IoT Key Exchange Performance Analysis

Francesco Raimondo
f.raimondo@bristol.ac.uk
University of Bristol
United Kingdom

Ufuk Erol
u.erol@bristol.ac.uk
University of Bristol
United Kingdom

Samuel Gunner
sam.gunner@bristol.ac.uk
University of Bristol
United Kingdom

James Pope
jp16127@bristol.ac.uk
University of Bristol
United Kingdom

Robert Zakrzewski
robert.zakrzewski@bristol.ac.uk
University of Bristol
United Kingdom

Mike Faulks
mike@ioetec.com
Ioetec Ltd
United Kingdom

Ryan McConville
ryan.mcconville@bristol.ac.uk
University of Bristol
United Kingdom

Thomas Pasquier
tfjmp@cs.ubc.ca
University of British Columbia
Canada

Rob Piechocki
r.j.piechocki@bristol.ac.uk
University of Bristol
United Kingdom

George Oikonomou
g.oikonomou@bristol.ac.uk
University of Bristol
United Kingdom

ABSTRACT

The security of data, in motion and at rest, depends on the ability to exchange session keys between communicating parties. Key agreement approaches can provide the additional security assurance of perfect forward secrecy, however, for many Internet of Things resource-constrained devices the session key establishment process is too costly in terms of energy consumption and processing time. In this paper, we quantify the energy consumption and execution load when performing session key establishment. We develop a software security framework, implementing both lightweight key transport and key agreement, the latter based on elliptic curve Diffie-Hellman (ECDH). Measurements are taken using energy and digital-events monitoring tools. We find that key agreement implemented via software requires a quantity of energy thousand of times greater than a key transport approach. Also, we measure and quantify how much a hardware implementation can improve energy and execution time performance. Our research provides critical information for practitioners in selecting the appropriate hardware and security scheme for IoT applications.

CCS CONCEPTS

• **Networks → Security protocols**; Network reliability; • **Security and privacy → Key management**.

KEYWORDS

Internet of things, ECC, key exchange, key transport, energy, security

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
USWC, October 03–05, 2022, Linz, Austria
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Francesco Raimondo, Ufuk Erol, Samuel Gunner, James Pope, Robert Zakrzewski, Mike Faulks, Ryan McConville, Thomas Pasquier, Rob Piechocki, and George Oikonomou. 2018. IoT Key Exchange Performance Analysis. In *Proceedings of Unconventional Security for Wireless Communications (USWC).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The emergence of the Internet of Things (IoT) over a decade ago started intense research into resource constrained devices. Initial attempts at securing communications using traditional asymmetric key exchange protocols was deemed infeasible given the limited computational capabilities of the devices at the time. However, more recent IoT devices provide increased energy efficient computational performance. Additionally, some devices now also include asymmetric cryptographic hardware support. Understating the applicability of these approaches to constrained IoT applications mandates a thorough understanding of power and resource requirements, and yet this information is largely lacking from the literature [18]. In this paper we carry out a performance analysis of key exchange approaches on recent IoT devices. The analysis compares energy and time performance for various key exchange approaches, across multiple hardware platforms. Refresh of agreed keys increases the security of a connection, but places more burden on the constrained hardware, making the choice of refresh period a compromise. Quantitative data into the resource requirements will allow engineers to optimise this design choice.

It is becoming increasingly important that, in wireless networks, security be extended to the resource constrained 'Endpoint'. This is for two reasons. Firstly, there is a paradigm shift away from wireless networks being accessed through the secure network gateway [21] which has traditionally protected the Endpoint devices from malicious actors. Protocols now allow IPv6 to extend all the way to the resource constrained device, and with this the threats that direct access to the internet can bring. Secondly, IoT applications are becoming increasingly important to society [2], with the data they generate now informing many real-world decisions. As the importance of the data increases, so does the number of parties who might wish to disrupt, corrupt or eavesdrop on it.

The implementation of secure protocols is therefore one of the bottlenecks that can hold back wider IoT technology adoption. Building an understanding of the requirements that these protocols have is therefore a crucial step in enabling this technology. Our paper contributes to this by providing two novel insights:

(1) We perform a comparison between different key establishment schemes, based both on hardware and software key exchange implementations.
(2) We provide quantitative data on the energy and time requirements of implementing Diffie-Hellman-based key agreement on previously unevaluated IoT hardware.

To improve the applicability of this research we do this analysis using real devices connected to real wireless networks. Our findings are therefore based on a whole-stack implementation, which might not be the case if derived from simulated results.

2 RELATED WORK

There have been a number of studies comparing the performance and resource requirements of implementing cryptography on IoT-style constrained devices. However, the range of available protocols, algorithms, approaches and hardware means that there is still much work to be done in this field.

