x. Available online: <https://github.com/Infineon/optiga-trust-x> (accessed on 28 September 2021).
16. Maxim MAXQ1061. Available online: <https://www.maximintegrated.com/en/products/embedded-security/MAXQ1061.html> (accessed on 28 September 2021).
17. Microchip ATECC608B. Available online: <https://www.microchip.com/en-us/product/ATECC608B> (accessed on 28 September 2021).
18. Cryptoauthlib. Available online: <https://github.com/MicrochipTech/cryptoauthlib> (accessed on 28 September 2021).
19. NXP SE050. Available online: <https://www.nxp.com/products/security-and-authentication/authentication/edgelock-se050-plugin-trust-secure-element-family-enhanced-iot-security-with-maximum-flexibility:SE050> (accessed on 28 September 2021).
20. plug-and-trust. Available online: <https://github.com/NXP/plug-and-trust> (accessed on 28 September 2021).
21. The Common Criteria. Available online: <https://www.commoncriteriaportal.org> (accessed on 6 May 2021).
22. Turan, M.S.; Barker, E.; Kelsey, J.; McKay, K.A.; Baish, M.L.; Boyle, M. NIST Special Publication 800-90B: Recommendation for the Entropy Sources Used for Random Bit Generation. *NIST*. January 2018. Available online: <https://csrc.nist.gov/publications/detail/sp/800-90b/final> (accessed on 11 June 2021).
23. Barker, E.; Kelsey, J. (Second Draft) NIST Special Publication 800-90C: Recommendation for Random Bit Generator (RBG) Constructions. *NIST*. April 2016. Available online: <https://csrc.nist.gov/publications/detail/sp/800-90c/draft> (accessed on 11 June 2021).
24. German Federal Office for Information Security. AIS-31. May 2013. Available online: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_pdf.pdf?__blob=publicationFile&v=1 (accessed on 29 September 2021).
25. The Zephyr Project. Available online: <https://www.zephyrproject.org> (accessed on 6 May 2021).
26. OpenThread. Available online: <https://openthread.io> (accessed on 6 May 2021).
27. Alternative Cryptography Engines Implementation. Available online: https://tls.mbed.org/kb/development/hw_acc_guidelines (accessed on 12 June 2021).
28. Eclipse Californium. Available online: <https://www.eclipse.org/californium> (accessed on 6 May 2021).
29. Apache Maven Project. Available online: <https://maven.apache.org> (accessed on 6 May 2021).
30. Moriarty, K.; Nystrom, M.; Parkinson, S.; Rusch, A.; Scott, M. RFC7292: PKCS #12: Personal Information Exchange Syntax v1.1. *IETF*. July 2014. Available online: <https://www.rfc-editor.org/rfc/rfc7292.html> (accessed on 6 May 2021).
31. Kietzmann, P.; Boeckmann, L.; Lanzieri, L.; Schmidt, T.C.; Wählich, M. A Performance Study of Crypto-Hardware in the Low-End IoT. *EWSN, ACM*. 2021. Available online: <https://eprint.iacr.org/2021/058.pdf> (accessed on 31 May 2021).
32. Kietzmann, P.; Schmidt, T.C.; Wählich, M. A Guideline on Pseudorandom Number Generation (PRNG) in the IoT. *ACM*, 2021. Available online: <https://dl.acm.org/doi/abs/10.1145/3453159> (accessed on 30 November 2021). [CrossRef]

33. Logaswamy, K. AN2589: Differences between the ATECC608A and ATECC508A CryptoAuthentication Devices. Available online: <http://ww1.microchip.com/downloads/en/Appnotes/00002589A.pdf> (accessed on 30 November 2021).
34. Pearson, B.; Luo, L.; Zhang, Y.; Dey, R.; Ling, Z.; Bassiouni, M.; Fu, X. On Misconception of Hardware and Cost in IoT Security and Privacy. In Proceedings of the 2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019. Available online: <https://ieeexplore.ieee.org/abstract/document/8761062> (accessed on 31 May 2021).
35. Nofal, R.A.; Tran, N.; Garcia, C.; Liu, Y.; Dezfouli, B. A Comprehensive Empirical Analysis of TLS Handshake and Record Layer on IoT Platforms. In Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM '19), Miami Beach, FL, USA, 25–29 November 2019. Available online: <https://dl.acm.org/doi/10.1145/3345768.3355924> (accessed on 31 May 2021).
36. Micro-ecc. Available online: <https://github.com/kmackay/micro-ecc> (accessed on 12 June 2021).
37. Nascimento, E.; Chmielewski, L.; Oswald, D.; Schwabe, P. Attacking Embedded ECC Implementations through Cmov Side Channels. Selected Areas in Cryptography 2016, St. John's, Canada, 2016. Available online: <https://cryptojedi.org/papers/cmovsca-20160718.pdf> (accessed on 12 June 2021).
38. Silde, T. Comparative Study of ECC Libraries for Embedded Devices. Norwegian University of Science and Technology. March 2019. Available online: <https://tjerandsilde.no/files/Comparative-Study-of-ECC-Libraries-for-Embedded-Devices.pdf> (accessed on 12 June 2021).