Noseda et al. [18] presents a comparison between five different secure IoT platforms, evaluating their resource consumption (energy usage and execution time) when undertaking a number of secure channel initiation steps, termed cryptographic primitives. They demonstrate the impact that appropriate hardware selection can have on device battery life, however they put out of scope the key-exchange element that is the subject of our study.

Kietzmann et al. [11] provide a similar bench-marking report, comparing the cryptographic primitive execution of five different pieces of IoT hardware, this time including memory requirement in their list of performance matrices. In this analysis, however, they do not include key exchange protocols integrated in a full networking stack, relying instead on previously exchanged keys. Kietzmann et al. [12] then present a further study, looking in greater detail at the mechanism for pseudo-random number generation, and comparing different approaches.

Ledwaba et al. [13] undertakes a thorough investigation into the Cortex-M series processors, evaluating this hardware's suitability when it comes to supporting cryptography mechanisms.

Further hardware comparison is given by Pearsons et al. [20], this time focusing on Espressif's ESP32 and TI's CC3220, demonstrating the advantages of implementing a cryptographic co-processor (the ATECC608A) alongside the main microprocessor unit.

Nofal et al. [16] employ the EMPIOT IoT energy evaluation platform [6] to assess a number of commonly used cryptographic algorithms. By annotating the source-code of these algorithms, and running a number of empirical measurements, they are able to generate resource consumption metrics for the different building blocks that make up the handshake and record layer algorithms.

Mössinger et al. [15] demonstrate some of the resource requirements (in the form of run time, memory and energy consumption) for performing Elliptic Curve Cryptography (ECC) signatures on the TI CC2538 chipset.

3 EXPERIMENT OVERVIEW

Our results are derived from an empirical investigation that has been organized in the following main tasks:

(1) Design of key exchange protocols, one based on key transport and one based on key agreement.
(2) Selection of the IoT devices, executing the designed key exchange protocols and their implementation.
(3) Preparation of the testbed, consisting of the measuring devices and circuitry required.
(4) Data collection and preparation of tools for the analysis of the information extracted from the experiments.

The experiment aims to derive a numerical comparison of key exchange protocols, using energy comparison, execution time and memory usage as metrics. We want to measure system performance of protocols when executed on a real environment. For our purpose, this consists of hardware boards, executing typical IoT networking applications. The network is made of a series of IoT resource constrained boards, communicating with a device acting as network coordinator. We do not include other components, such as backend or proxy servers, as this would add complexity to our setup, without affecting the metrics we are interested in. We have deployed those protocols on real IoT devices, executing a standard complete network stack, from physical to application layer. A number of different open source network stack implementations exist, in our work we have chosen Contiki-ng [19].

We have designed a series of test cases and each test case has been executed 20 times. The hardware infrastructure used to perform the measurements relies on an experimental setup where by measurements are obtained by connecting the device under test to an oscilloscope, via dedicated external circuitry. Finally, numerical results have been analysed, allowing us to compare the cost of key transport and key agreement protocols, and to derive a series of conclusions that can be used as guidelines when starting to design a secure IoT network. The rest of this paper is organised as follow: in Section 4 we describe the design of the key exchange strategies. In Section 5 we provide a high-level description of what we will use as our IoT, resource constrained devices. In Section 6, we describe the implementation of the designed protocols. In Section 7, the hardware setup of the experiments is reported while in Section 8 we present the data that was collected, as well as providing analysis of the results.

4 KEY EXCHANGE DESIGN

Key exchange protocols involve two entities trying to establish a common shared secret over a channel exposed to eavesdropping, such as a wireless network [4]. We consider two main families of protocols, one based on key transport and one based on key agreement. The two parties involved in the key establishment are the Endpoint, a resource-constrained device requesting the security key, and a second device acting as the authorising entity, that we will call the 'Edge'. Key exchange algorithms have been designed to run on resource constrained devices (with more detail given in Section 5), and this puts requirements on the algorithm's characteristics. In

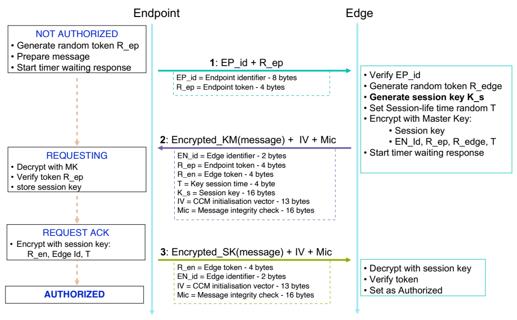

Endpoint
Edge
NOT AUTHORIZED
• Generate random token R_ep
• Prepare message
• Start timer waiting response
1: EP_id + R_ep
EP_id = Endpoint identifier - 8 bytes
R_ep = Endpoint token - 4 bytes
• Verify EP_id
• Generate random token R_edge
• Generate session key K_s
• Set Session-life time random T
• Encrypt with Master Key:
• Session key
• EN_Id, R_ep, R_edge, T
• Start timer waiting response
2: Encrypted_KM(message) + IV + Mic
EN_id = Edge identifier - 2 bytes
R_ep = Endpoint token - 4 bytes
R_en = Edge token - 4 bytes
T = Key session time - 4 byte
K_s = Session key - 16 bytes
IV = CCM initialisation vector - 13 bytes
Mic = Message integrity check - 16 bytes
REQUESTING
• Decrypt with MK
• Verify token R_ep
• store session key
REQUEST ACK
• Encrypt with session key:
R_en, Edge Id, T
3: Encrypted_SK(message) + IV + Mic
R_en = Edge token - 4 bytes
EN_id = Edge identifier - 2 bytes
IV = CCM initialisation vector - 13 bytes
Mic = Message integrity check - 16 bytes
• Decrypt with session key
• Verify token
• Set as Authorized
AUTHORIZED

Figure 1: Key transport

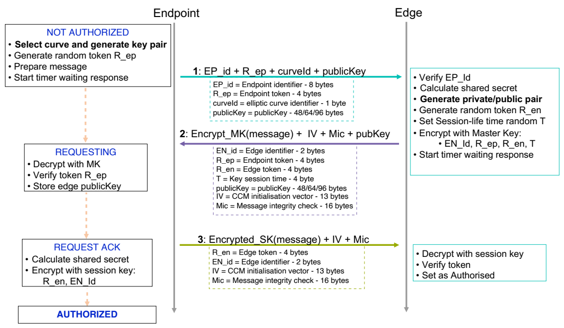

Endpoint
Edge
NOT AUTHORIZED
• Select curve and generate key pair
• Generate random token R_ep
• Prepare message
• Start timer waiting response
1: EP_id + R_ep + curveId + publicKey
EP_id = Endpoint identifier - 8 bytes
R_ep = Endpoint token - 4 bytes
curveId = elliptic curve identifier - 1 byte
publicKey = publicKey - 48/64/96 bytes
• Verify EP_Id
• Calculate shared secret
• Generate private/public pair
• Generate random token R_en
• Set Session-life time random T
• Encrypt with Master Key:
• EN_Id, R_ep, R_en, T
• Start timer waiting response
2: Encrypt_MK(message) + IV + Mic + pubKey
EN_id = Edge identifier - 2 bytes
R_ep = Endpoint token - 4 bytes
R_en = Edge token - 4 bytes
T = Key session time - 4 byte
publicKey = publicKey - 48/64/96 bytes
IV = CCM initialisation vector - 13 bytes
Mic = Message integrity check - 16 bytes
REQUESTING
• Decrypt with MK
• Verify token R_ep
• Store edge publicKey
REQUEST ACK
• Calculate shared secret
• Encrypt with session key:
R_en, EN_Id
3: Encrypted_SK(message) + IV + Mic
R_en = Edge token - 4 bytes
EN_id = Edge identifier - 2 bytes
IV = CCM initialisation vector - 13 bytes
Mic = Message integrity check - 16 bytes
• Decrypt with session key
• Verify token
• Set as Authorised
AUTHORIZED

Figure 2: Key agreement

a key transport scenario, the authorising entity, usually equip-ped with a true random number generator [3], is able to generate a random string, that will be sent to the endpoint in an encrypted message. This random string is used as a symmetric key to encrypt the channel. In the case of key agreement, we use a protocol based on elliptic curve Diffie-Hellman [1], where both parties generate a private-public key pair of values, exploiting the properties of elliptic curves. The two parties exchange the public key and use it to derive a common shared secret. Differently from the key transport case, this approach does not require the session key to be sent across the network, improving the link security. Although beneficial from a security perspective, we will show how this form of key agreement is more expensive in terms of energy consumption, execution time and memory usage, but we also show how dedicated hardware acceleration can improve the performance.

For both key transport and key agreement protocols the endpoint is authenticated: the authentication relies on an identical master key being pre-deployed on the Endpoint and the Edge. The Edge is also provided with a list of Endpoints identifiers. The Endpoint generates a random token and sends an initial unencrypted message, specifying the Endpoint identifier and a generated token. The Edge verifies the validity of the identifier and sends back a new message, encrypted using the pre-deployed master key. The Endpoint receives and decrypts the message, verifying the correctness of the token received back and storing the new session key. The diagrams in Fig. 1 and Fig. 2 illustrate in more detail the session key generation in the case of transport and agreement.

5 RESOURCE CONSTRAINED DEVICES

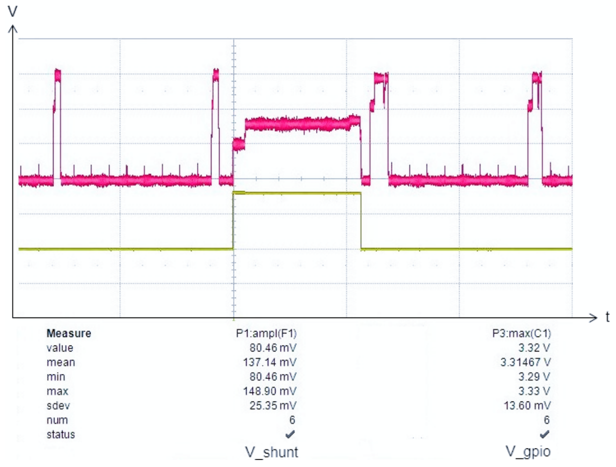
The devices used for the performance evaluation are typically defined as resource constrained, these are identified as the Endpoint in Fig. 1 and Fig. 2. The main features that these devices have in common are their limited flash/RAM memory and processing capabilities. Some of these devices may be equipped with security hardware acceleration, relieving the main processing unit of computationally heavy tasks. Another common peculiarity is their battery-sourced nature that usually restricts the average processing time: low level sleep mode APIs are available and executed periodically, as soon as the device completes all the scheduled tasks. Finally, because of these limitations the data rate available is restricted.

The IoT devices selected for these experiments are the nRF8240-DK board and the TI LAUNCHXL-CC2650. The main reasons for this choice is the rich availability of security software and hardware backends and frontends. Both boards have a set of digital IO pins (GPIO), and a 2.4GHz radio that can be used to deploy low-power networking solutions, such as 802.15.4, Thread and Bluetooth low-energy. Looking in more detail at the two devices used, the nRF52840 board relies on a 64 MHz 32-bit processor, 1 MB flash and 256 kB RAM memory, and is equipped with hardware acceleration for executing both AES encryption and ECC calculation. The TI CC2650, is provided with a 48 MHz 32-bit processor, 128 kB flash and 20 kB RAM memory. Its security hardware acceleration allows us to execute AES encryption but does not provide ECC calculation. Both the boards are equipped with a 2.4 GHz RF transceiver, used in our case to implement the physical and MAC layers of the 802.15.4 standard. The achievable bandwidth is in the order or 250 kbps.

6 KEY EXCHANGE IMPLEMENTATION

Contiki-ng is an operating system designed for wireless low-power constrained embedded devices. It provides an implementation of an IEEE 802.15.4 [8] based network. In our experiments, for the Medium Access Control (MAC) layer, TSCH protocol was used as it has previously demonstrated high reliability and energy efficiency [9], although there is a small price to pay in terms of required memory when compared to alternatives, such as CSMA. With regard to the Network Layer, the 6LoWPAN protocol was implemented. Fragmentation was enabled on the nRF52840 board, but disabled on the CC2650 due to the smaller memory available. For the Transport Layer UDP has been used, again because it has lower complexity and a smaller memory requirements when compared to TCP. The micro IP (uIP) buffer size was left at the default 1280 bytes for the nRF52840, while it was reduced to 140 bytes for the CC2650. The network configuration selected for the CC2650 introduced a limitation in the maximum UDP payload size, set at 64 bytes. This limitation required us to implement both message 1 and 3 in Fig. 2 in two separate messages.

The logic of the key exchange algorithms has been implemented as a dedicated Contiki-ng proto-thread [5]. The implementation is organized into two parts, one platform-independent part allowing for straightforward operation across different hardware types, and one platform-specific part calling on the security, networking and digital IO services provided by the board. For the nRF52850, platform-specific services are provided by nRF52-SDK 16.0.0. For the CC2650 we used TI-RTOS 2.21.01.08. The key transport algorithm requires a true random number generator (TRNG) service and



Measure	P1:ampl(F1)	P3:max(C1)
value	80.46 mV	3.32 V
mean	137.14 mV	3.31467 V
min	80.46 mV	3.29 V
max	148.90 mV	3.33 V
sdev	25.35 mV	13.60 mV
num	6	6
status	✓	✓
	V_shunt	V_gpio

Figure 3: Voltage measured and digital event

the AES encryption service. The key agreement algorithm requires these, as well as the elliptic curve cryptography library. Hardware acceleration for TRNG and AES is available for both nRF52840 and CC2650, while ECDH is available only on nRF52840. We implemented both hardware and software versions of ECDH, using the backend CC3310 [10] and MBED-TLS [14] respectively.

The Edge device is deployed on a laptop running a Linux Ubuntu OS. A nRF52840 dongle is attached to the laptop via USB, providing 802.15.4 radio capabilities. In this configuration, the dongle acts as border-router and provides a network interface to the OS. The protocols logic was implemented in Python using the *tinyEC* Python package for the elliptic curves operations.

7 TESTBED DESCRIPTION

To capture the power consumption and execution time of the IoT devices under test an oscilloscope based measurement setup was deployed. Fig. 4 shows the experimental set up, including the IoT hardware under test and the measurement circuitry. The oscilloscope used in the experiments is a Teledyne Lecroy HDO4104 series with the capability of capturing 2.5GS/s. To collect consistent measurements, both nRF52840-DK and LAUNCHXL-CC2650 IoT devices were supplied externally by the same AIM-TTI MX100TP Multi-Range DC Power Supply. The shunt resistor in the setup is used to measure a voltage drop, from which the current drawn from the DC power supply can be calculated. A low value (10 Ω) was selected for the shunt resistor to limit the voltage drop across this component, making it small when compared with the voltage drop across the device under test. The resistance of the component was tested empirically to avoid inaccuracies in the calculated values. To eliminate completely the effects of the shunt resistor from our results we used only the voltage across the device under test (V) to calculate the energy consumed by the constrained device, and subtract the voltage drop on the shunt resistor (V_{shunt}).

A voltage of 3.3V DC was applied to the testbed circuit, as this is within the rated supply voltage range of both the nRF52840-DK [17] and LAUNCHXL-CC2650 [22] boards. The current through the shunt resistor (I_{shunt}), and therefore through the device under test (I), was calculated by dividing the instantaneous voltage across that

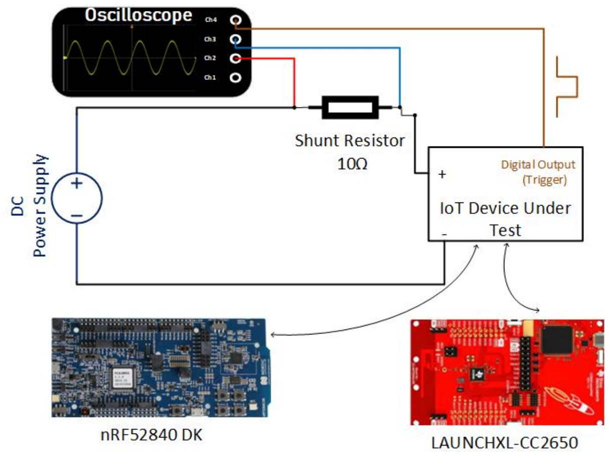


Figure 4: Measurement Setup with Oscilloscope

Table 1: Energy consumption and Execution time

Scenario	Board	Energy (mJ)	Exec time (ms)
Transport	nRF52840 HW	0.023±0.001	0.953±0.016
secp192r1	nRF52840 HW	0.709±0.001	21.86±0.070
secp256r1	nRF52840 HW	1.348±0.001	40.57±0.068
secp384r1	nRF52840 HW	3.870±0.035	111.7±0.519
secp192r1	nRF52840 SW	16.54±0.163	535.6±4.060
secp256r1	nRF52840 SW	32.82±0.264	1070.5±9.309
secp384r1	nRF52840 SW	58.70±0.652	1876.3±26.28
Transport	cc2650 HW	0.051±0.002	1.131±0.021
secp192r1	cc2650 SW	48.55±1.239	696.5±20.58
secp256r1	cc2650 SW	87.93±0.366	1280.5±5.66
secp384r1	cc2650 SW	140.8±1.012	1964.5±18.40

component (V_{shunt}) by the shunt resistance (R_{shunt}) value (eq. (1)).

$$I_{shunt} = \frac{V_{shunt}}{R_{shunt}} = I \tag{1}$$

The instantaneous power consumed by the device under test can then be calculated by multiplying V and I.

A number of the device under tests digital "GPIO" pins were also connected to the oscilloscope. These GPIO pins were configured in software to be set high when the key transport/agreement operation started, and then low again when the operation was completed. Thus, the oscilloscope recording was triggered to co-inside with the key transport and key agreement messages being exchanged. The recorded set of instantaneous power measurements were multiplied by the oscilloscope sample period, and summed to give the total energy consumed during the experiment (E). In eq. (2) ω denotes the oscilloscope sample frequency. The GPIO output is set high at sample $n = 0$, denoting the start of the sampling window, and then set low at sample $n = N$ denoting the end.

$$E = \frac{1}{\omega} \sum_{n=0}^{N} V_n I_n \tag{2}$$

Where a scenario requires multiple distinct windows, such as the sending of multiple messages, then only those samples that co-inside with a window are summed, and those samples between windows are not included.

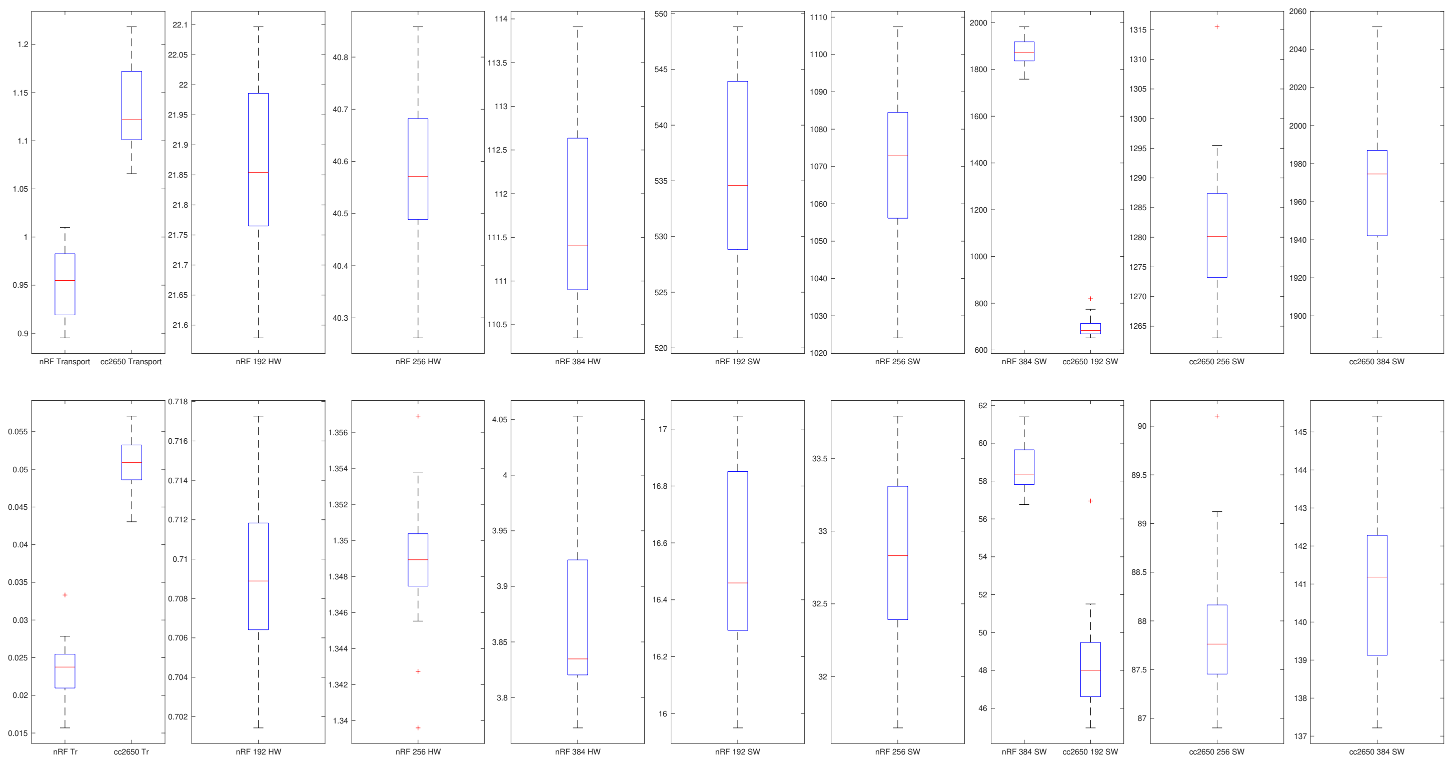


Figure 5: Box plots for the Execution Time (top, values in ms) and Energy consumption (bottom, values in mJ)

8 RESULTS AND ANALYSIS

In this section, we present and compare the results gathered during the experiments described above. We provide analysis of the execution time, energy consumption and memory usage for each of the protocols running on the two devices under test. Each experiment was repeated 20 times, and in Table 1 the mean value for energy consumption and execution time is presented. Every mean value is reported with the related confidence interval, calculated using a confidence level of 95% and a number of samples equal to 20. A visual representation of the statistical content of each experiment is shown with box plots in Fig. 5.

8.1 Execution time

To calculate key exchange protocol execution times, digital GPIO output signals were employed, as they operate quickly compared to the execution time of the algorithm. A digital event, captured with the oscilloscope, is shown in Fig. 3, where we can see the digital output signal in yellow and the recorded shunt resistor voltage in pink. Referring to Fig. 1 and Fig. 2, three events are considered to sum up the total execution time. The software sets high the board digital output signal before starting to execute the algorithm related instructions, and low when the message is delivered to the UDP service of the operating system. Similarly, when receiving a message, the digital output is set high only when data are available at the Application Layer and the algorithm starts to be executed. In this way we do not include in the time window of interest all the lower level networking activity. Results are illustrated in Table 1. We observe how the hardware acceleration can reduce the execution time compared to the software implementation and how the size of the key seems to affect significantly the computation time. In real-time systems, time allocation can be critical, and executing tasks requiring several hundreds of milliseconds may not be feasible.

8.2 Energy consumption

As can be observed in Table 1, the energy consumption increases as the protocol bit length is increased, and this is true across all devices, and hardware or software implementations. Comparing the software implementations on the two devices we see that the energy consumption on the two devices is of the same order of magnitude. However, comparing the performance of the hardware and software implementations on the nRF device shows a much starker difference. As can be observed in the Table 1, energy consumption is significantly reduced for the hardware implementation. When using the hardware support, the nRF device performs approximately 23 times better at ECDH with 192-bit and 256-bit key sizes, and 15 times better at ECDH with 384-bit key size.

Avoiding the key agreement process altogether, however, provides even greater efficiency. Key transport algorithms deployed on nRF52840 outperformed the hardware key agreement process by approximately 31 times for ECC secp192r1, 59 times for ECC secp256r1 and 167 times for ECC secp384r1. The key transport protocol surpasses the software implementation of ECC curve by 719, 1426 and 2545 times respectively when the numerical results in Table 1 are taken into account.

The CC2650 board performs similarly to the nRF52840 board on the software implementation of ECC curve when it is compared to

Table 2: Memory report from static analysis (values in bytes)

Scenario	Board	text	data	bss
Transport	nRF52840 HW	80880	1012	14852
secp192r1	nRF52840 HW	101720	1012	17468
secp256r1	nRF52840 HW	101720	1012	17468
secp384r1	nRF52840 HW	101720	1012	17468
secp192r1	nRF52840 SW	115416	1016	46976
secp256r1	nRF52840 SW	116056	1016	46976
secp384r1	nRF52840 SW	116448	1016	46976
Transport	cc2650 HW	74540	1331	12052
secp192r1	cc2650 SW	93276	3455	12344
secp256r1	cc2650 SW	94116	3455	12344
secp384r1	cc2650 SW	97096	3455	12344
All curves	nRF52840 SW	125080	1040	46976

the transport protocol. The CC2650 board consumes less energy with the transport protocol by 951, 1723 and 2747 times compared to ECC secp192r1, ECC secp256r1 and ECC secp384r1 software deployments respectively. It should be highlighted that the CC2650 and nRF52840 boards have quite different specifications, and this should be considered when making comparisons between the two devices.

8.3 Memory

A static memory analysis has been conducted using the GNU size utility [7]. Results are reported in Table 2. The first column indicates the test scenario, the other columns report the size of the segments in which the memory is organized. Segment 'text' includes executable instructions and constant variables, and is placed in the Flash memory. Segment 'data' includes initialized global and static variables and is placed in RAM. Segment 'bss' includes non initialized variables and is also placed in RAM.

Considering the Flash and RAM memory information presented in Section 5, Transport protocol is an affordable solution for both nRF52840 and CC2650. When implementing key agreement, we enabled only one elliptic curve at compile time. This is true for all scenarios in table except the last, named 'nrf SW All curves', where all curves are included, although this was not possible on the CC2650 due to memory limitations. We observe also in the software implementation case, the text segment increases when increasing the curve size. The same does not happen for the hardware implementation, as the curves static definition is placed in dedicated hardware, without using Flash or RAM memory. Finally, comparing nRF52840 and CC2650 text segments, we observe smaller values in the second board. This is due to a lighter version of the networking stack, as described in Section 6.

9 CONCLUSION AND FUTURE WORK

Efficient key exchange protocols allow for the frequent regeneration of a secure shared secret. We have conducted an empirical study, considering protocols based on Key Transport and Key Agreement approach. The details of the protocols has been described, as well as implementation on two IoT boards. We have illustrated the testbed used to measure the performance and the methodology applied to process the acquired data. Numerical results have been presented and analyzed. Future activity will consist of evaluating a wider range of security key size, validation of our results using alternative test beds, and deployment on different IoT boards using alternative real-time operating systems.

ACKNOWLEDGMENTS

This work was supported by UK Research and Innovation, Innovate UK [grant number 53707]

REFERENCES

[1] Stephen Aikins-Bekoe, Kwame Nkrumah, and James B. Hayfron-Acquah. 2020. Elliptic Curve, Diffie-Hellman, Wireless Sensor Network, Public key, Encryption, Decryption.

[2] Ali Ismail Awad, Steven Furnell, Abbas M. Hassan, and Theo Tryfonas. 2019. Special issue on security of IoT-enabled infrastructures in smart cities. *Ad Hoc Networks* 92 (2019), 101850. Special Issue on Security of IoT-enabled Infrastructures in Smart Cities.

[3] Boaz Barak, Ronen Shaltiel, and Eran Tromer. 2003. True Random Number Generators Secure in a Changing Environment. In *Cryptographic Hardware and Embedded Systems - CHES 2003*, Colin D. Walter, Çetin K. Koç, and Christof Paar (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 166–180.

[4] Ran Canetti and Hugo Krawczyk. 2001. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *Advances in Cryptology — EUROCRYPT 2001*, Birgit Pfitzmann (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 453–474.

[5] Contiki-NG. 2021. Protothreads. https://contiki-ng.readthedocs.io/en/master/_api/group__pt.html#details. [Online; accessed 30-May-2022].

[6] Behnam Dezfouli, Immanuel Amirtharaj, and Chia-Chi (Chelsey) Li. 2018. EMPIOT: An energy measurement platform for wireless IoT devices. *Journal of Network and Computer Applications* 121 (2018), 135–148.

[7] Free Software Foundation. 2021. size Debian manpage. https://manpages.debian.org/testing/binutils-arm-none-eabi/arm-none-eabi-size.1.en.html. [Online; accessed 01-Jun-2022].

[8] IEEE. 2016. IEEE Standard for Low-Rate Wireless Networks. *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)* (2016), 1–709.

[9] Hosni Ines. 2018. Performance of IEEE802.15.4e TSCH Protocol for Multi-hop Wireless Sensor Networks. In *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. 603–608.

[10] Nordic Semiconductor infocenter. 2022. nRF52 cc3310. https://infocenter.nordicsemi.com/index.jsp. [Online; accessed 31-May-2022].

[11] Peter Kietzmann, Lena Boeckmann, Leandro Lanzieri, Thomas C. Schmidt, and Matthias Wählisch. 2021. A Performance Study of Crypto-Hardware in the Low-End IoT. In *Proceedings of the 2021 International Conference on Embedded Wireless Systems and Networks* (Delft, The Netherlands) *(EWSN '21)*. Junction Publishing, USA, 79–90.

[12] Peter Kietzmann, Thomas C. Schmidt, and Matthias Wählisch. 2021. A Guideline on Pseudorandom Number Generation (PRNG) in the IoT. *ACM Comput. Surv.* 54, 6, Article 112 (jul 2021), 38 pages.

[13] Lehlogonolo P. I. Ledwaba, Gerhard P. Hancke, Hein S. Venter, and Sherrin J. Isaac. 2018. Performance Costs of Software Cryptography in Securing New-Generation Internet of Energy Endpoint Devices. *IEEE Access* 6 (2018), 9303–9323.

[14] ARM Limited. 2016. Mbed TLS - Core features. https://tls.mbed.org/core-features. [Online; accessed 31-May-2022].

[15] Max Mössinger, Benedikt Petschkuhn, Johannes Bauer, Ralf C. Staudemeyer, Marcin Wójcik, and Henrich C. Pöhls. 2016. Towards quantifying the cost of a secure IoT: Overhead and energy consumption of ECC signatures on an ARM-based device. In *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. 1–6.

[16] Ramzi A. Nofal, Nam Tran, Carlos Garcia, Yuhong Liu, and Behnam Dezfouli. 2019. A Comprehensive Empirical Analysis of TLS Handshake and Record Layer on IoT Platforms *(MSWIM '19)*. Association for Computing Machinery, New York, NY, USA, 61–70.

[17] Nordic Semiconductor 2021. *nRF52840*. Nordic Semiconductor. Rev. 7.

[18] Mario Noseda, Lea Zimmerli, Tobias Schläpfer, and Andreas Rüst. 2022. Performance Analysis of Secure Elements for IoT. *IoT* 3, 1 (2022), 1–28.

[19] George Oikonomou, Simon Duquennoy, Atis Elsts, Joakim Eriksson, Yasuyuki Tanaka, and Nicolas Tsiftes. 2022. The Contiki-ng open source operating system for next generation IOT devices. *SoftwareX* 18 (2022), 101089.

[20] Bryan Pearson, Lan Luo, Yue Zhang, Rajib Dey, Zhen Ling, Mostafa Bassiouni, and Xinwen Fu. 2019. On Misconception of Hardware and Cost in IoT Security and Privacy. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. 1–7.

[21] Tobias Schläpfer and Andreas Rüst. 2019. Security on IoT devices with secure elements. In *Embedded World Conference 2019 - Proceedings*. WEKA.

[22] Texas Instruments 2016. *CC2650 SimpleLink™ Multistandard Wireless MCU*. Texas Instruments. Rev. 2.